University of BRISTOL

Advanced Algorithms – COMS31900

Approximation algorithms part three

(Fully) Polynomial Time Approximation Schemes

Raphaël Clifford

Slides by Benjamin Sach

Approximation Algorithms Recap

An algorithm A is an α-approximation algorithm for problem P if,

- A runs in polynomial time
- A always outputs a solution with value s
  within an α factor of Opt

- Here P is an optimisation problem with optimal solution of value Opt

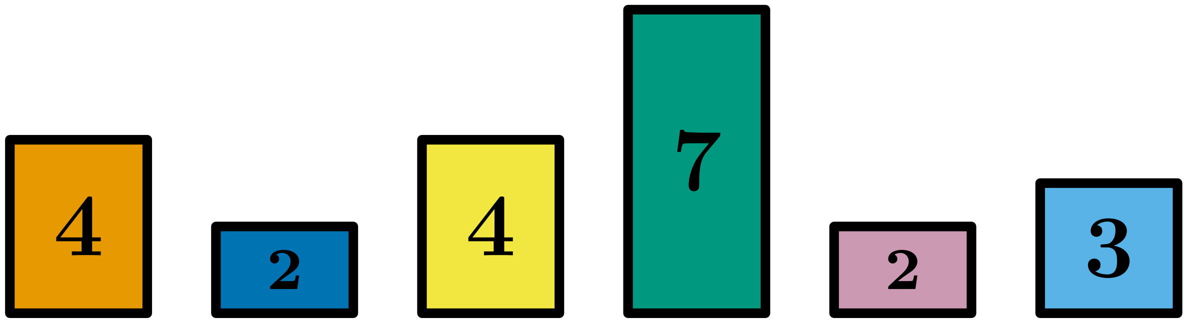
- If P is a *maximisation* problem, $\dfrac{\mathrm{Opt}}{\alpha} \leqslant s \leqslant \mathrm{Opt}$
- If P is a *minimisation* problem, $\mathrm{Opt} \leqslant s \leqslant \alpha \cdot \mathrm{Opt}$

We have seen:

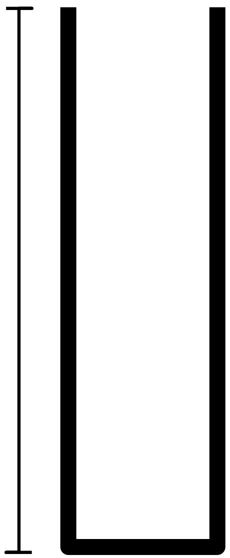
a $3/2$-approximation algorithm for Bin Packing
a $3/2$-approximation algorithm for scheduling multiple machines

a 2-approximation algorithm for k-centers

The Subset Sum problem

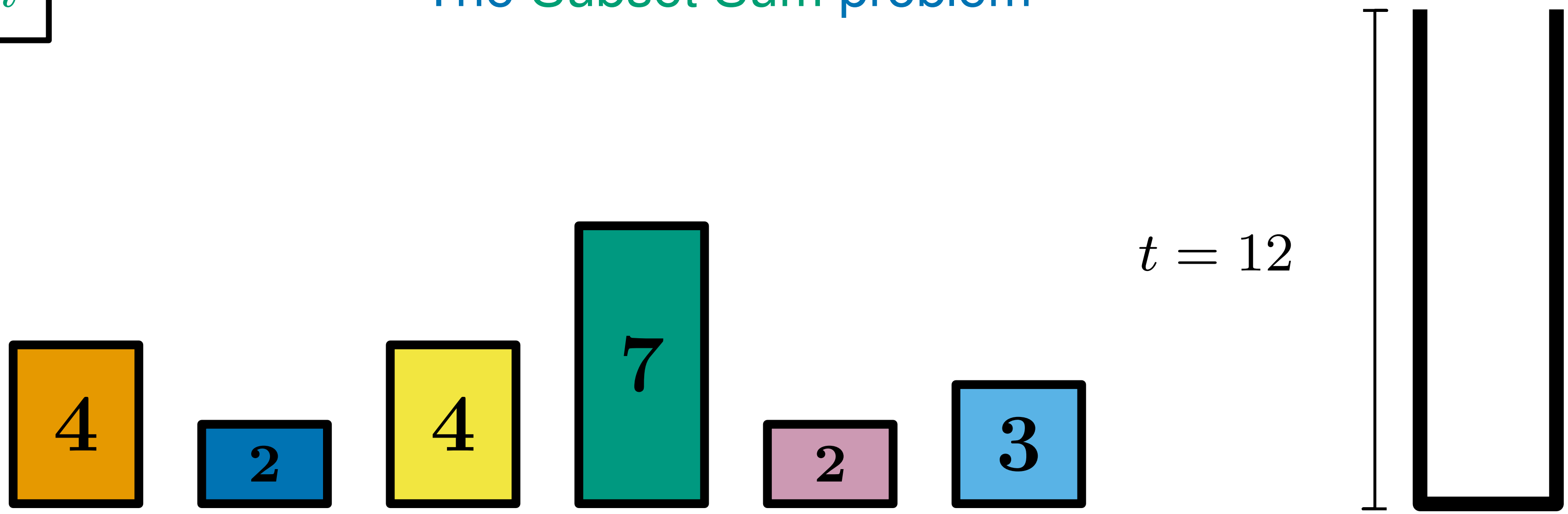


$t = 12$



$\boxed{|S| = m}$

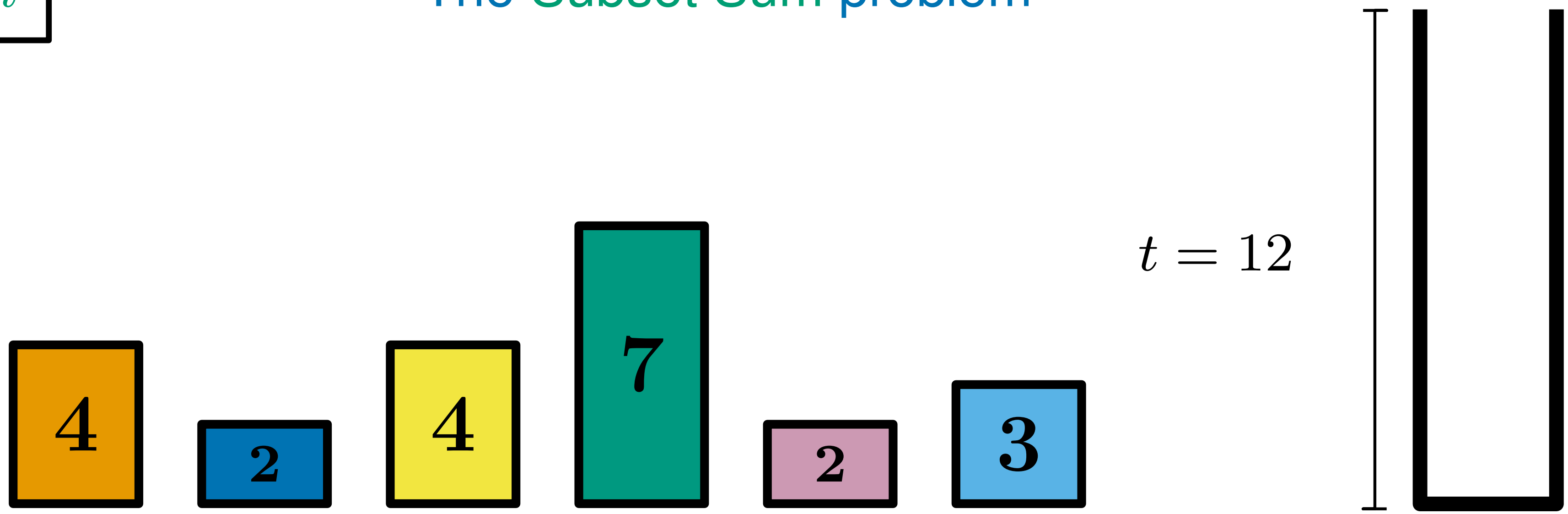
The Subset Sum problem



- Let S be a multi-set of positive integers and t be a positive integer

$\boxed{|S| = m}$

The Subset Sum problem

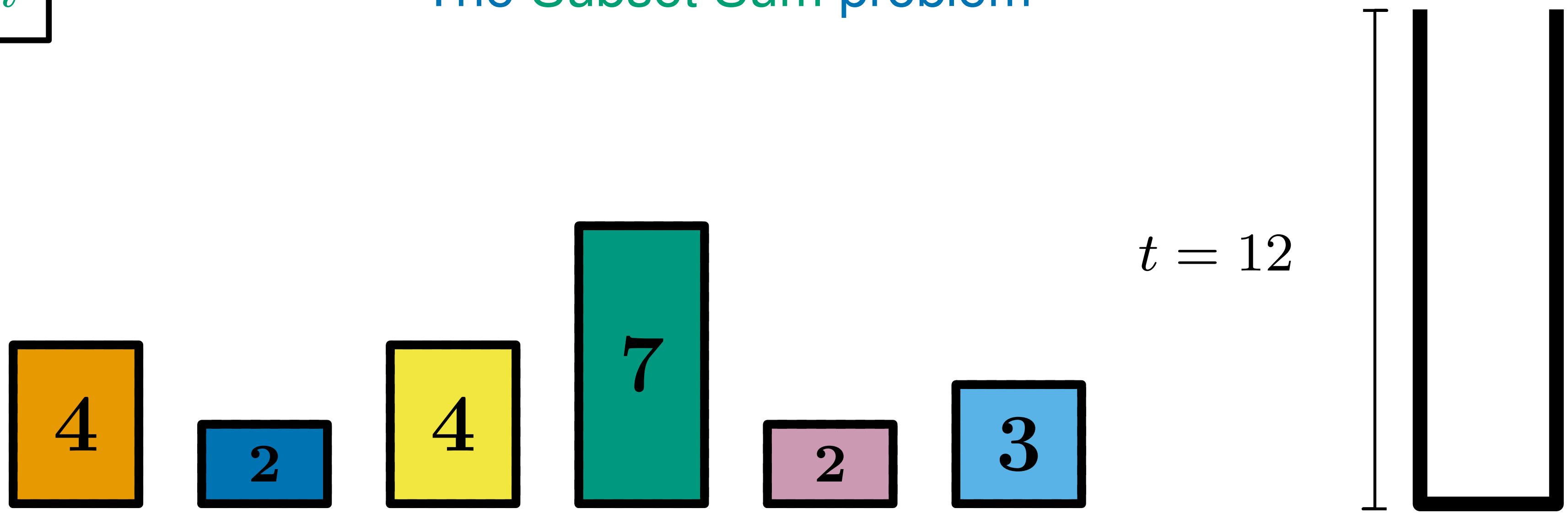


- Let S be a multi-set of positive integers and t be a positive integer

here $S = \{4, 2, 4, 7, 2, 3\}$ *and* $t = 12$

$\boxed{|S| = m}$

The Subset Sum problem



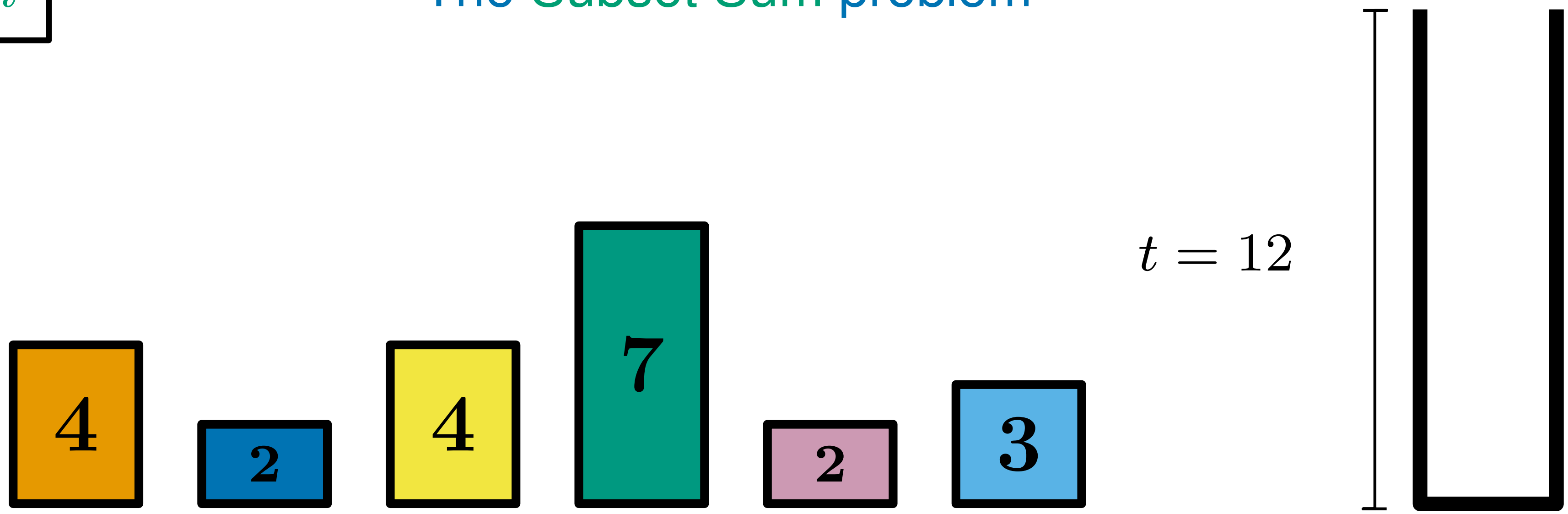
- Let S be a multi-set of positive integers and t be a positive integer

here $S = \{4, 2, 4, 7, 2, 3\}$ *and* $t = 12$

Decision Problem Is there a subset, $S' \subseteq S$ with size t?

$\boxed{|S| = m}$

The Subset Sum problem



- Let S be a multi-set of positive integers and t be a positive integer

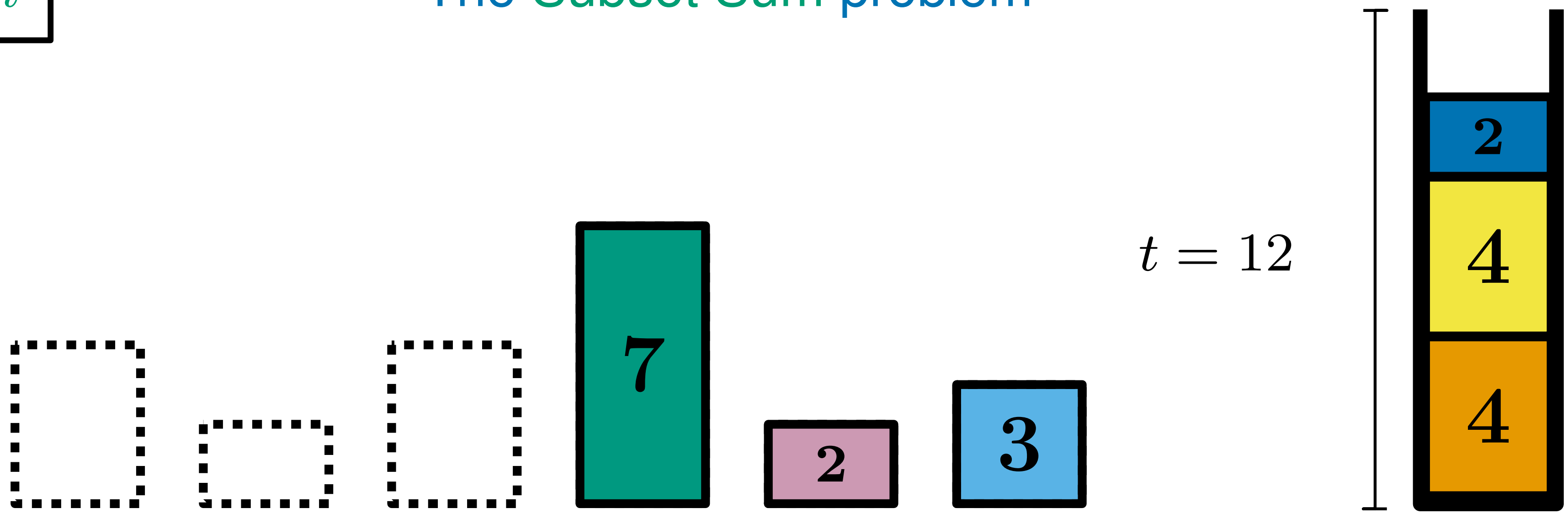
here $S = \{4, 2, 4, 7, 2, 3\}$ *and* $t = 12$

Decision Problem Is there a subset, $S' \subseteq S$ with size t?

the size of S' *is* $\sum_{a \in S'} a$

$\boxed{|S| = m}$

The Subset Sum problem



- Let S be a multi-set of positive integers and t be a positive integer

here $S = \{4, 2, 4, 7, 2, 3\}$ *and* $t = 12$

Decision Problem Is there a subset, $S' \subseteq S$ with size t?

the size of S' *is* $\sum_{a \in S'} a$

$\boxed{|S| = m}$

The Subset Sum problem

- Let S be a multi-set of positive integers and t be a positive integer

here $S = \{4, 2, 4, 7, 2, 3\}$ *and* $t = 12$

Decision Problem Is there a subset, $S' \subseteq S$ with size t?

the size of S' *is* $\sum_{a \in S'} a$

$\boxed{|S| = m}$

The Subset Sum problem

- Let S be a multi-set of positive integers and t be a positive integer

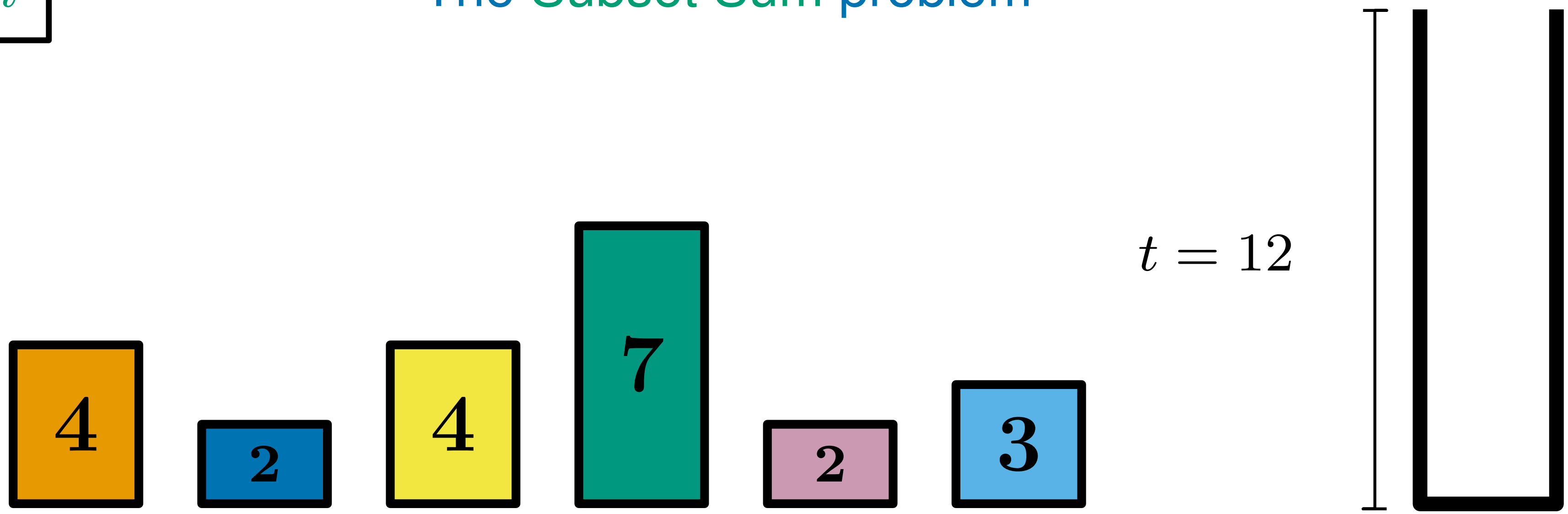
here $S = \{4, 2, 4, 7, 2, 3\}$ *and* $t = 12$

Decision Problem Is there a subset, $S' \subseteq S$ with size t?

the size of S' *is* $\sum_{a \in S'} a$

$\boxed{|S| = m}$

The Subset Sum problem



- Let S be a multi-set of positive integers and t be a positive integer

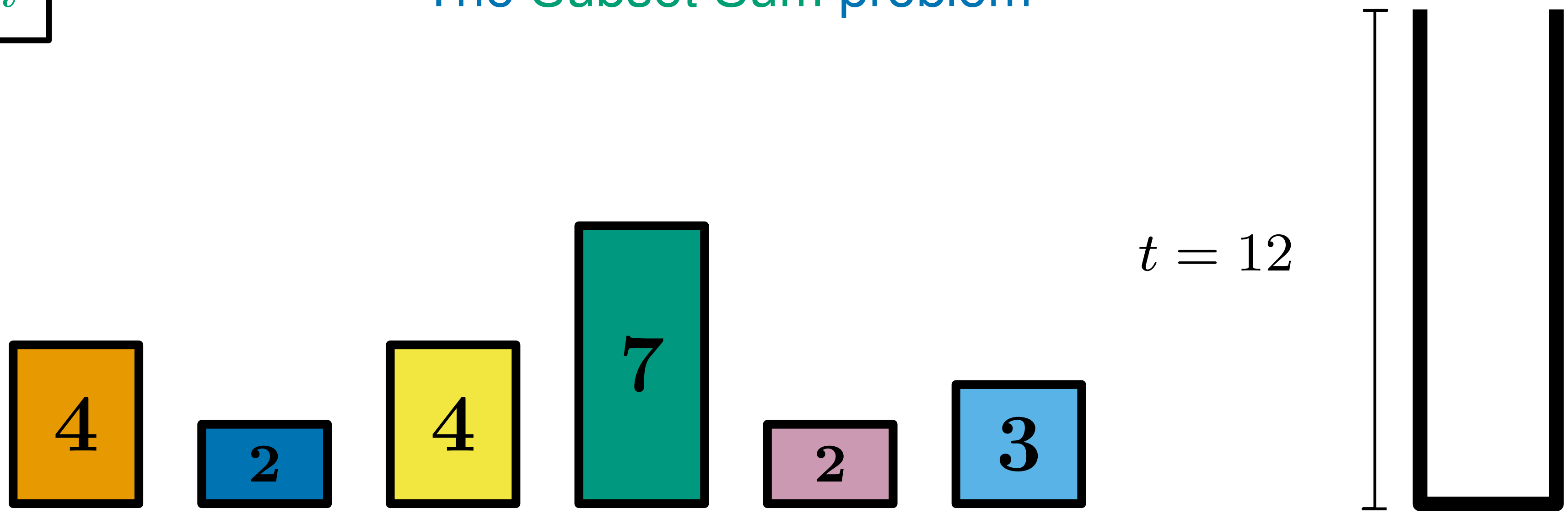
here $S = \{4, 2, 4, 7, 2, 3\}$ *and* $t = 12$

Decision Problem Is there a subset, $S' \subseteq S$ with size t?

the size of S' *is* $\sum_{a \in S'} a$

$\boxed{|S| = m}$

The Subset Sum problem



- Let S be a multi-set of positive integers and t be a positive integer

here $S = \{4, 2, 4, 7, 2, 3\}$ *and* $t = 12$

Decision Problem Is there a subset, $S' \subseteq S$ with size t?

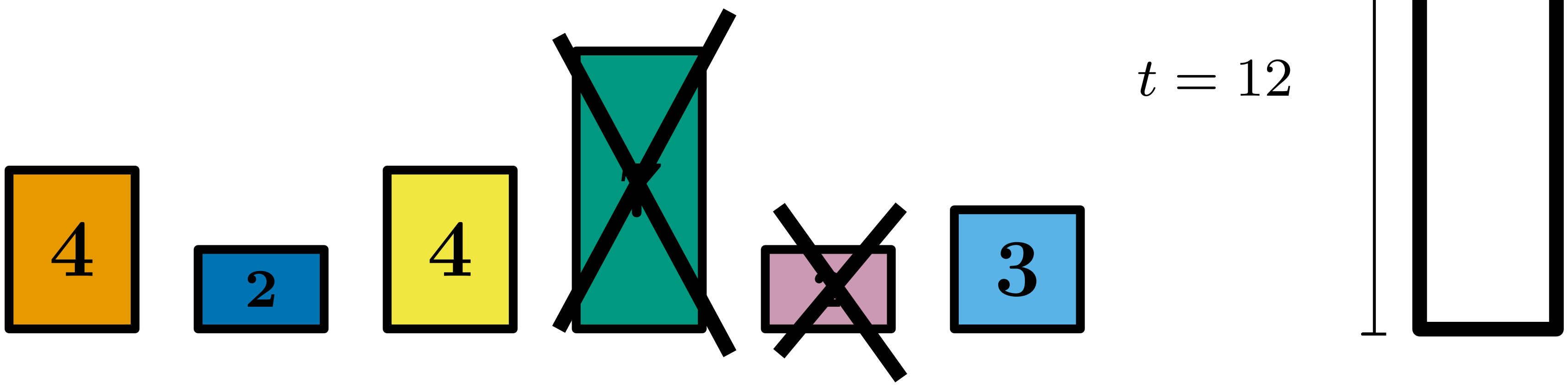
the size of S' *is* $\sum_{a \in S'} a$

Optimisation Problem

Find the size of the largest subset of S which is no larger than t

$\boxed{|S| = m}$

The Subset Sum problem



- Let S be a multi-set of positive integers and t be a positive integer

here $S = \{4, 2, 4, \cancel{7}, \cancel{2}, 3\}$ *and* $t = 12$

Decision Problem Is there a subset, $S' \subseteq S$ with size t?

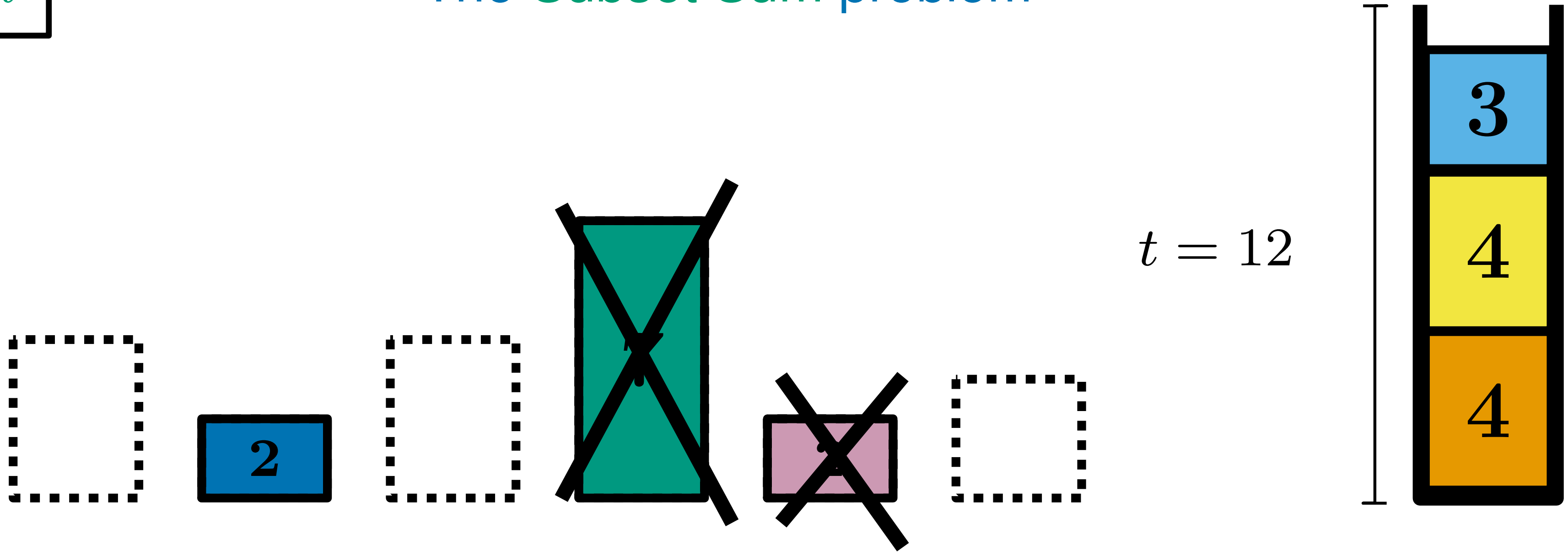
the size of S' *is* $\sum_{a \in S'} a$

Optimisation Problem

Find the size of the largest subset of S which is no larger than t

$\boxed{|S| = m}$

The Subset Sum problem



- Let S be a multi-set of positive integers and t be a positive integer

here $S = \{4, 2, 4, \cancel{7}, \cancel{2}, 3\}$ *and* $t = 12$

Decision Problem Is there a subset, $S' \subseteq S$ with size t?

the size of S' *is* $\sum_{a \in S'} a$

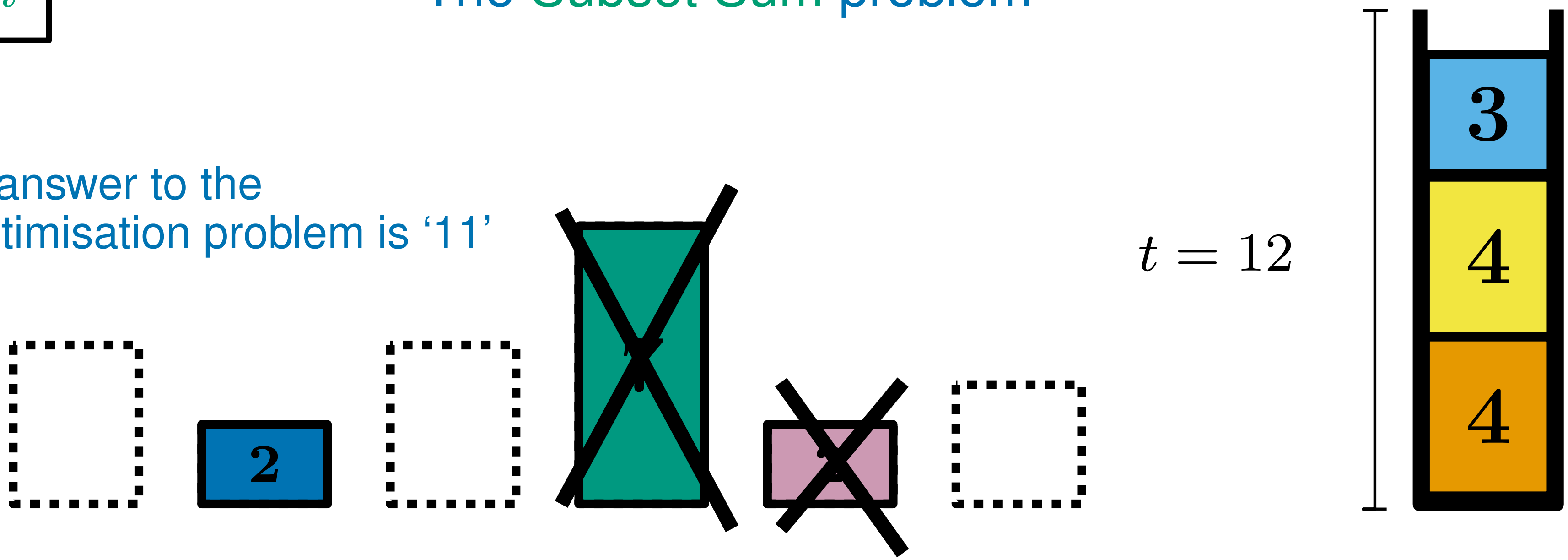
Optimisation Problem

Find the size of the largest subset of S which is no larger than t

$\boxed{|S| = m}$

The Subset Sum problem

The answer to the optimisation problem is ‘11’



- Let S be a multi-set of positive integers and t be a positive integer

here $S = \{4, 2, 4, \cancel{7}, \cancel{2}, 3\}$ *and* $t = 12$

Decision Problem Is there a subset, $S' \subseteq S$ with size t?

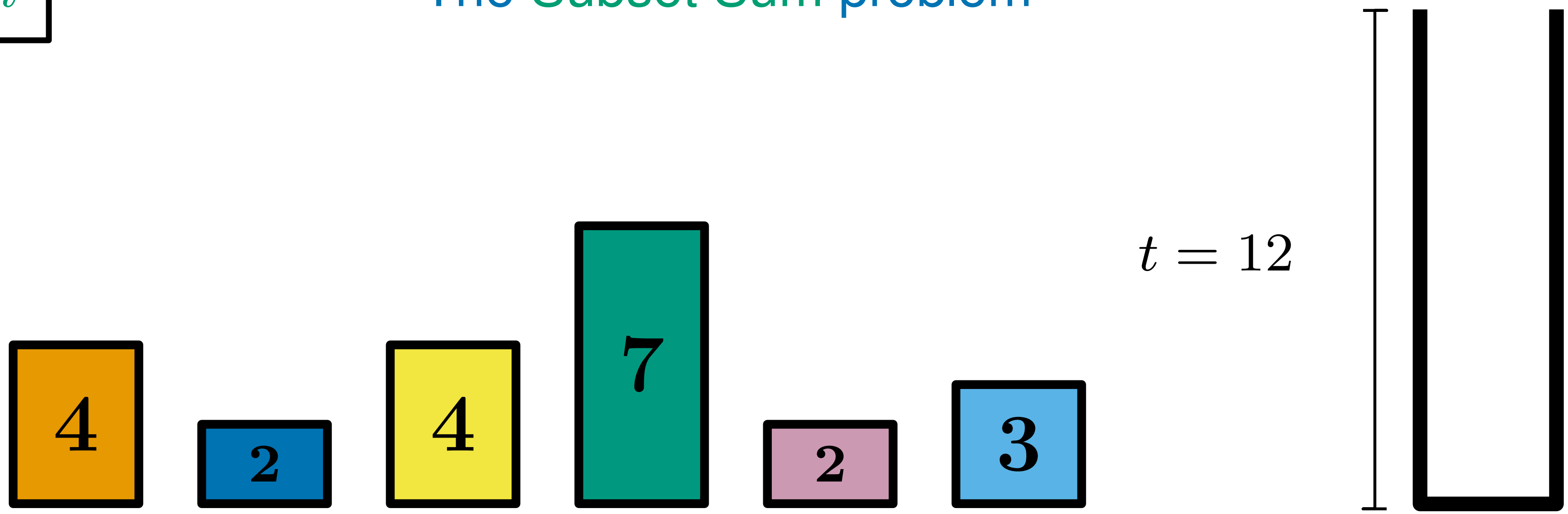
the size of S' *is* $\sum_{a \in S'} a$

Optimisation Problem

Find the size of the largest subset of S which is no larger than t

$\boxed{|S| = m}$

The Subset Sum problem



- Let S be a multi-set of positive integers and t be a positive integer

here $S = \{4, 2, 4, 7, 2, 3\}$ *and* $t = 12$

Decision Problem Is there a subset, $S' \subseteq S$ with size t?

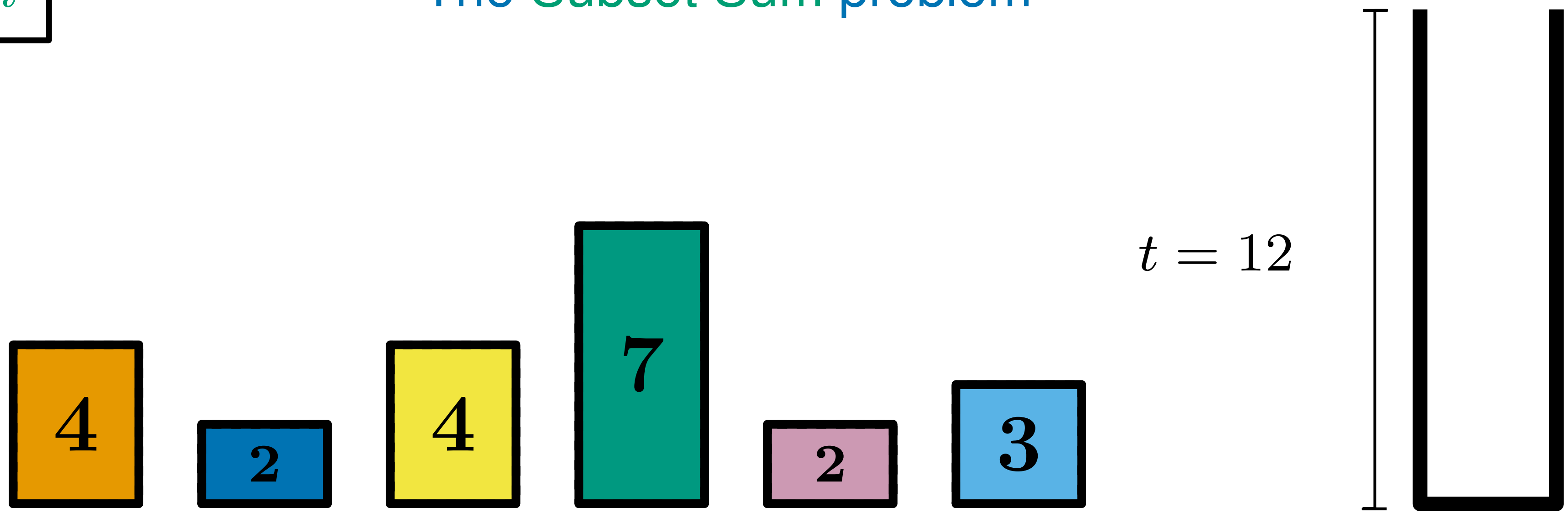
the size of S' *is* $\sum_{a \in S'} a$

Optimisation Problem

Find the size of the largest subset of S which is no larger than t

$\boxed{|S| = m}$

The Subset Sum problem



- Let S be a multi-set of positive integers and t be a positive integer

here $S = \{4, 2, 4, 7, 2, 3\}$ *and* $t = 12$

Decision Problem Is there a subset, $S' \subseteq S$ with size t?

the size of S' *is* $\sum_{a \in S'} a$

Optimisation Problem

Find the size of the largest subset of S which is no larger than t

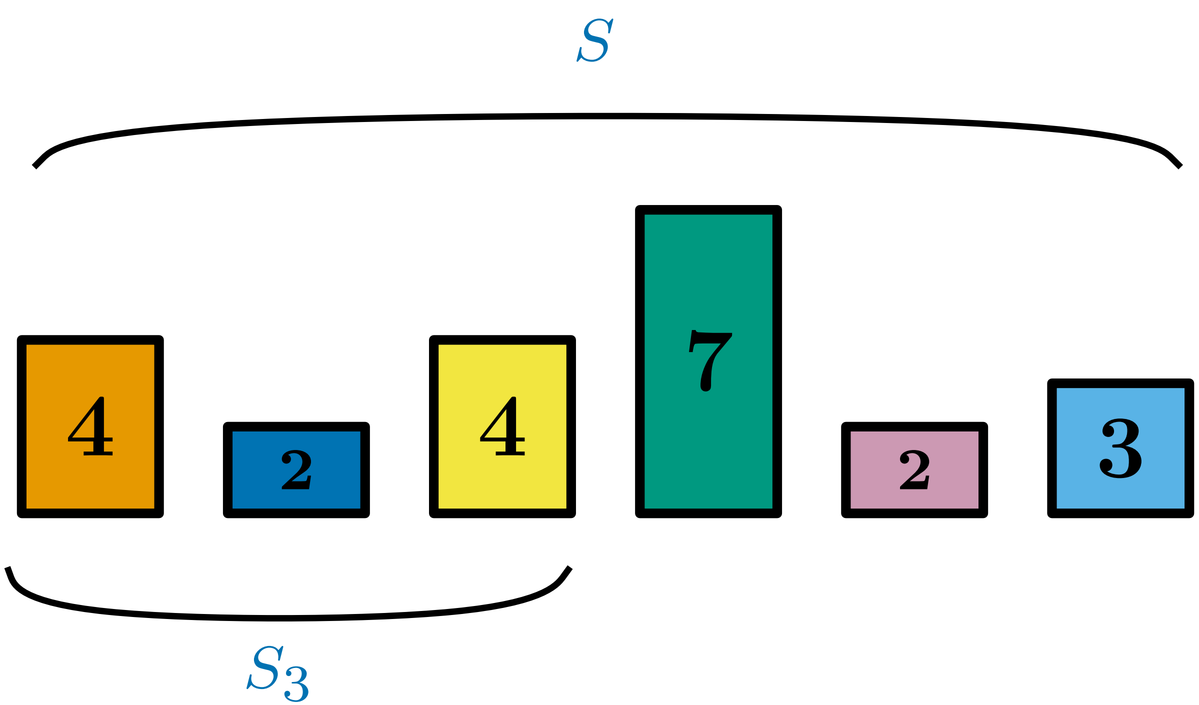
The optimisation version is NP-hard

and the decision version is NP-complete

$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

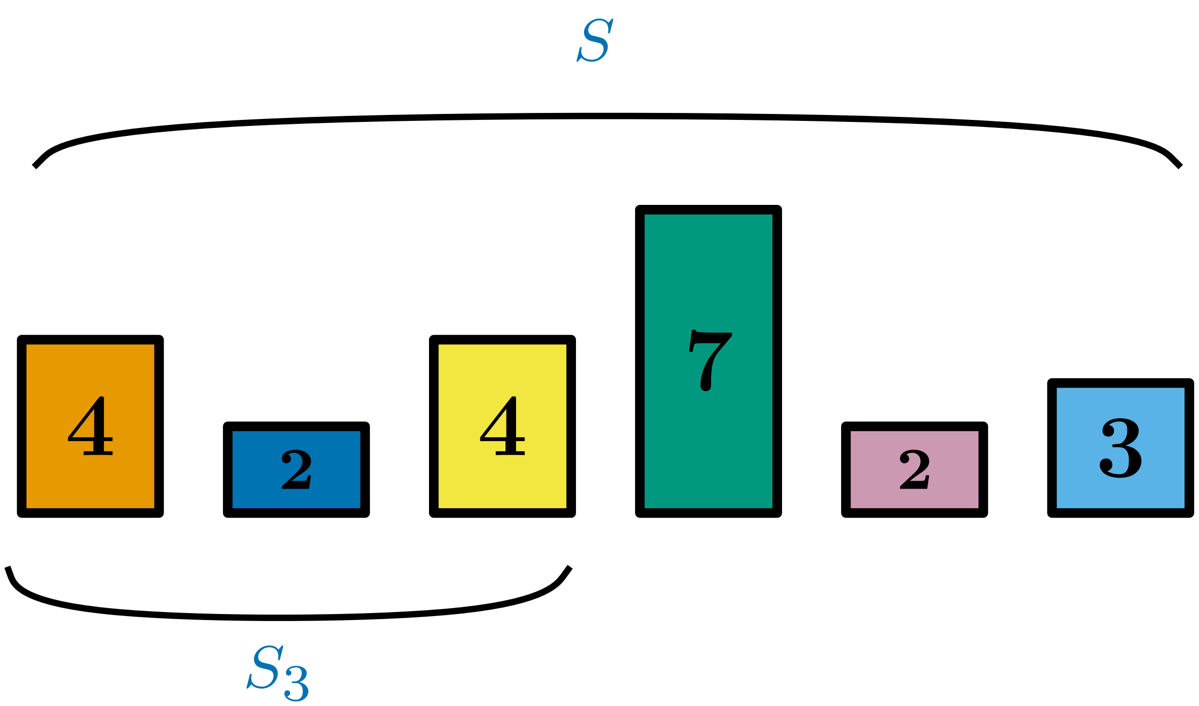


$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t



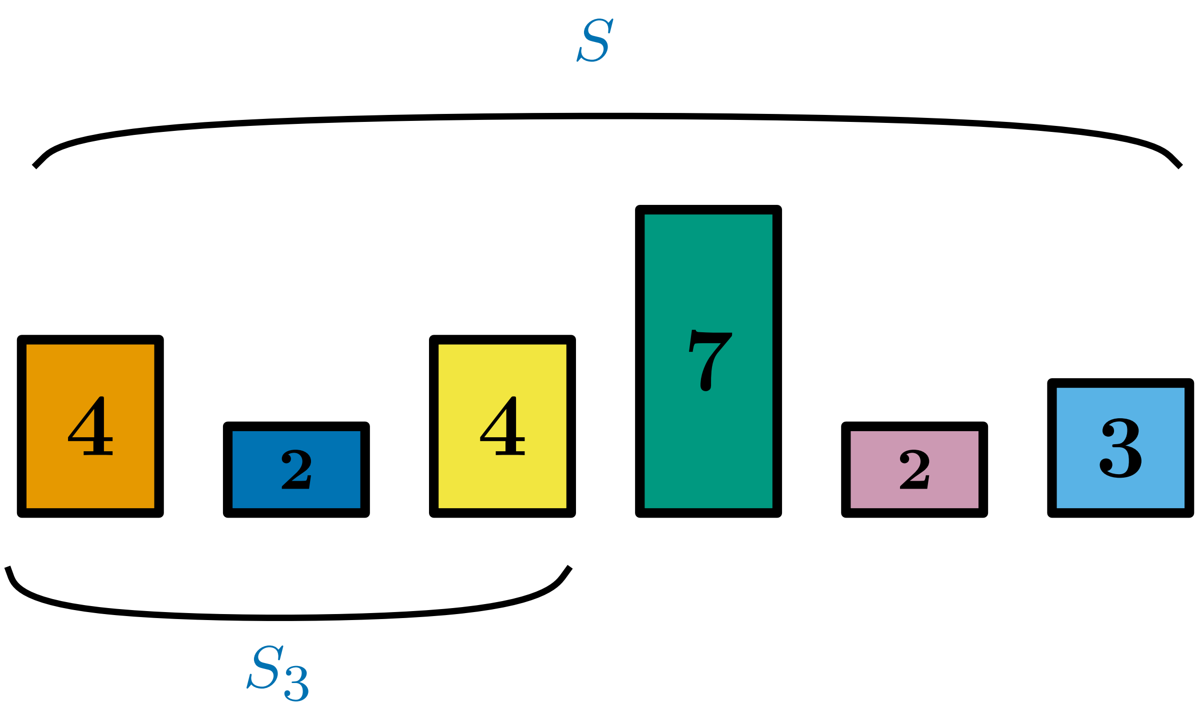
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

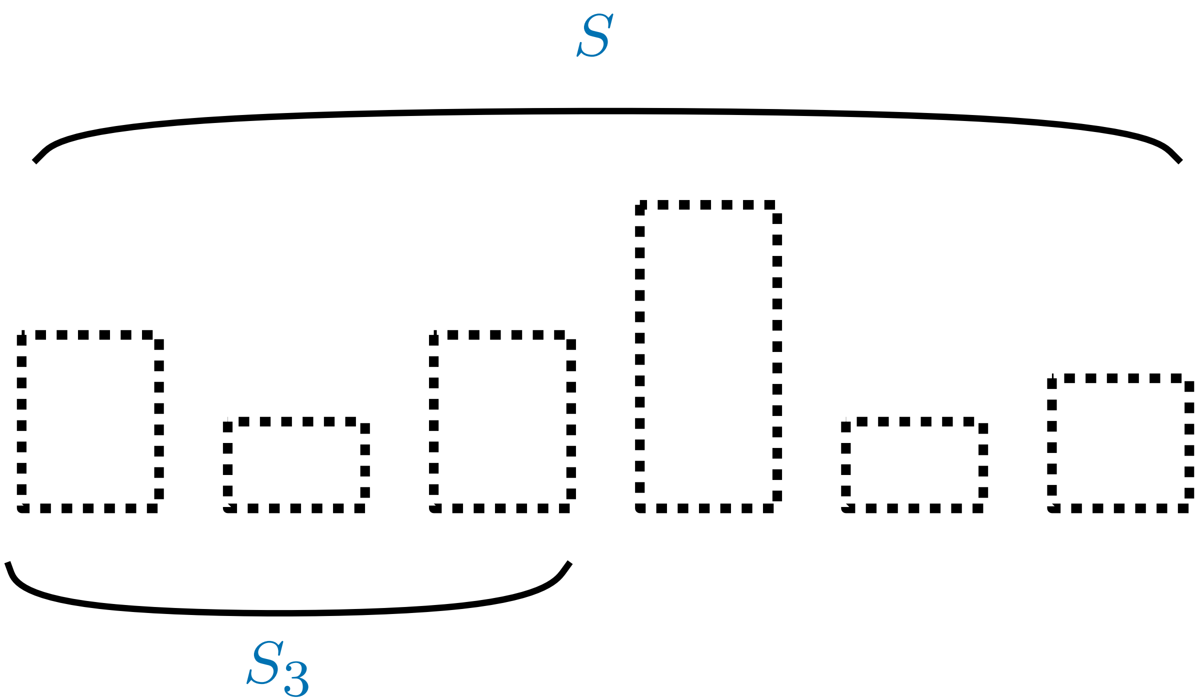
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

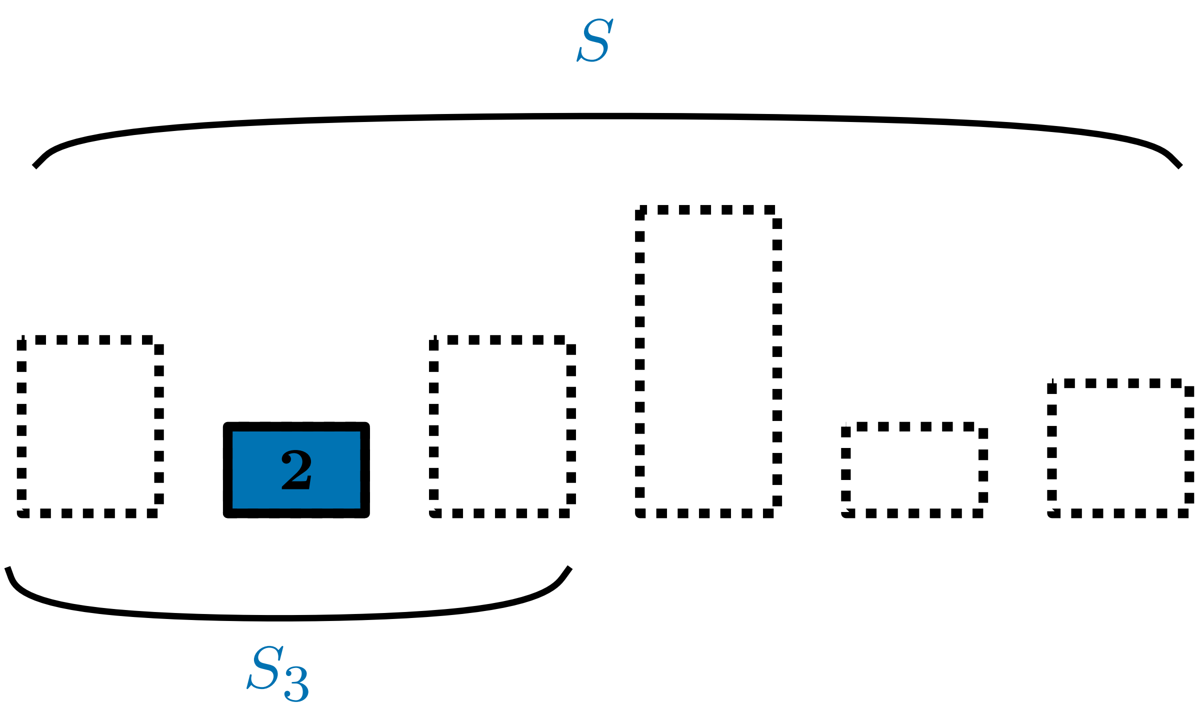
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

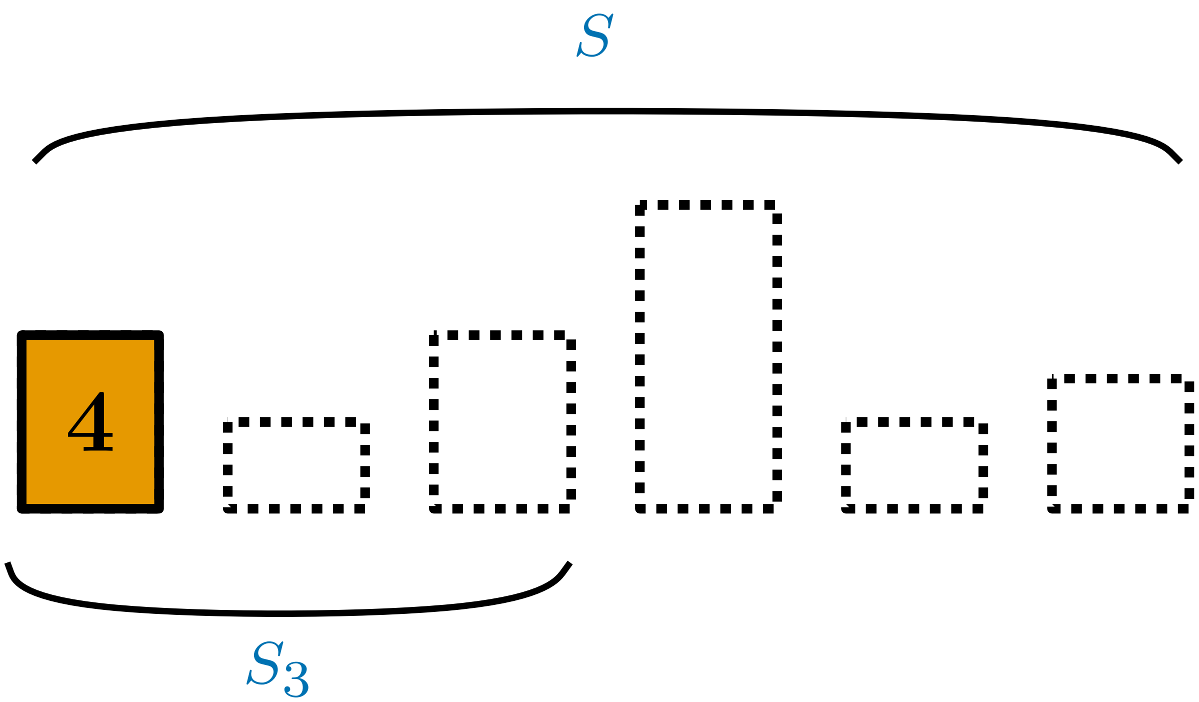
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

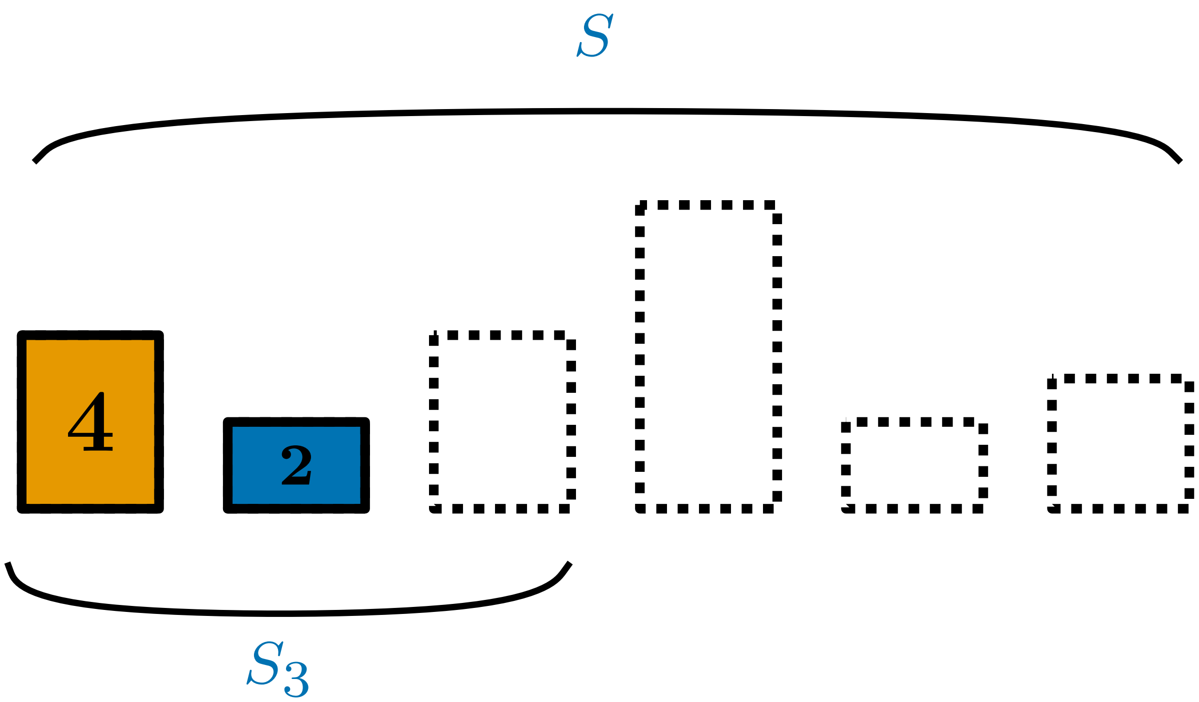
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

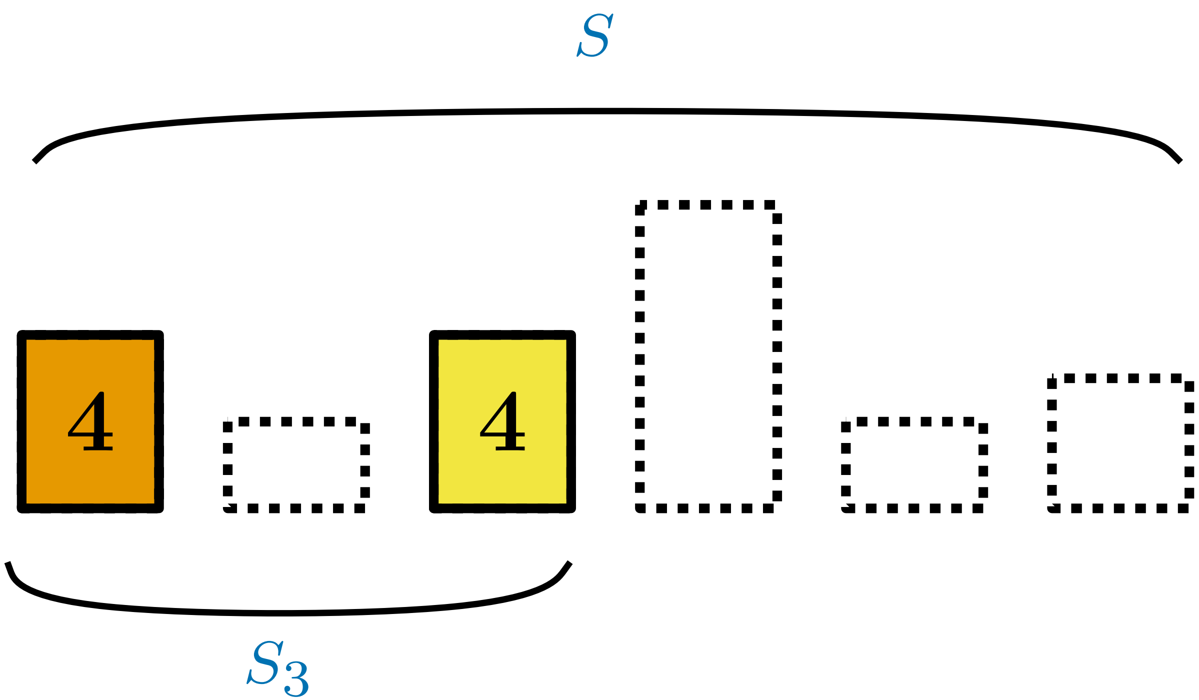
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

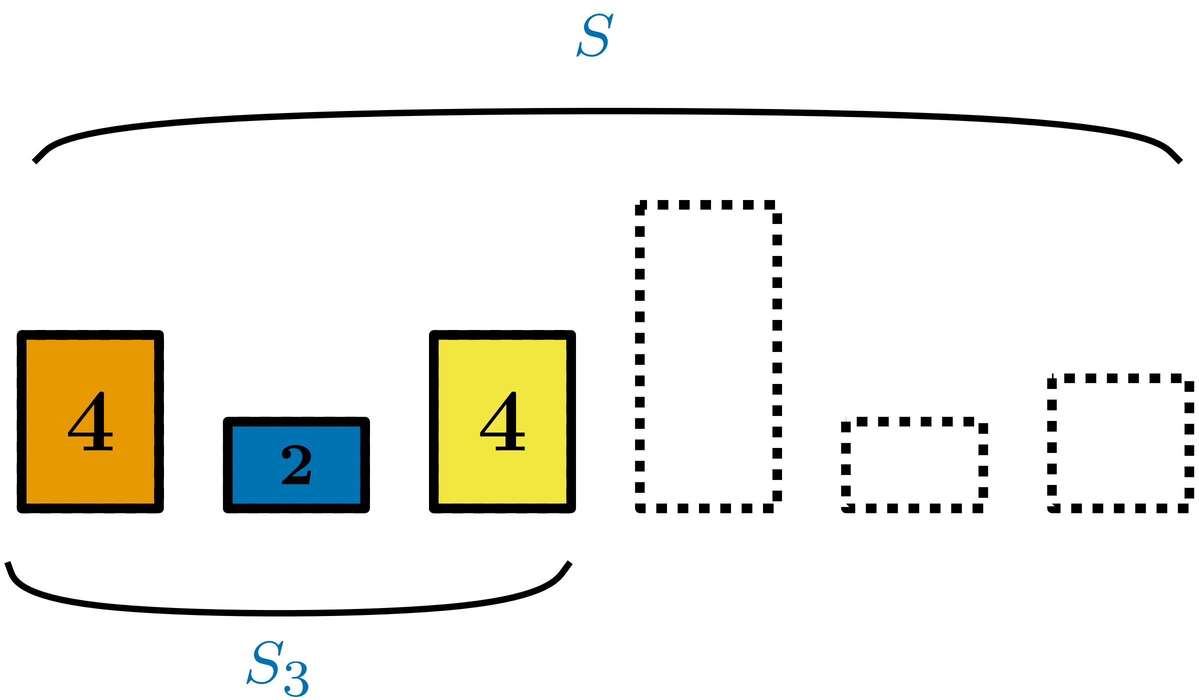
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

University of BRISTOL

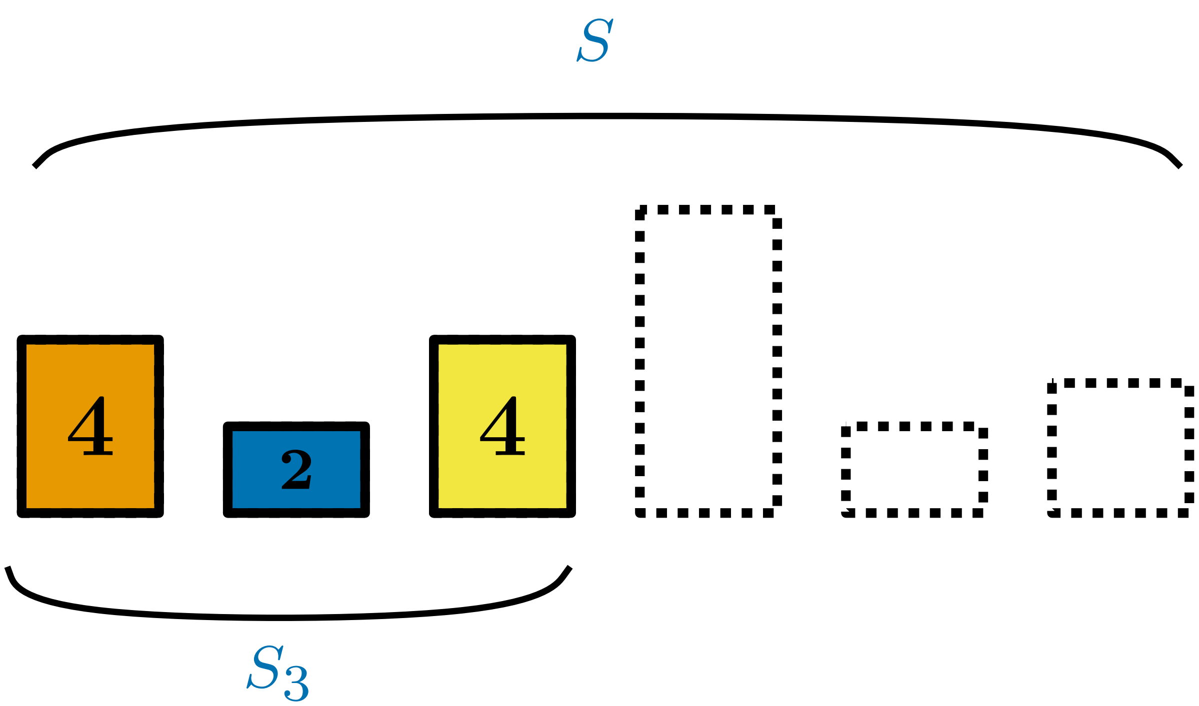
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \dots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \dots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

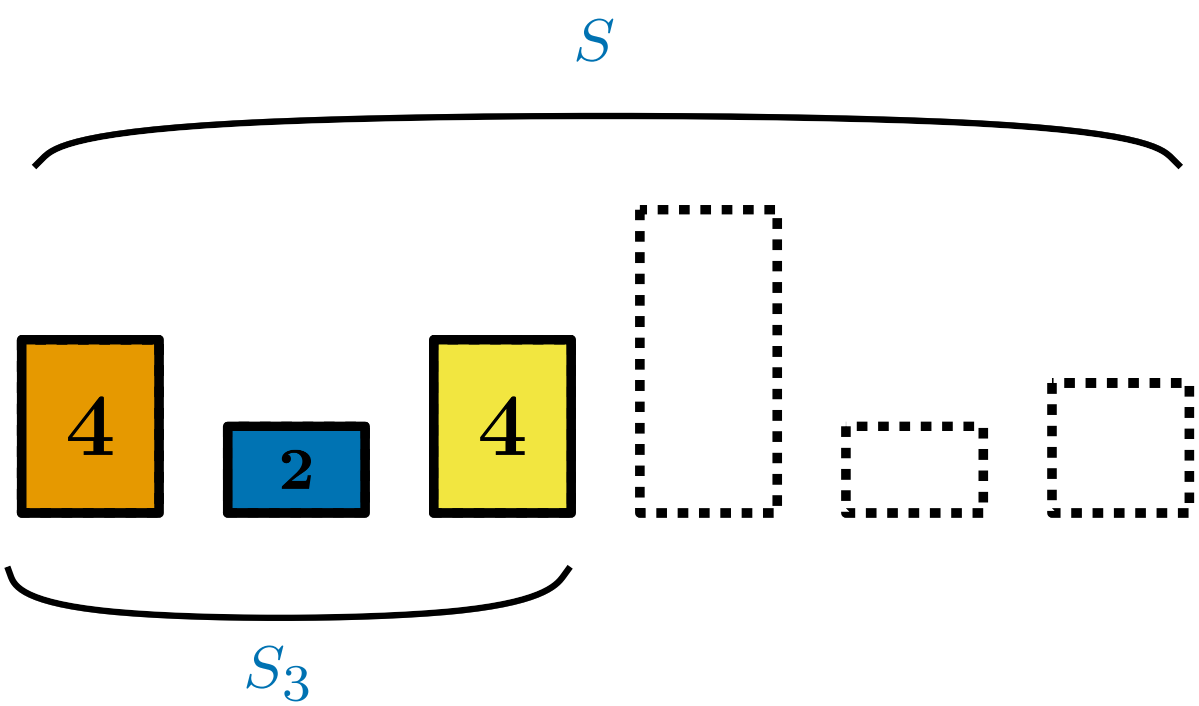
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

The largest subset of S (of size at most t) is the largest number in L_m

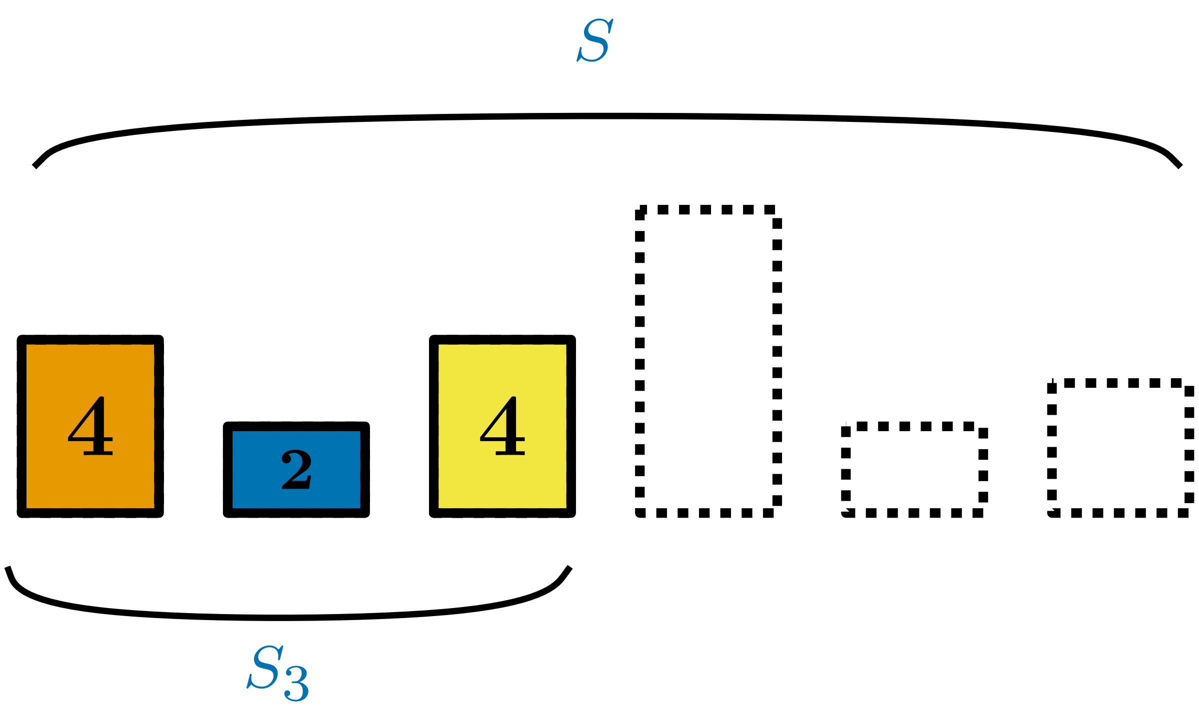
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

The largest subset of S (of size at most t) is the largest number in L_m

We compute L_i from L_{i-1}:

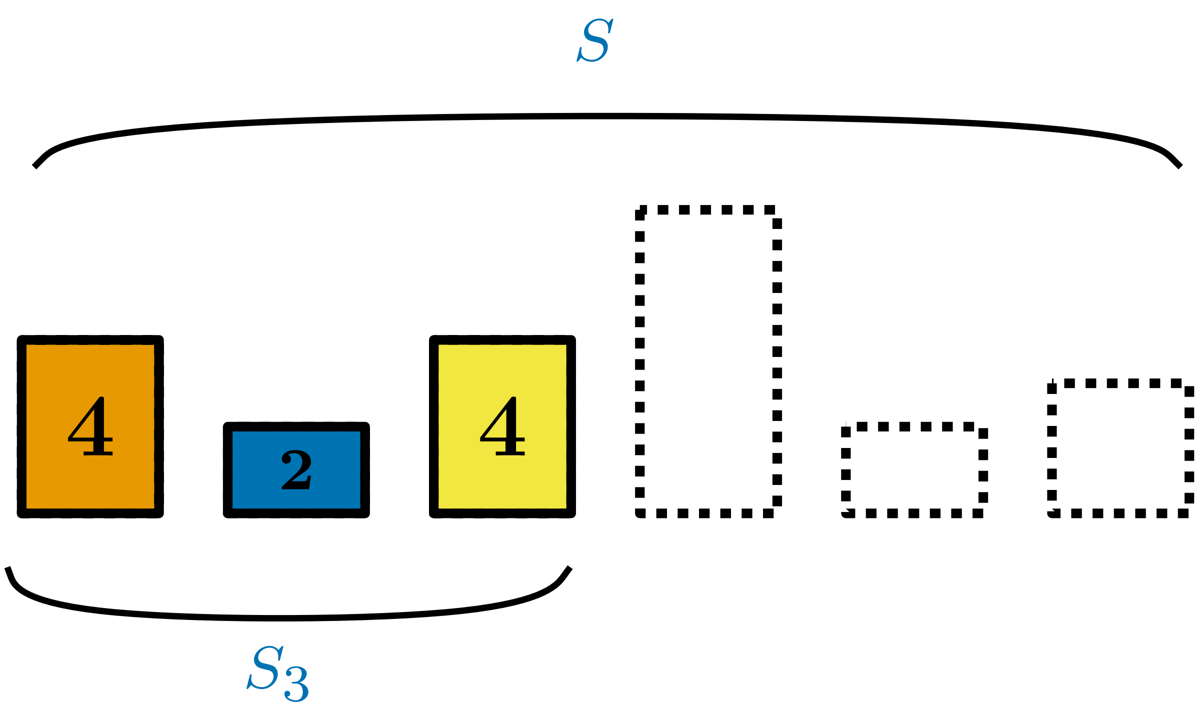
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

The largest subset of S (of size at most t) is the largest number in L_m

We compute L_i from L_{i-1}: $\quad L_i = L_{i-1} \cup (L_{i-1} + s_i)$

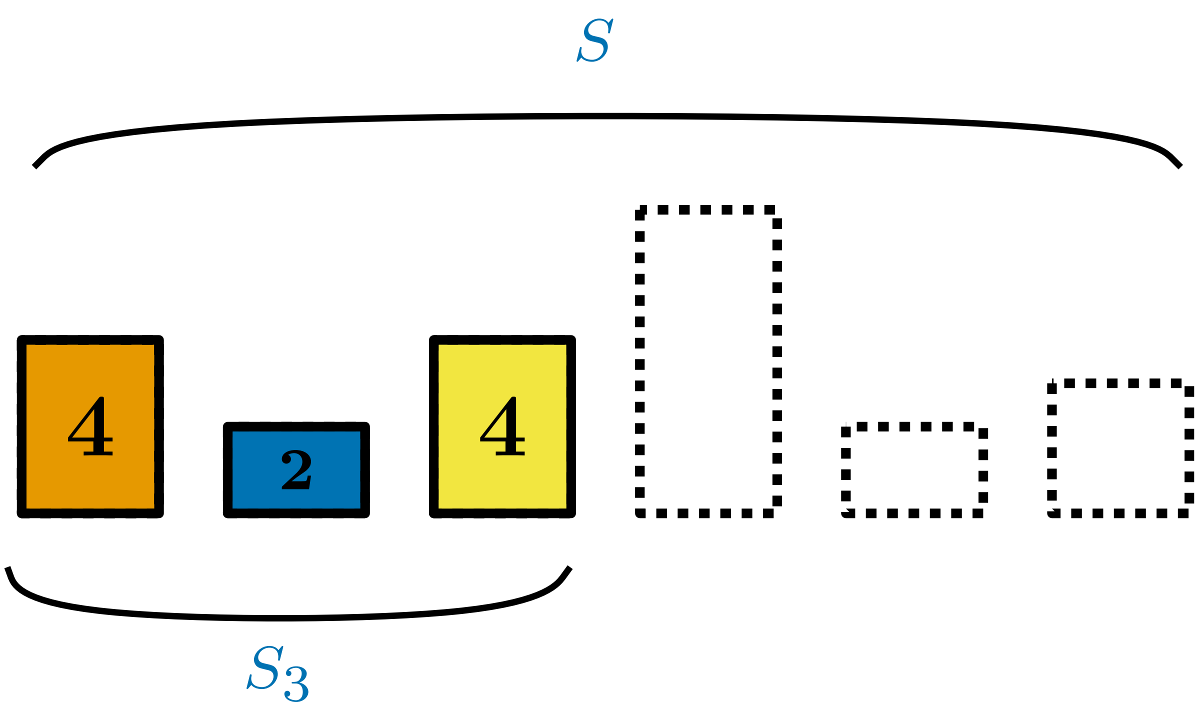
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

The largest subset of S (of size at most t) is the largest number in L_m

We compute L_i from L_{i-1}: $\quad L_i = L_{i-1} \cup (L_{i-1} + s_i)$

where $(x + s_i) \in (L_{i-1} + s_i)$ iff $x \in L_{i-1}$ and $x + s_i \leqslant t$

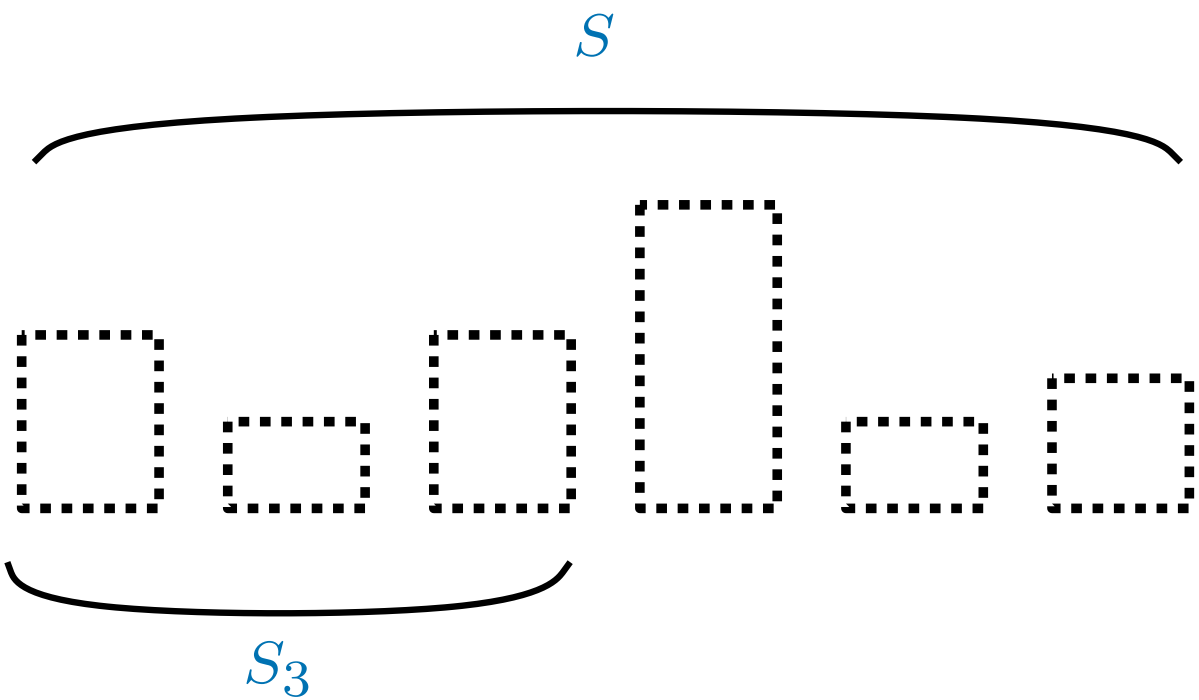
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \dots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \dots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

The largest subset of S (of size at most t) is the largest number in L_m

We compute L_i from L_{i-1}: $L_i = L_{i-1} \cup (L_{i-1} + s_i)$

where $(x + s_i) \in (L_{i-1} + s_i)$ iff $x \in L_{i-1}$ and $x + s_i \leqslant t$

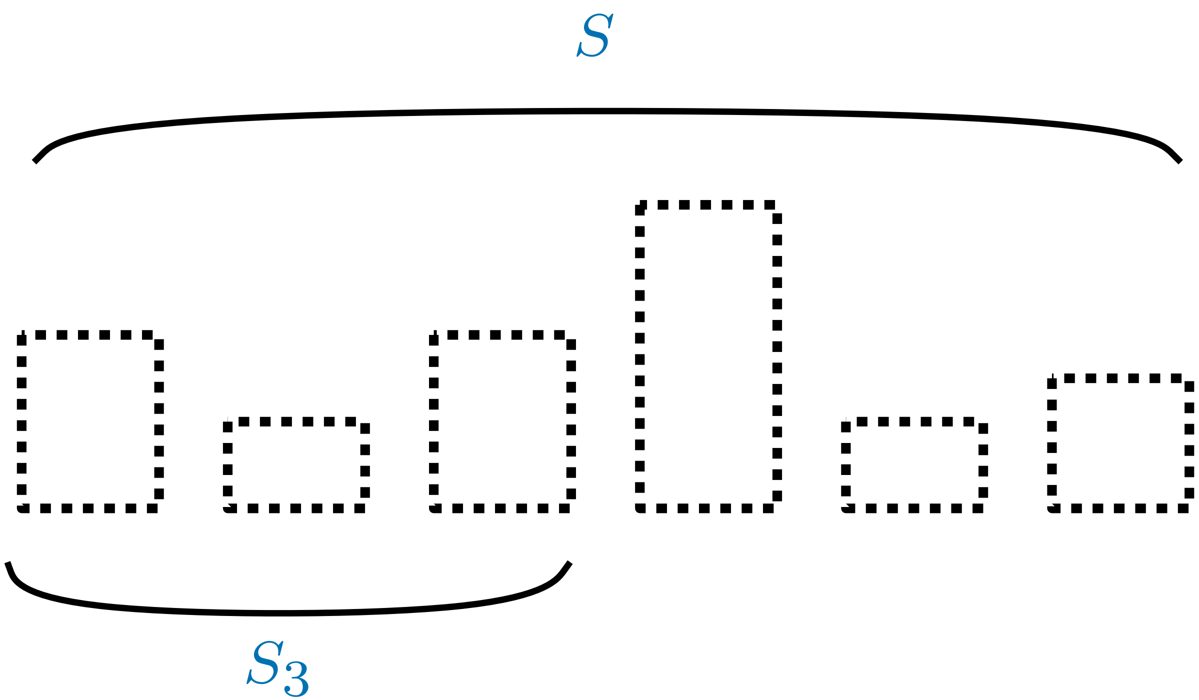
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \dots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \dots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

$L_3 + s_4 = L_3 + 7 =$

$\{7, 9, 11\}$

The largest subset of S (of size at most t) is the largest number in L_m

We compute L_i from L_{i-1}: $\quad L_i = L_{i-1} \cup (L_{i-1} + s_i)$

where $(x + s_i) \in (L_{i-1} + s_i)$ iff $x \in L_{i-1}$ and $x + s_i \leqslant t$

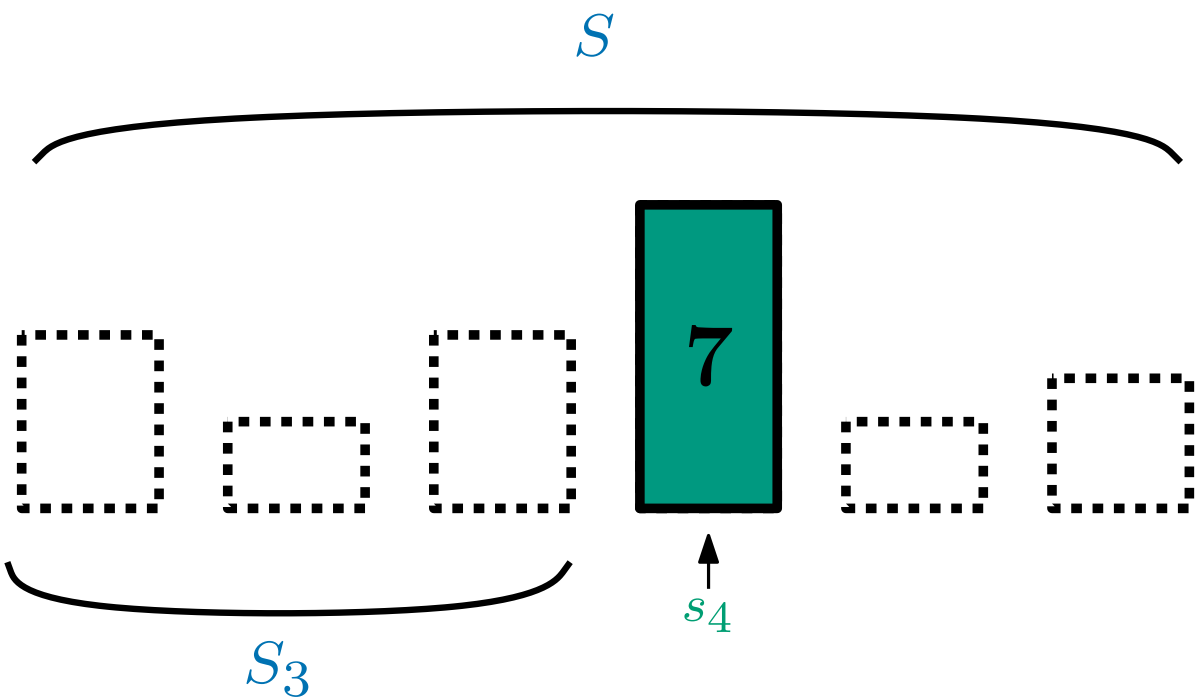
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

$L_3 + s_4 = L_3 + 7 =$

$\{7, 9, 11\}$

The largest subset of S (of size at most t) is the largest number in L_m

We compute L_i from L_{i-1}: $\quad L_i = L_{i-1} \cup (L_{i-1} + s_i)$

where $(x + s_i) \in (L_{i-1} + s_i)$ iff $x \in L_{i-1}$ and $x + s_i \leqslant t$

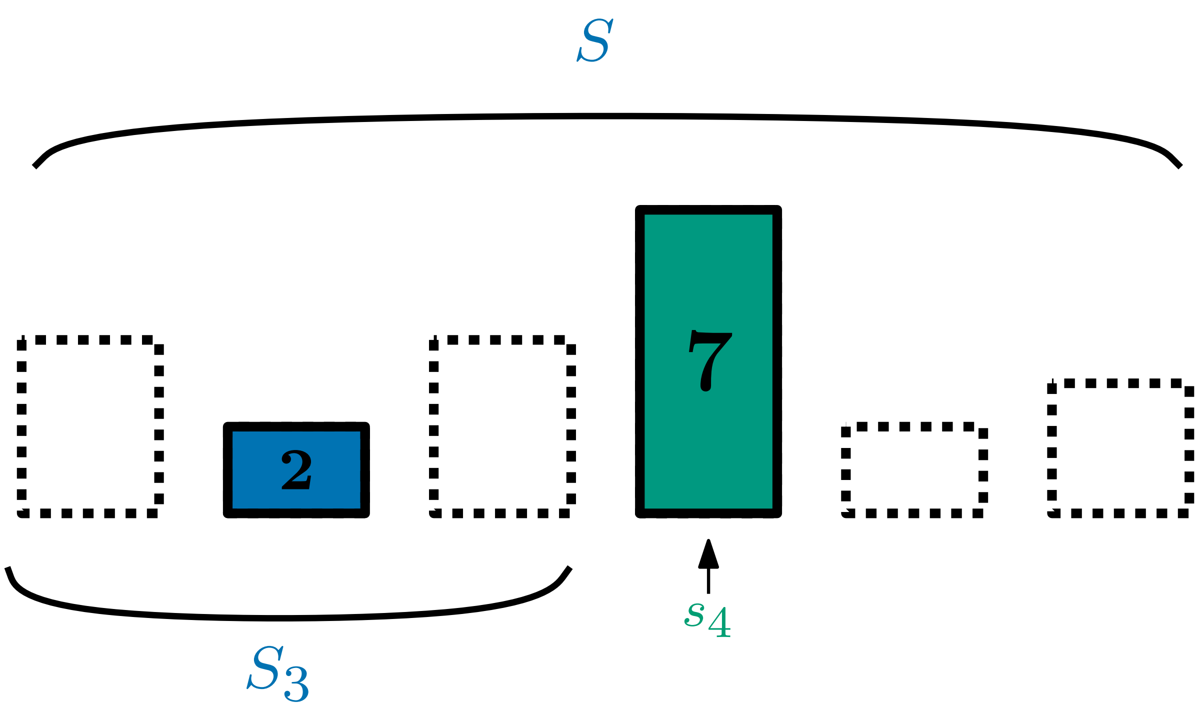
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \dots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \dots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

$L_3 + s_4 = L_3 + 7 =$

$\{7, 9, 11\}$

The largest subset of S (of size at most t) is the largest number in L_m

We compute L_i from L_{i-1}: $\quad L_i = L_{i-1} \cup (L_{i-1} + s_i)$

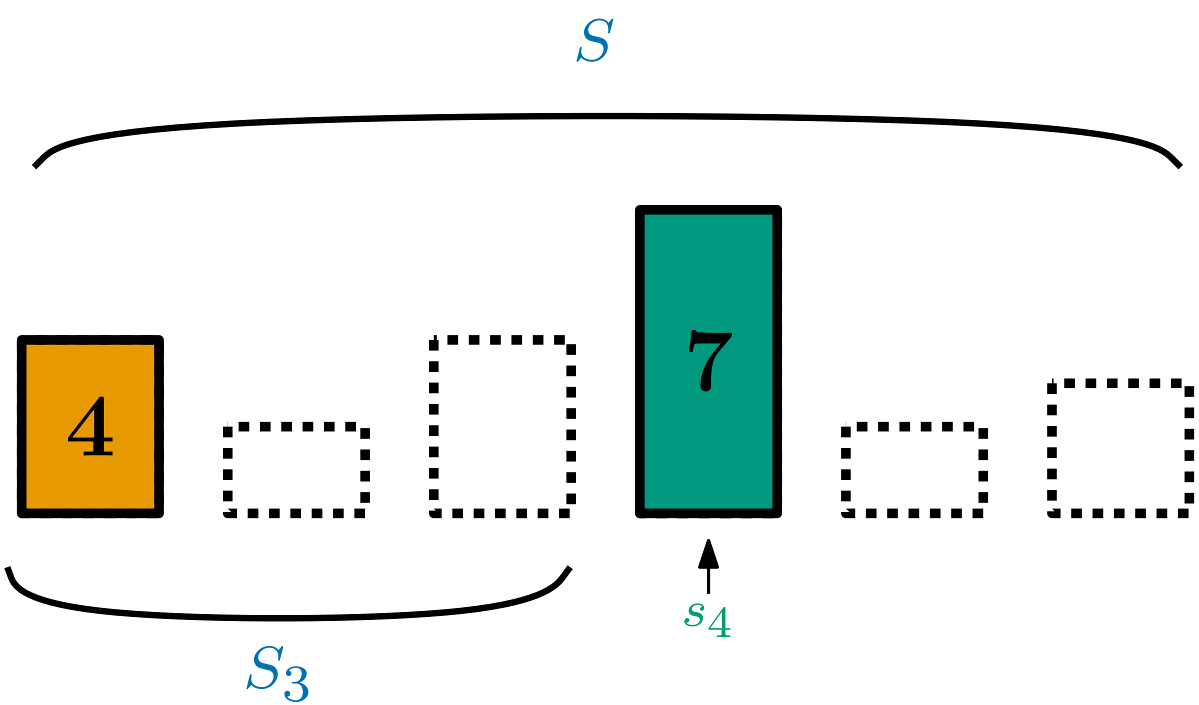
where $(x + s_i) \in (L_{i-1} + s_i)$ iff $x \in L_{i-1}$ and $x + s_i \leqslant t$

$|S| = m$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t (here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

$L_3 + s_4 = L_3 + 7 =$

$\{7, 9, 11\}$

The largest subset of S (of size at most t) is the largest number in L_m

We compute L_i from L_{i-1}: $L_i = L_{i-1} \cup (L_{i-1} + s_i)$

where $(x + s_i) \in (L_{i-1} + s_i)$ iff $x \in L_{i-1}$ and $x + s_i \leqslant t$

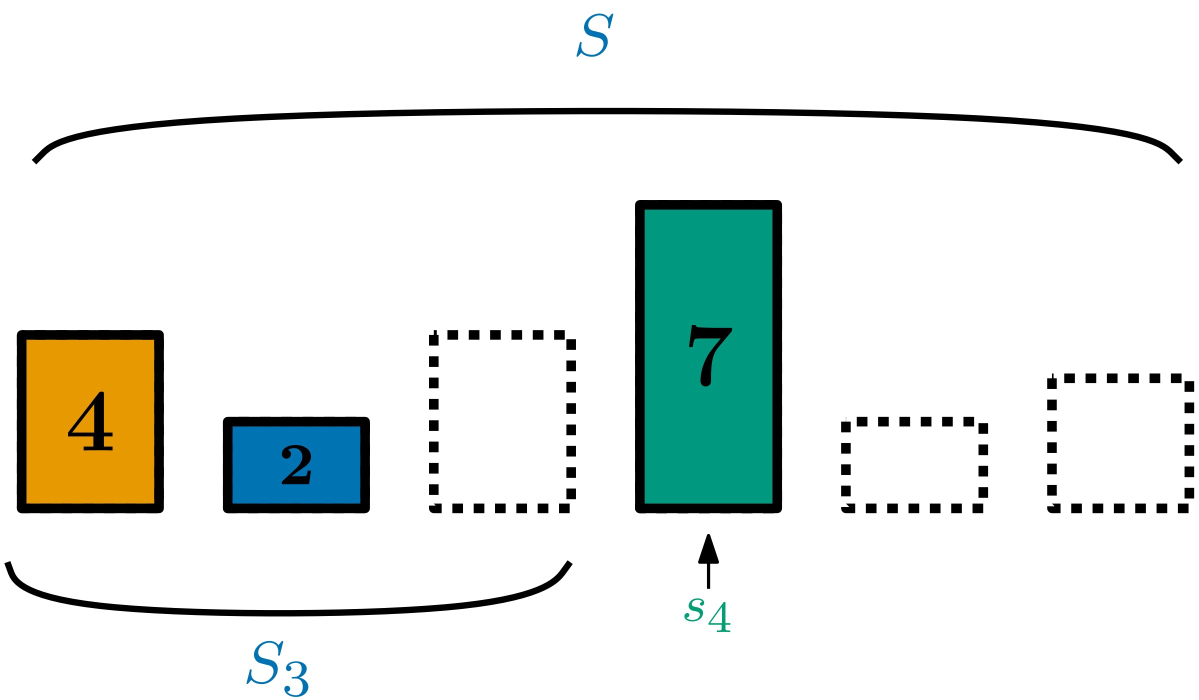
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

$L_3 + s_4 = L_3 + 7 =$

$\{7, 9, 11\}$

The largest subset of S (of size at most t) is the largest number in L_m

We compute L_i from L_{i-1}: $\quad L_i = L_{i-1} \cup (L_{i-1} + s_i)$

where $(x + s_i) \in (L_{i-1} + s_i)$ iff $x \in L_{i-1}$ and $x + s_i \leqslant t$

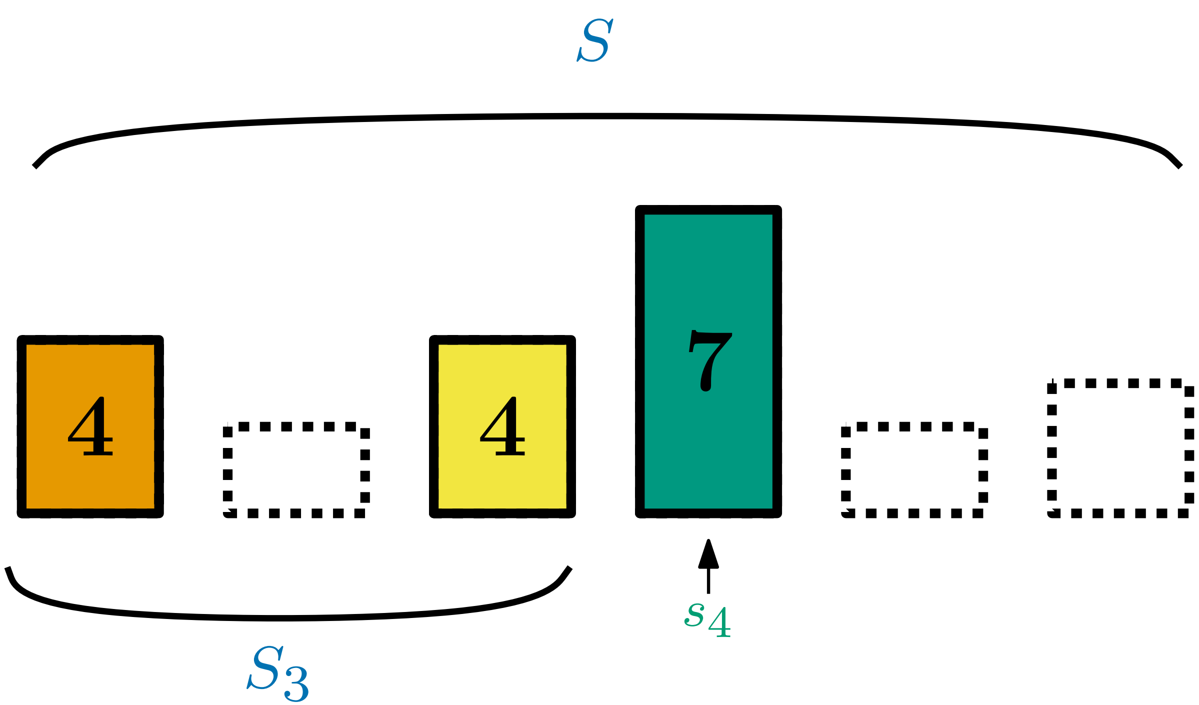
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

$L_3 + s_4 = L_3 + 7 =$

$\{7, 9, 11\}$

The largest subset of S (of size at most t) is the largest number in L_m

We compute L_i from L_{i-1}: $\quad L_i = L_{i-1} \cup (L_{i-1} + s_i)$

where $(x + s_i) \in (L_{i-1} + s_i)$ iff $x \in L_{i-1}$ and $x + s_i \leqslant t$

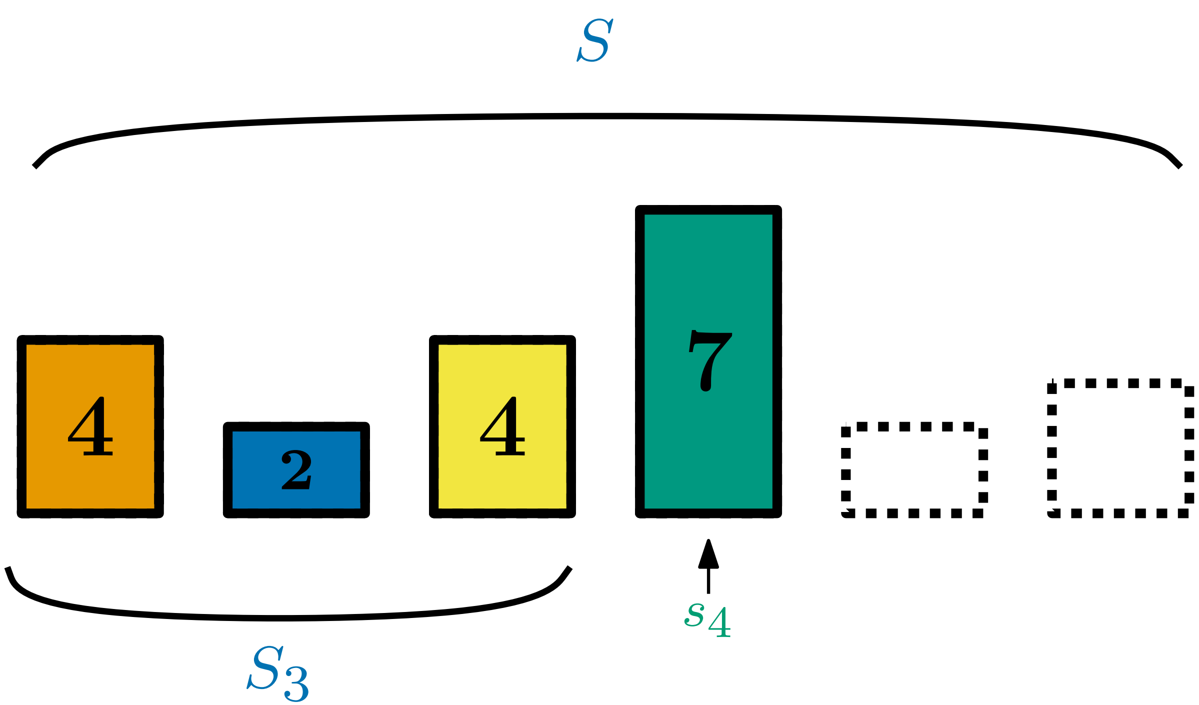
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \ldots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \ldots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

$L_3 + s_4 = L_3 + 7 =$

$\{7, 9, 11\}$

The largest subset of S (of size at most t) is the largest number in L_m

We compute L_i from L_{i-1}: $L_i = L_{i-1} \cup (L_{i-1} + s_i)$

where $(x + s_i) \in (L_{i-1} + s_i)$ iff $x \in L_{i-1}$ and $x + s_i \leqslant t$

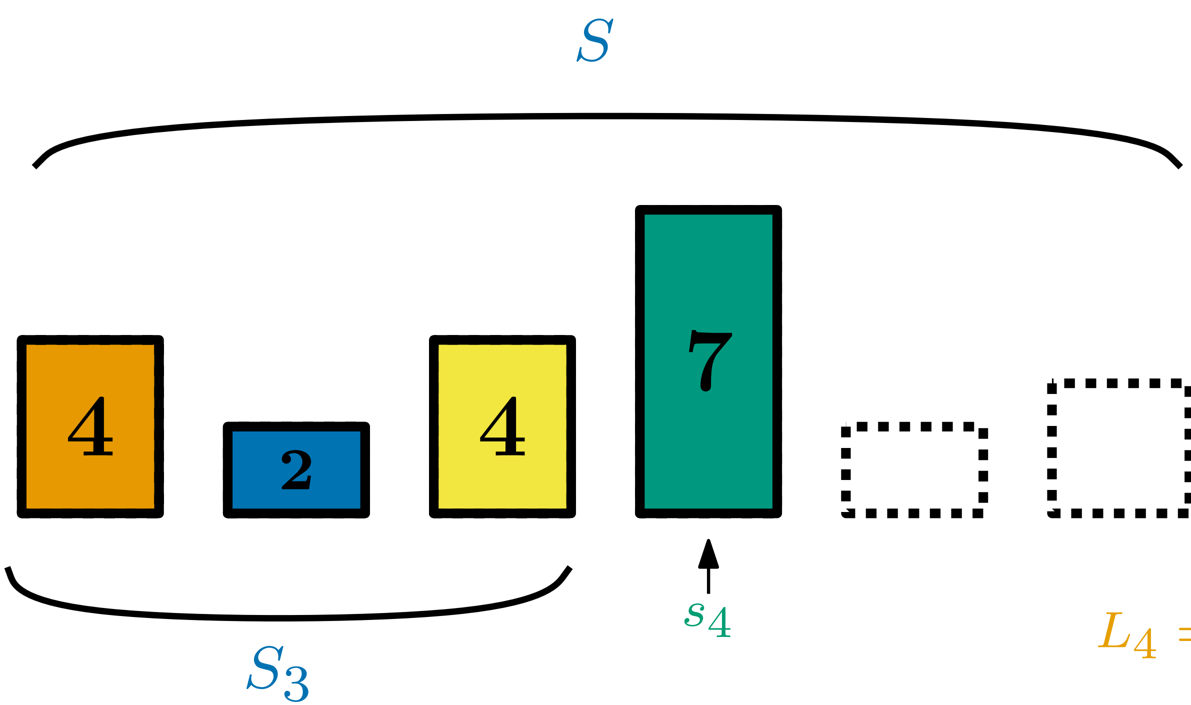
$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \dots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \dots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(here $t = 12$)



$L_3 =$

$\{0, 2, 4, 6, 8, 10\}$

$L_3 + s_4 = L_3 + 7 =$

$\{7, 9, 11\}$

$L_4 = \{0, 2, 4, 6, 7, 8, 9, 10, 11\}$

The largest subset of S (of size at most t) is the largest number in L_m

We compute L_i from L_{i-1}: $\quad L_i = L_{i-1} \cup (L_{i-1} + s_i)$

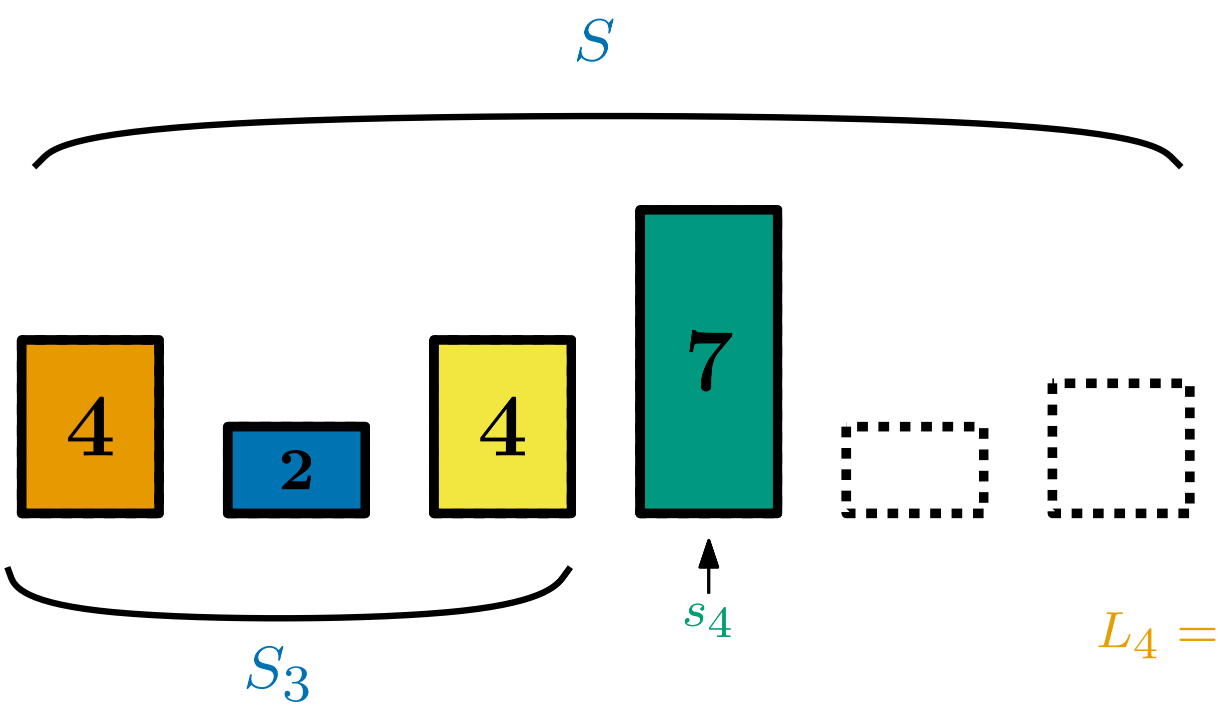
where $(x + s_i) \in (L_{i-1} + s_i)$ iff $x \in L_{i-1}$ and $x + s_i \leqslant t$

$\boxed{|S| = m}$

An exact solution

Let $S = \{s_1, s_2, s_3 \dots s_m\}$ be the set of items and $S_i = \{s_1, s_2, \dots, s_i\}$

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t (here $t = 12$)



$L_3 =$ $\{0, 2, 4, 6, 8, 10\}$

$L_3 + s_4 = L_3 + 7 =$ $\{7, 9, 11\}$

$L_4 = \{0, 2, 4, 6, 7, 8, 9, 10, 11\}$

The largest subset of S (of size at most t) is the largest number in L_m

We compute L_i from L_{i-1}: $\quad L_i = L_{i-1} \cup (L_{i-1} + s_i)$

where $(x + s_i) \in (L_{i-1} + s_i)$ iff $x \in L_{i-1}$ and $x + s_i \leqslant t$

We don't have any duplicates in L_i - so $|L_i| \leqslant t$

$\boxed{|S| = m}$

An exact solution

The algorithm

- Let $L_0 = \{0\}$
- For $i = 1 \ldots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}
  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$\boxed{|S| = m}$

An exact solution

$O(1)$ time

The algorithm

- Let $L_0 = \{0\}$
- For $i = 1 \dots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}
  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$\boxed{|S| = m}$

An exact solution

The algorithm

$O(1)$ time

$O(|L_{i-1}|)$ time

- Let $L_0 = \{0\}$
- For $i = 1 \ldots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}
  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$\boxed{|S| = m}$

An exact solution

The algorithm

$O(1)$ time

- Let $L_0 = \{0\}$

$O(|L_{i-1}|)$ time

- For $i = 1 \ldots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}

$O(|L_i|)$ time

  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$\boxed{|S| = m}$

An exact solution

$O(1)$ time

The algorithm

- Let $L_0 = \{0\}$
- For $i = 1 \ldots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}
  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$O(|L_{i-1}|)$ time

$O(|L_i|)$ time

$O(|L_m|)$ time

$\boxed{|S| = m}$

An exact solution

$O(1)$ time

The algorithm

- Let $L_0 = \{0\}$

$O(|L_{i-1}|)$ time

- For $i = 1 \ldots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}

$O(|L_i|)$ time

  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$O(|L_m|)$ time

Each L_i is of length $|L_i| \leqslant t$

$\boxed{|S| = m}$

An exact solution

The algorithm

$O(1)$ time

- Let $L_0 = \{0\}$
- For $i = 1 \ldots m$:

$O(|L_{i-1}|)$ time

  - Compute $(L_{i-1} + s_i)$ from L_{i-1}

$O(|L_i|)$ time

  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$O(|L_m|)$ time

Each L_i is of length $|L_i| \leqslant t$

The overall time complexity is therefore $O(mt)$

$\boxed{|S| = m}$

An exact solution

$O(1)$ time

The algorithm

- Let $L_0 = \{0\}$
- For $i = 1 \ldots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}
  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$O(|L_{i-1}|)$ time

$O(|L_i|)$ time

$O(|L_m|)$ time

Each L_i is of length $|L_i| \leqslant t$

The overall time complexity is therefore $O(mt)$

Is this polynomial in n?

$\boxed{|S| = m}$

An exact solution

The algorithm

$O(1)$ time

- Let $L_0 = \{0\}$
- For $i = 1 \ldots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}
  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$O(|L_{i-1}|)$ time

$O(|L_i|)$ time

$O(|L_m|)$ time

Each L_i is of length $|L_i| \leqslant t$

The overall time complexity is therefore $O(mt)$

Is this polynomial in n?

What even is n?

$\boxed{|S| = m}$

An exact solution

The algorithm

$O(1)$ time

- Let $L_0 = \{0\}$
- For $i = 1 \ldots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}
  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$O(|L_{i-1}|)$ time

$O(|L_i|)$ time

$O(|L_m|)$ time

Each L_i is of length $|L_i| \leqslant t$

The overall time complexity is therefore $O(mt)$

Is this polynomial in n?

What even is n?

n is the *length of the input* (measured in words)

$\boxed{|S| = m}$

An exact solution

The algorithm

$O(1)$ time

$O(|L_{i-1}|)$ time

- Let $L_0 = \{0\}$
- For $i = 1 \ldots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}
  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$

$O(|L_i|)$ time

- Output the largest number in L_m

$O(|L_m|)$ time

Each L_i is of length $|L_i| \leqslant t$

Is this polynomial in n?

The overall time complexity is therefore $O(mt)$

What even is n?

n is the *length of the input* (measured in words)

Input

n words

$\boxed{|S| = m}$

An exact solution

The algorithm

$O(1)$ time

- Let $L_0 = \{0\}$

$O(|L_{i-1}|)$ time

- For $i = 1 \ldots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}

$O(|L_i|)$ time

  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$

- Output the largest number in L_m

$O(|L_m|)$ time

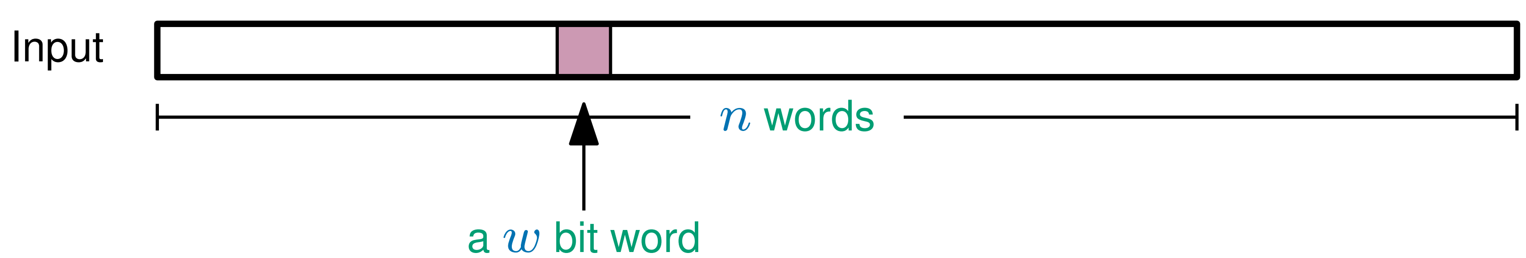
Each L_i is of length $|L_i| \leqslant t$

Is this polynomial in n?

The overall time complexity is therefore $O(mt)$

What even is n?

n is the *length of the input* (measured in words)



$\boxed{|S| = m}$

An exact solution

The algorithm

$O(1)$ time

- Let $L_0 = \{0\}$

$O(|L_{i-1}|)$ time

- For $i = 1 \ldots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}

$O(|L_i|)$ time

  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$O(|L_m|)$ time

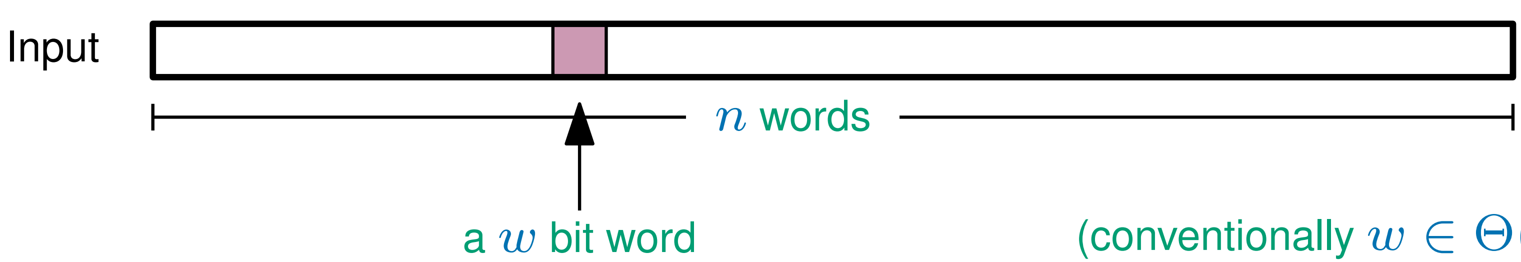
Each L_i is of length $|L_i| \leqslant t$

Is this polynomial in n?

The overall time complexity is therefore $O(mt)$

What even is n?

n is the *length of the input* (measured in words)



(conventionally $w \in \Theta(\log n)$)

$\boxed{|S| = m}$

An exact solution

The algorithm

$O(1)$ time

- Let $L_0 = \{0\}$
- For $i = 1 \ldots m$:

$O(|L_{i-1}|)$ time

  - Compute $(L_{i-1} + s_i)$ from L_{i-1}

$O(|L_i|)$ time

  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$O(|L_m|)$ time

Each L_i is of length $|L_i| \leqslant t$

Is this polynomial in n?

The overall time complexity is therefore $O(mt)$

What even is n?

n is the *length of the input* (measured in words)

Input

n words

$\boxed{|S| = m}$

An exact solution

The algorithm

- Let $L_0 = \{0\}$ — $O(1)$ time
- For $i = 1 \ldots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1} — $O(|L_{i-1}|)$ time
  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$ — $O(|L_i|)$ time
- Output the largest number in L_m — $O(|L_m|)$ time

Each L_i is of length $|L_i| \leqslant t$

Is this polynomial in n?

The overall time complexity is therefore $O(mt)$

What even is n?

n is the *length of the input* (measured in words)

Input

n words

The input to the Subset Sum problem is a list of the elements of S along with t encoded in binary in a total of n words

$\boxed{|S| = m}$

An exact solution

The algorithm

$O(1)$ time

- Let $L_0 = \{0\}$
- For $i = 1 \ldots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}
  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$O(|L_{i-1}|)$ time

$O(|L_i|)$ time

$O(|L_m|)$ time

Each L_i is of length $|L_i| \leqslant t$

The overall time complexity is therefore $O(mt)$

Is this polynomial in n?

What even is n?

n is the *length of the input* (measured in words)

Input	s_1	s_2	s_3	$\bullet\bullet\bullet$	t

n words

The input to the Subset Sum problem is a list of the elements of S along with t

encoded in binary in a total of n words

$\boxed{|S| = m}$

An exact solution

The algorithm

$O(1)$ time

- Let $L_0 = \{0\}$
- For $i = 1 \ldots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}
  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$O(|L_{i-1}|)$ time

$O(|L_i|)$ time

$O(|L_m|)$ time

Each L_i is of length $|L_i| \leqslant t$

The overall time complexity is therefore $O(mt)$

Is this polynomial in n?

What even is n?

n is the *length of the input* (measured in words)

Input

s_1	s_2	s_3	$\cdots$	t

n words

The input to the Subset Sum problem is a list of the elements of S along with t encoded in binary in a total of n words

As $m \leqslant n$, the time is $O(nt)$

$\boxed{|S| = m}$

An exact solution

$O(1)$ time

The algorithm

- Let $L_0 = \{0\}$

$O(|L_{i-1}|)$ time

- For $i = 1 \dots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}

$O(|L_i|)$ time

  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$O(|L_m|)$ time

Each L_i is of length $|L_i| \leqslant t$

Is this polynomial in n?

The overall time complexity is therefore $O(mt)$

What even is n?

n is the *length of the input* (measured in words)

Input

s_1	s_2	s_3	$\bullet\bullet\bullet$	t

n words

The input to the Subset Sum problem is a list of the elements of S along with t encoded in binary in a total of n words

As $m \leqslant n$, the time is $O(nt)$ ...but t could be (for example) 2^n

$\boxed{|S| = m}$

An exact solution

The algorithm

$O(1)$ time

- Let $L_0 = \{0\}$
- For $i = 1 \ldots m$:
  - Compute $(L_{i-1} + s_i)$ from L_{i-1}
  - Compute $L_i = L_{i-1} \cup (L_{i-1} + s_i)$
- Output the largest number in L_m

$O(|L_{i-1}|)$ time

$O(|L_i|)$ time

$O(|L_m|)$ time

Each L_i is of length $|L_i| \leqslant t$

The overall time complexity is therefore $O(mt)$

Is this polynomial in n?

What even is n?

n is the *length of the input* (measured in words)

Input	s_1	s_2	s_3	$\bullet\bullet\bullet$	t

n words

The input to the Subset Sum problem is a list of the elements of S along with t

encoded in binary in a total of n words

As $m \leqslant n$, the time is $O(nt)$ ...but t could be (for example) 2^n ...in other words $O(n2^n)$ time!

Pseudo-polynomial time algorithms

We say that an algorithm is *pseudo-polynomial time*

if it runs in polynomial time when all the numbers are integers $\leqslant n^c$

for some constant c

Pseudo-polynomial time algorithms

We say that an algorithm is *pseudo-polynomial time*

if it runs in polynomial time when all the numbers are integers $\leqslant n^c$

for some constant c

The algorithm for Subset Sum given takes $O(nt) = O(n^{c+1})$ time

(in this case)

Pseudo-polynomial time algorithms

We say that an algorithm is *pseudo-polynomial time*

if it runs in polynomial time when all the numbers are integers $\leqslant n^c$

for some constant c

The algorithm for Subset Sum given takes $O(nt) = O(n^{c+1})$ time

(in this case)

So there is a pseudo-polynomial time algorithm for Subset Sum

Pseudo-polynomial time algorithms

We say that an algorithm is *pseudo-polynomial time*

if it runs in polynomial time when all the numbers are integers $\leqslant n^c$

for some constant c

The algorithm for Subset Sum given takes $O(nt) = O(n^{c+1})$ time

(in this case)

So there is a pseudo-polynomial time algorithm for Subset Sum

------- A diversion into computational complexity --

Pseudo-polynomial time algorithms

We say that an algorithm is *pseudo-polynomial time*

if it runs in polynomial time when all the numbers are integers $\leqslant n^c$

for some constant c

The algorithm for Subset Sum given takes $O(nt) = O(n^{c+1})$ time

(in this case)

So there is a pseudo-polynomial time algorithm for Subset Sum

------- A diversion into computational complexity ---

We say that an NP-complete problem is weakly NP-complete if

there is a pseudo-polynomial time algorithm for it

Pseudo-polynomial time algorithms

We say that an algorithm is *pseudo-polynomial time*

if it runs in polynomial time when all the numbers are integers $\leqslant n^c$

for some constant c

The algorithm for Subset Sum given takes $O(nt) = O(n^{c+1})$ time

(in this case)

So there is a pseudo-polynomial time algorithm for Subset Sum

------- A diversion into computational complexity ---

We say that an NP-complete problem is weakly NP-complete if

there is a pseudo-polynomial time algorithm for it

The decision version of Subset Sum is weakly NP-complete

Pseudo-polynomial time algorithms

We say that an algorithm is *pseudo-polynomial time*

if it runs in polynomial time when all the numbers are integers $\leqslant n^c$

for some constant c

The algorithm for Subset Sum given takes $O(nt) = O(n^{c+1})$ time

(in this case)

So there is a pseudo-polynomial time algorithm for Subset Sum

------ A diversion into computational complexity ------

We say that an NP-complete problem is weakly NP-complete if

there is a pseudo-polynomial time algorithm for it

The decision version of Subset Sum is weakly NP-complete

We say that an NP-complete problem is strongly NP-complete if

it remains NP-complete when all the numbers are integers $\leqslant n^c$

Pseudo-polynomial time algorithms

We say that an algorithm is *pseudo-polynomial time*

if it runs in polynomial time when all the numbers are integers $\leqslant n^c$

for some constant c

The algorithm for Subset Sum given takes $O(nt) = O(n^{c+1})$ time

(in this case)

So there is a pseudo-polynomial time algorithm for Subset Sum

------ A diversion into computational complexity --

We say that an NP-complete problem is weakly NP-complete if

there is a pseudo-polynomial time algorithm for it

The decision version of Subset Sum is weakly NP-complete

We say that an NP-complete problem is strongly NP-complete if

it remains NP-complete when all the numbers are integers $\leqslant n^c$

The decision version of Bin packing is strongly NP-complete

Pseudo-polynomial time algorithms

We say that an algorithm is *pseudo-polynomial time*

if it runs in polynomial time when all the numbers are integers $\leqslant n^c$

for some constant c

The algorithm for Subset Sum given takes $O(nt) = O(n^{c+1})$ time

(in this case)

So there is a pseudo-polynomial time algorithm for Subset Sum

------ A diversion into computational complexity --

We say that an NP-complete problem is weakly NP-complete if

there is a pseudo-polynomial time algorithm for it

The decision version of Subset Sum is weakly NP-complete

We say that an NP-complete problem is strongly NP-complete if

it remains NP-complete when all the numbers are integers $\leqslant n^c$

The decision version of Bin packing is strongly NP-complete

(this only makes sense if you rephrase the problem)

Pseudo-polynomial time algorithms

We say that an algorithm is *pseudo-polynomial time*

if it runs in polynomial time when all the numbers are integers $\leqslant n^c$

for some constant c

The algorithm for Subset Sum given takes $O(nt) = O(n^{c+1})$ time

(in this case)

So there is a pseudo-polynomial time algorithm for Subset Sum

------ A diversion into computational complexity ------------------------------------

We say that an NP-complete problem is weakly NP-complete if

there is a pseudo-polynomial time algorithm for it

The decision version of Subset Sum is weakly NP-complete

We say that an NP-complete problem is strongly NP-complete if

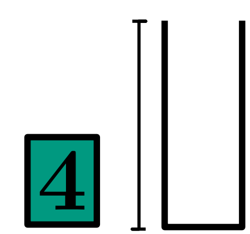
it remains NP-complete when all the numbers are integers $\leqslant n^c$

The decision version of Bin packing is strongly NP-complete

(this only makes sense if you rephrase the problem)

bins have size

$t \in [n^c]$



item sizes are

integers in $[n^c]$

Polynomial time approximation schemes

A Polynomial Time Approximation Scheme (PTAS) for problem P

is a family of algorithms:

For any constant $\epsilon > 0$ there is an algorithm in the family, A_ϵ

such that A_ϵ is a $(1 + \epsilon)$-approximation algorithm for P

Polynomial time approximation schemes

A Polynomial Time Approximation Scheme (PTAS) for problem P

is a family of algorithms:

For any constant $\epsilon > 0$ there is an algorithm in the family, A_ϵ

such that A_ϵ is a $(1 + \epsilon)$-approximation algorithm for P

- If we had a PTAS for Subset Sum we could:

Polynomial time approximation schemes

A Polynomial Time Approximation Scheme (PTAS) for problem P

is a family of algorithms:

For any constant $\epsilon > 0$ there is an algorithm in the family, A_ϵ

such that A_ϵ is a $(1+\epsilon)$-approximation algorithm for P

- If we had a PTAS for Subset Sum we could:

Let $\epsilon = 0.1$ so that $A_{0.1}$ runs in polynomial time and

outputs a subset of size at least $\frac{\text{Opt}}{1.1} > 0.9 \cdot \text{Opt}$

Polynomial time approximation schemes

A Polynomial Time Approximation Scheme (PTAS) for problem P

is a family of algorithms:

For any constant $\epsilon > 0$ there is an algorithm in the family, A_ϵ

such that A_ϵ is a $(1+\epsilon)$-approximation algorithm for P

- If we had a PTAS for Subset Sum we could:

  Let $\epsilon = 0.1$ so that $A_{0.1}$ runs in polynomial time and

  outputs a subset of size at least $\frac{\text{Opt}}{1.1} > 0.9 \cdot \text{Opt}$

  Let $\epsilon = 0.01$ so that $A_{0.01}$ *also* runs in polynomial time and

  outputs a subset of size at least $\frac{\text{Opt}}{1.01} > 0.99 \cdot \text{Opt}$

Polynomial time approximation schemes

A Polynomial Time Approximation Scheme (PTAS) for problem P

is a family of algorithms:

For any constant $\epsilon > 0$ there is an algorithm in the family, A_ϵ

such that A_ϵ is a $(1+\epsilon)$-approximation algorithm for P

- If we had a PTAS for Subset Sum we could:

Let $\epsilon = 0.1$ so that $A_{0.1}$ runs in polynomial time and

outputs a subset of size at least $\dfrac{\text{Opt}}{1.1} > 0.9 \cdot \text{Opt}$

Let $\epsilon = 0.01$ so that $A_{0.01}$ *also* runs in polynomial time and

outputs a subset of size at least $\dfrac{\text{Opt}}{1.01} > 0.99 \cdot \text{Opt}$

Let $\epsilon = 0.001$ so that $A_{0.001}$ *also* runs in polynomial time and

outputs a subset of size at least $\dfrac{\text{Opt}}{1.001} > 0.999 \cdot \text{Opt}$

Polynomial time approximation schemes

A Polynomial Time Approximation Scheme (PTAS) for problem P

is a family of algorithms:

For any constant $\epsilon > 0$ there is an algorithm in the family, A_ϵ

such that A_ϵ is a $(1+\epsilon)$-approximation algorithm for P

- If we had a PTAS for Subset Sum we could:

  Let $\epsilon = 0.1$ so that $A_{0.1}$ runs in polynomial time and

  outputs a subset of size at least $\frac{\text{Opt}}{1.1} > 0.9 \cdot \text{Opt}$

Polynomial time approximation schemes

A Polynomial Time Approximation Scheme (PTAS) for problem P

is a family of algorithms:

For any constant $\epsilon > 0$ there is an algorithm in the family, A_ϵ

such that A_ϵ is a $(1+\epsilon)$-approximation algorithm for P

- If we had a PTAS for Subset Sum we could:

  Let $\epsilon = 0.1$ so that $A_{0.1}$ runs in polynomial time and

  outputs a subset of size at least $\frac{\text{Opt}}{1.1} > 0.9 \cdot \text{Opt}$

A PTAS does not have to have a time complexity which is polynomial in $1/\epsilon$

Polynomial time approximation schemes

A Polynomial Time Approximation Scheme (PTAS) for problem P

is a family of algorithms:

For any constant $\epsilon > 0$ there is an algorithm in the family, A_ϵ

such that A_ϵ is a $(1+\epsilon)$-approximation algorithm for P

- If we had a PTAS for Subset Sum we could:

Let $\epsilon = 0.1$ so that $A_{0.1}$ runs in polynomial time and

outputs a subset of size at least $\frac{\text{Opt}}{1.1} > 0.9 \cdot \text{Opt}$

A PTAS does not have to have a time complexity which is polynomial in $1/\epsilon$

A_ϵ can have a time complexity of $O(n^{\frac{c}{\epsilon}})$ for example

Polynomial time approximation schemes

A Polynomial Time Approximation Scheme (PTAS) for problem P

is a family of algorithms:

For any constant $\epsilon > 0$ there is an algorithm in the family, A_ϵ

such that A_ϵ is a $(1+\epsilon)$-approximation algorithm for P

- If we had a PTAS for Subset Sum we could:

Let $\epsilon = 0.1$ so that $A_{0.1}$ runs in polynomial time and

outputs a subset of size at least $\frac{\text{Opt}}{1.1} > 0.9 \cdot \text{Opt}$

A PTAS does not have to have a time complexity which is polynomial in $1/\epsilon$

A_ϵ can have a time complexity of $O(n^{\frac{c}{\epsilon}})$ for example

$O(n^{10c})$ vs. $O(n^{100c})$ vs. $O(n^{1000c})$ in our example

$\epsilon = 0.1$ $\quad$ $\epsilon = 0.01$ $\quad$ $\epsilon = 0.001$

Polynomial time approximation schemes

A Polynomial Time Approximation Scheme (PTAS) for problem P

is a family of algorithms:

For any constant $\epsilon > 0$ there is an algorithm in the family, A_ϵ

such that A_ϵ is a $(1+\epsilon)$-approximation algorithm for P

- If we had a PTAS for Subset Sum we could:

Let $\epsilon = 0.1$ so that $A_{0.1}$ runs in polynomial time and

outputs a subset of size at least $\frac{\text{Opt}}{1.1} > 0.9 \cdot \text{Opt}$

A PTAS does not have to have a time complexity which is polynomial in $1/\epsilon$

Polynomial time approximation schemes

A Polynomial Time Approximation Scheme (PTAS) for problem P

is a family of algorithms:

For any constant $\epsilon > 0$ there is an algorithm in the family, A_ϵ

such that A_ϵ is a $(1+\epsilon)$-approximation algorithm for P

- If we had a PTAS for Subset Sum we could:

Let $\epsilon = 0.1$ so that $A_{0.1}$ runs in polynomial time and

outputs a subset of size at least $\dfrac{\text{Opt}}{1.1} > 0.9 \cdot \text{Opt}$

A PTAS does not have to have a time complexity which is polynomial in $1/\epsilon$

A fully PTAS (FPTAS) has a time complexity which is polynomial in $1/\epsilon$ (as well as polynomial in n)

Polynomial time approximation schemes

A Polynomial Time Approximation Scheme (PTAS) for problem P

is a family of algorithms:

For any constant $\epsilon > 0$ there is an algorithm in the family, A_ϵ

such that A_ϵ is a $(1+\epsilon)$-approximation algorithm for P

- If we had a PTAS for Subset Sum we could:

Let $\epsilon = 0.1$ so that $A_{0.1}$ runs in polynomial time and

outputs a subset of size at least $\dfrac{\mathrm{Opt}}{1.1} > 0.9 \cdot \mathrm{Opt}$

A PTAS does not have to have a time complexity which is polynomial in $1/\epsilon$

A fully PTAS (FPTAS) has a time complexity which is polynomial in $1/\epsilon$ (as well as polynomial in n)

i.e. the time complexity is $O((n/\epsilon)^c)$ for some constant c

Polynomial time approximation schemes

A Polynomial Time Approximation Scheme (PTAS) for problem P

is a family of algorithms:

For any constant $\epsilon > 0$ there is an algorithm in the family, A_ϵ

such that A_ϵ is a $(1+\epsilon)$-approximation algorithm for P

- If we had a PTAS for Subset Sum we could:

Let $\epsilon = 0.1$ so that $A_{0.1}$ runs in polynomial time and

outputs a subset of size at least $\frac{\text{Opt}}{1.1} > 0.9 \cdot \text{Opt}$

A PTAS does not have to have a time complexity which is polynomial in $1/\epsilon$

A fully PTAS (FPTAS) has a time complexity which is polynomial in $1/\epsilon$ (as well as polynomial in n)

i.e. the time complexity is $O((n/\epsilon)^c)$ for some constant c

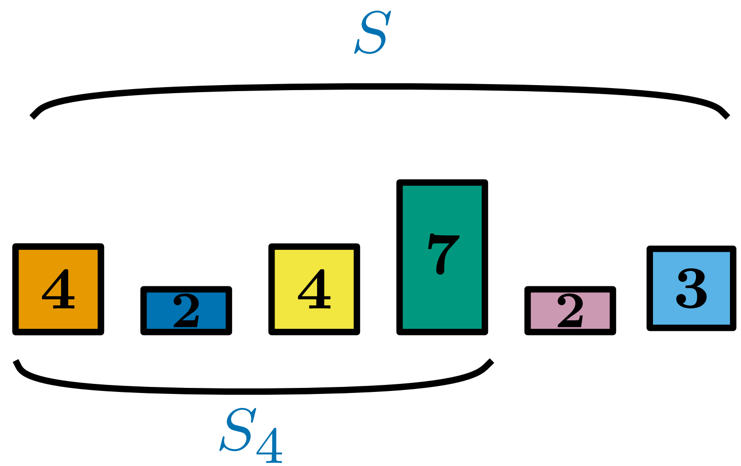
In our example $O((10n)^c) = O((100n)^c) = O((1000n)^c) = O(n^c)$

$\epsilon = 0.1$ $\epsilon = 0.01$ $\epsilon = 0.001$

A PTAS for Subset Sum

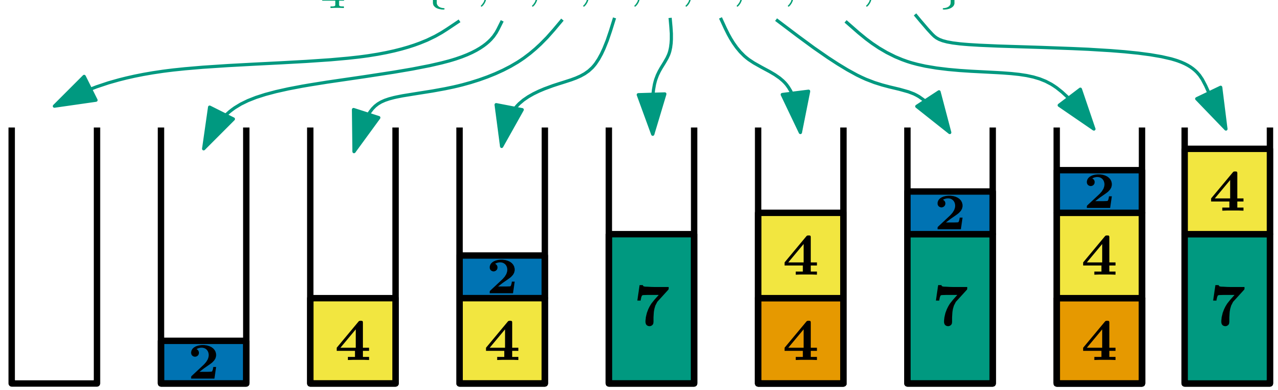
Recall that L_i is the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(where $S_i = \{s_1, s_2, \ldots, s_i\}$ - the first i numbers in the input)



(here $t = 12$)

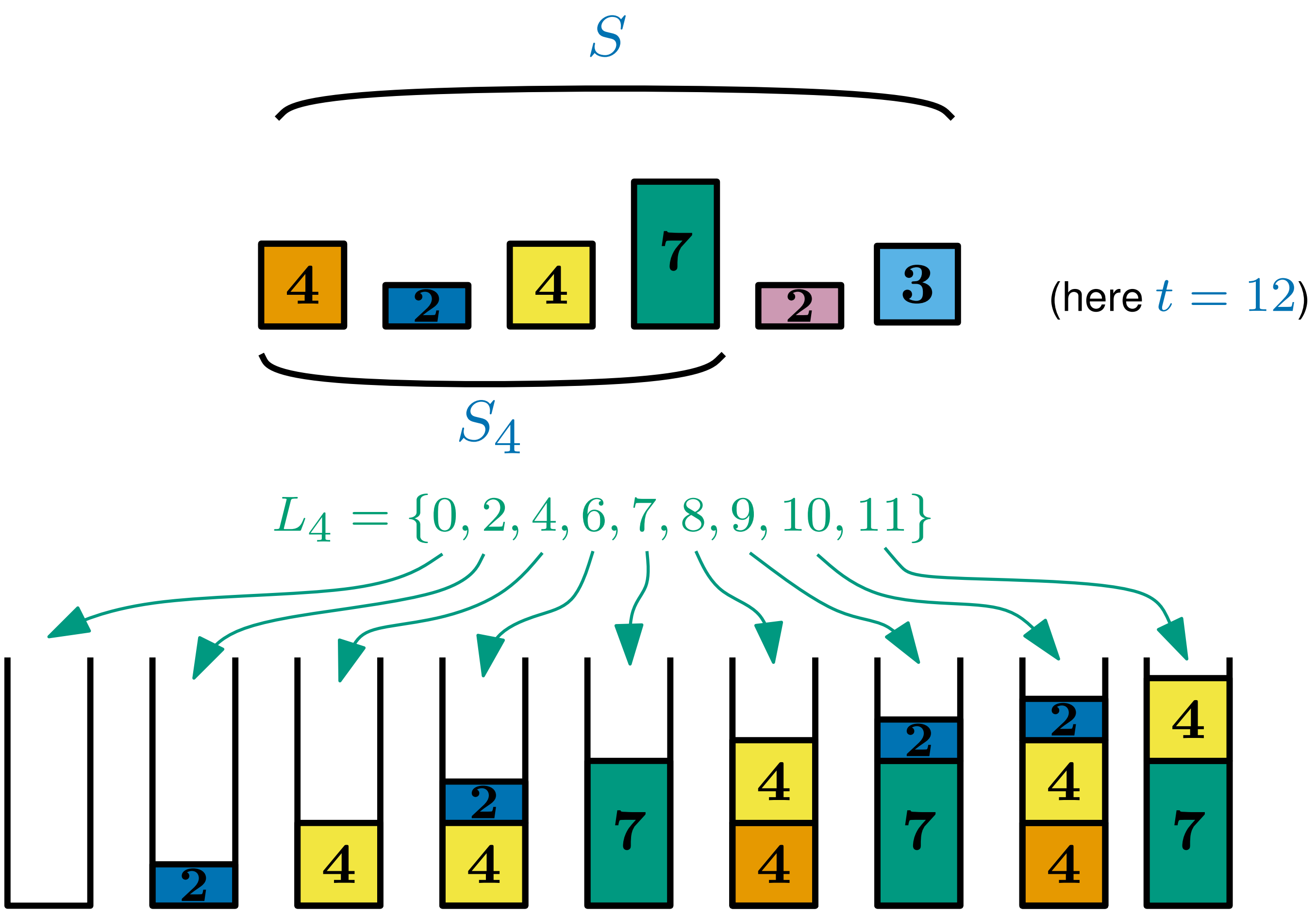
$$L_4 = \{0, 2, 4, 6, 7, 8, 9, 10, 11\}$$



A PTAS for Subset Sum

Recall that L_i is the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(where $S_i = \{s_1, s_2, \ldots, s_i\}$ - the first i numbers in the input)



The exact algorithm for Subset Sum was slow (in general) because each list of possible subset sizes L_i could become *very large*

A PTAS for Subset Sum

Recall that L_i is the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(where $S_i = \{s_1, s_2, \ldots, s_i\}$ - the first i numbers in the input)

Key Idea Construct a *trimmed* version of L_i (denoted $L'_i \subseteq L_i$) so that

A PTAS for Subset Sum

Recall that L_i is the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(where $S_i = \{s_1, s_2, \ldots, s_i\}$ - the first i numbers in the input)

Key Idea Construct a *trimmed* version of L_i (denoted $L'_i \subseteq L_i$) so that

L'_i is a subset of L_i (i.e. $L'_i \subseteq L_i$)

A PTAS for Subset Sum

Recall that L_i is the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(where $S_i = \{s_1, s_2, \ldots, s_i\}$ - the first i numbers in the input)

Key Idea Construct a *trimmed* version of L_i (denoted $L'_i \subseteq L_i$) so that

- L'_i is a subset of L_i (i.e. $L'_i \subseteq L_i$)
- The length of L'_i is polynomial in the input length (i.e. $|L'_i| \leqslant n^c$ for some c)

A PTAS for Subset Sum

Recall that L_i is the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(where $S_i = \{s_1, s_2, \ldots, s_i\}$ - the first i numbers in the input)

Key Idea Construct a *trimmed* version of L_i (denoted $L'_i \subseteq L_i$) so that

- L'_i is a subset of L_i (i.e. $L'_i \subseteq L_i$)
- The length of L'_i is polynomial in the input length (i.e. $|L'_i| \leqslant n^c$ for some c)
- For every $y \in L_i$, there is a $z \in L'_i$ which is *almost* as big

A PTAS for Subset Sum

Recall that L_i is the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(where $S_i = \{s_1, s_2, \ldots, s_i\}$ - the first i numbers in the input)

Key Idea Construct a *trimmed* version of L_i (denoted $L'_i \subseteq L_i$) so that

L'_i is a subset of L_i (i.e. $L'_i \subseteq L_i$)

The length of L'_i is polynomial in the input length (i.e. $|L'_i| \leqslant n^c$ for some c)

For every $y \in L_i$, there is a $z \in L'_i$ which is *almost* as big

Consider this process called **Trim**...

Trim(L_i, δ): Include $L_i[j]$ in L'_i iff

$$L_i[j] > (1+\delta) \cdot \text{prev}$$

where prev is the previous entry we included in L'_i

A PTAS for Subset Sum

Recall that L_i is the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(where $S_i = \{s_1, s_2, \ldots, s_i\}$ - the first i numbers in the input)

Key Idea Construct a *trimmed* version of L_i (denoted $L'_i \subseteq L_i$) so that

L'_i is a subset of L_i (i.e. $L'_i \subseteq L_i$)

The length of L'_i is polynomial in the input length (i.e. $|L'_i| \leqslant n^c$ for some c)

For every $y \in L_i$, there is a $z \in L'_i$ which is *almost* as big

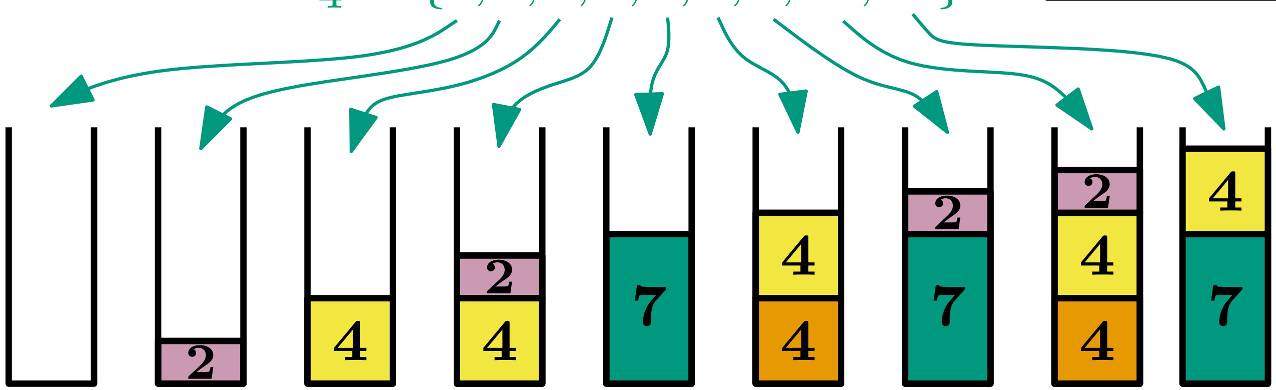
Consider this process called **Trim**. . .

Trim(L_i, δ): Include $L_i[j]$ in L'_i iff

$$L_i[j] > (1 + \delta) \cdot \text{prev}$$

where prev is the previous entry we included in L'_i

$$L_4 = \{0, 2, 4, 6, 7, 8, 9, 10, 11\}$$



A PTAS for Subset Sum

Recall that L_i is the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(where $S_i = \{s_1, s_2, \ldots, s_i\}$ - the first i numbers in the input)

Key Idea Construct a *trimmed* version of L_i (denoted $L'_i \subseteq L_i$) so that

L'_i is a subset of L_i (i.e. $L'_i \subseteq L_i$)

The length of L'_i is polynomial in the input length (i.e. $|L'_i| \leqslant n^c$ for some c)

For every $y \in L_i$, there is a $z \in L'_i$ which is *almost* as big

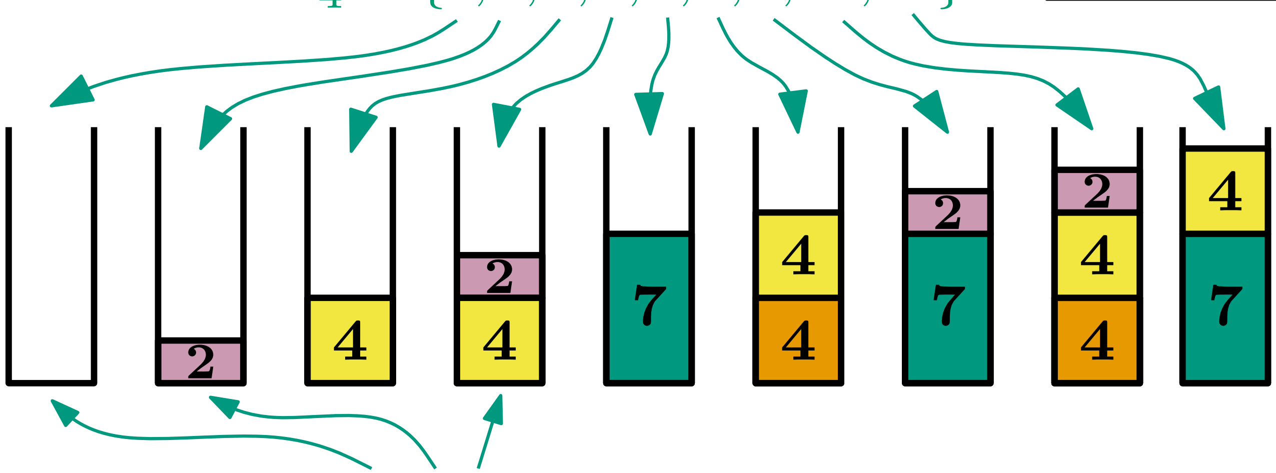
Consider this process called **Trim**. . .

Trim(L_i, δ): Include $L_i[j]$ in L'_i iff

$$L_i[j] > (1 + \delta) \cdot \textsf{prev}$$

where **prev** is the previous entry we included in L'_i

$$L_4 = \{0, 2, 4, 6, 7, 8, 9, 10, 11\}$$



for $\delta = 1 \ldots$ $\quad L'_4 = \{0, 2, 6\}$

A PTAS for Subset Sum

Recall that L_i is the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(where $S_i = \{s_1, s_2, \ldots, s_i\}$ - the first i numbers in the input)

Key Idea Construct a *trimmed* version of L_i (denoted $L'_i \subseteq L_i$) so that

L'_i is a subset of L_i (i.e. $L'_i \subseteq L_i$)

The length of L'_i is polynomial in the input length (i.e. $|L'_i| \leqslant n^c$ for some c)

For every $y \in L_i$, there is a $z \in L'_i$ which is *almost* as big

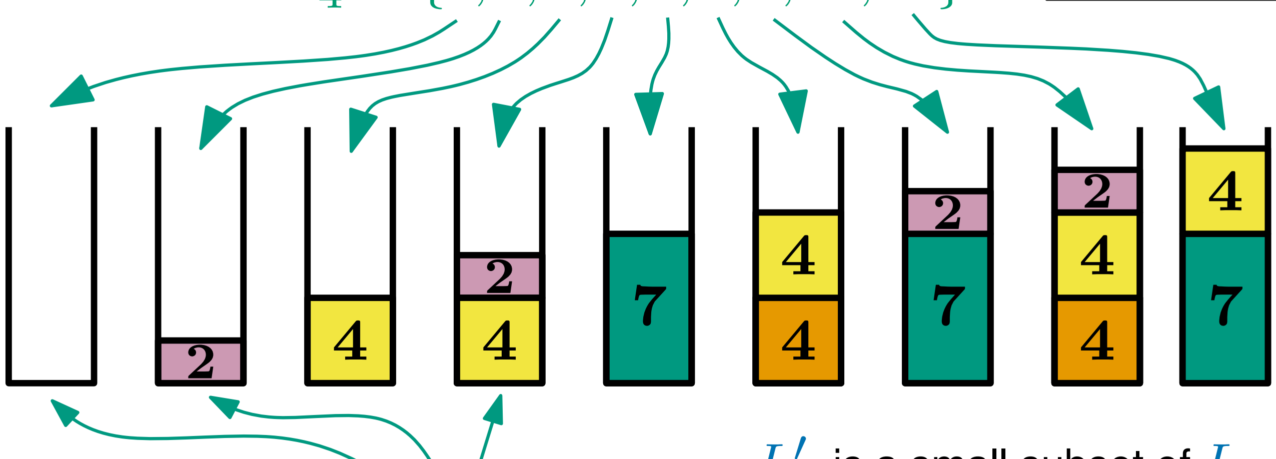
Consider this process called **Trim**...

Trim(L_i, δ): Include $L_i[j]$ in L'_i iff

$$L_i[j] > (1 + \delta) \cdot \text{prev}$$

where prev is the previous entry we included in L'_i

$$L_4 = \{0, 2, 4, 6, 7, 8, 9, 10, 11\}$$



for $\delta = 1\ldots$ $L'_4 = \{0, 2, 6\}$

L'_4 is a small subset of L_4 and for any $y \in L_4$,

there is an $z \in L'_4$ with $z \geqslant y/2$

A PTAS for Subset Sum

Recall that L_i is the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(where $S_i = \{s_1, s_2, \ldots, s_i\}$ - the first i numbers in the input)

Key Idea Construct a *trimmed* version of L_i (denoted $L'_i \subseteq L_i$) so that

L'_i is a subset of L_i (i.e. $L'_i \subseteq L_i$)

The length of L'_i is polynomial in the input length (i.e. $|L'_i| \leqslant n^c$ for some c)

For every $y \in L_i$, there is a $z \in L'_i$ which is *almost* as big

Consider this process called **Trim**. . .

Trim(L_i, δ): Include $L_i[j]$ in L'_i iff

$$L_i[j] > (1 + \delta) \cdot \text{prev}$$

where prev is the previous entry we included in L'_i

A PTAS for Subset Sum

Recall that L_i is the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(where $S_i = \{s_1, s_2, \ldots, s_i\}$ - the first i numbers in the input)

Key Idea Construct a *trimmed* version of L_i (denoted $L'_i \subseteq L_i$) so that

L'_i is a subset of L_i (i.e. $L'_i \subseteq L_i$)

The length of L'_i is polynomial in the input length (i.e. $|L'_i| \leqslant n^c$ for some c)

For every $y \in L_i$, there is a $z \in L'_i$ which is *almost* as big

Consider this process called **Trim**. . .

Trim(L_i, δ): Include $L_i[j]$ in L'_i iff

$$L_i[j] > (1 + \delta) \cdot \text{prev}$$

where prev is the previous entry we included in L'_i

Unfortunately, this hasn't really achieved anything. . .

A PTAS for Subset Sum

Recall that L_i is the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(where $S_i = \{s_1, s_2, \ldots, s_i\}$ - the first i numbers in the input)

Key Idea Construct a *trimmed* version of L_i (denoted $L'_i \subseteq L_i$) so that

L'_i is a subset of L_i (i.e. $L'_i \subseteq L_i$)

The length of L'_i is polynomial in the input length (i.e. $|L'_i| \leqslant n^c$ for some c)

For every $y \in L_i$, there is a $z \in L'_i$ which is *almost* as big

Consider this process called **Trim**...

Trim(L_i, δ): Include $L_i[j]$ in L'_i iff

$$L_i[j] > (1+\delta) \cdot \text{prev}$$

where prev is the previous entry we included in L'_i

Unfortunately, this hasn't really achieved anything...

we don't have time to compute L_i and then **trim** it

(because L_i might be very big)

A PTAS for Subset Sum

Recall that L_i is the set of sizes of all $S' \subseteq S_i$ which are not larger than t

(where $S_i = \{s_1, s_2, \ldots, s_i\}$ - the first i numbers in the input)

Key Idea Construct a *trimmed* version of L_i (denoted $L_i' \subseteq L_i$) so that

- L_i' is a subset of L_i (i.e. $L_i' \subseteq L_i$)
- The length of L_i' is polynomial in the input length (i.e. $|L_i'| \leqslant n^c$ for some c)
- For every $y \in L_i$, there is a $z \in L_i'$ which is *almost* as big

Consider this process called **Trim**...

Trim(L_i, δ): Include $L_i[j]$ in L_i' iff

$$L_i[j] > (1+\delta) \cdot \text{prev}$$

where prev is the previous entry we included in L_i'

Unfortunately, this hasn't really achieved anything...

we don't have time to compute L_i and then **trim** it

(because L_i might be very big)

Instead, we will **trim** as we go along...

$\boxed{|S| = m}$

A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L'_i is the *trimmed* version of L_i

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i = \textbf{Trim}(U, \delta)$
- Output the largest number in L'_m

$|S| = m$

A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L_i' is the *trimmed* version of L_i

The algorithm

- Let $L_0' = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L_{i-1}' + s_i)$ from L_{i-1}'
  - Compute $U = L_{i-1}' \cup (L_{i-1}' + s_i)$
  - Let $L_i' =$ **Trim**(U, δ)
- Output the largest number in L_m'

Trim(U, δ): Include $U[j]$ in L_i' iff $U[j] > (1 + \delta) \cdot$ prev

where prev is the previous thing we included in L_i'

$\boxed{|S| = m}$

A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L'_i is the *trimmed* version of L_i

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i = \textbf{Trim}(U, \delta)$
- Output the largest number in L'_m

$\boxed{|S| = m}$

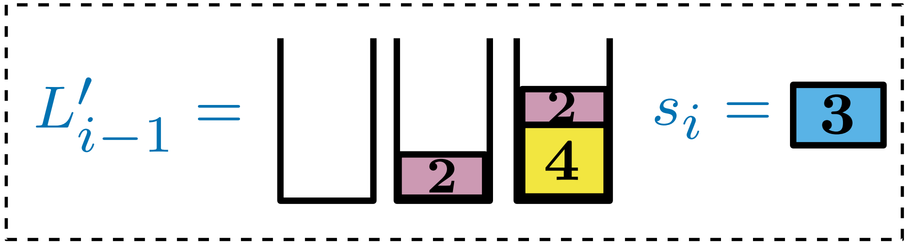
A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L'_i is the *trimmed* version of L_i

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i =$ **Trim**(U, δ)
- Output the largest number in L'_m



$$|S| = m$$

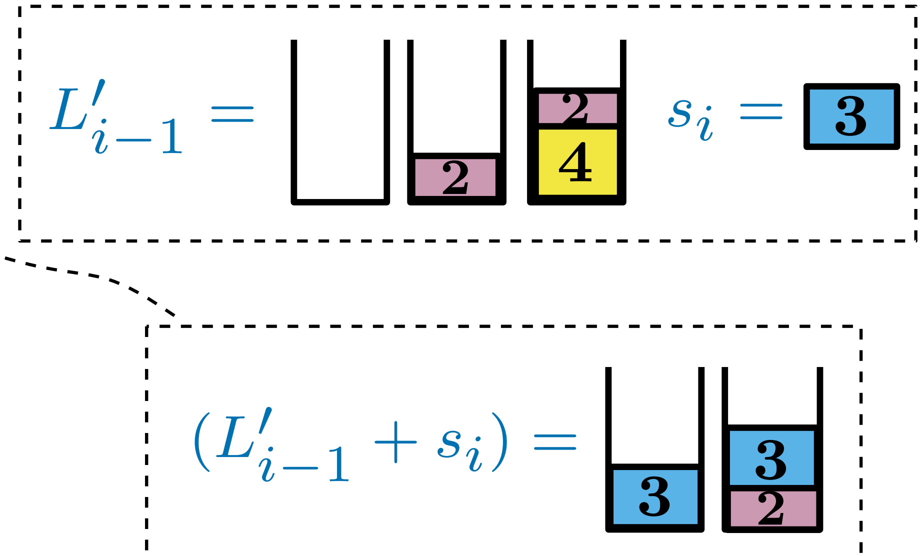
A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L'_i is the *trimmed* version of L_i

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i = \textbf{Trim}(U, \delta)$
- Output the largest number in L'_m



$\boxed{|S| = m}$

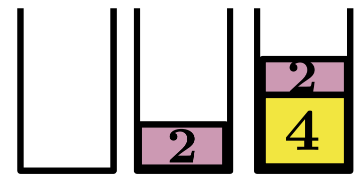
A PTAS for Subset Sum

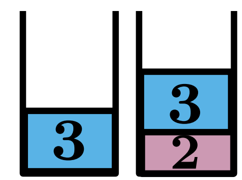
Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

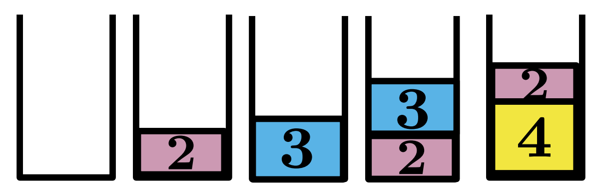
- L'_i is the *trimmed* version of L_i

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i =$ **Trim**(U, δ)
- Output the largest number in L'_m

$L'_{i-1} =$  $s_i =$ 

$(L'_{i-1} + s_i) =$ 

$U = L'_{i-1} \cup (L'_{i-1} + s_i) =$ 

$\boxed{|S| = m}$

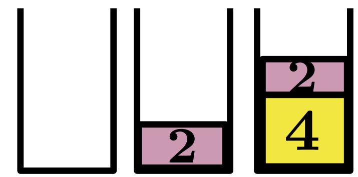
A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

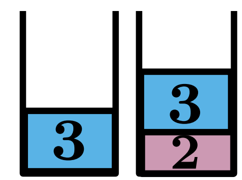
- L'_i is the *trimmed* version of L_i

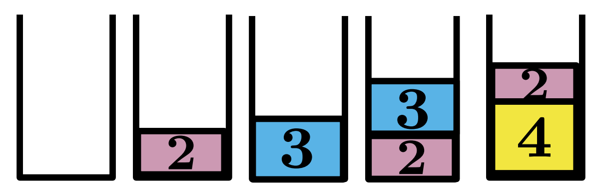
The algorithm

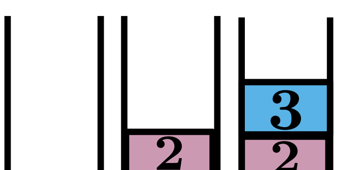
- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i = \textbf{Trim}(U, \delta)$
- Output the largest number in L'_m

$L'_{i-1} =$ 

$s_i =$ 

$(L'_{i-1} + s_i) =$ 

$U = L'_{i-1} \cup (L'_{i-1} + s_i) =$ 



$= L'_i = \textbf{Trim}(U, \delta)$ (with $\delta = 1$)

$\boxed{|S| = m}$

A PTAS for Subset Sum

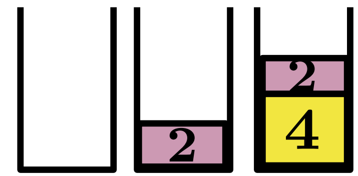
Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L'_i is the *trimmed* version of L_i

The algorithm

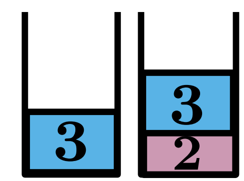
- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \dots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i = \textbf{Trim}(U, \delta)$
- Output the largest number in L'_m

$L'_{i-1} =$

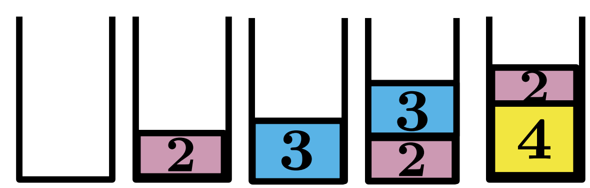


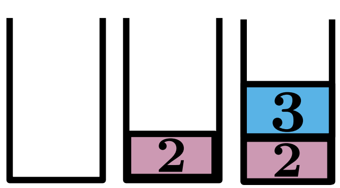
$s_i =$

$(L'_{i-1} + s_i) =$



$U = L'_{i-1} \cup (L'_{i-1} + s_i) =$





$= L'_i = \textbf{Trim}(U, \delta)$ (with $\delta = 1$)

keep each thing if it is more than $(1 + \delta)$ times as big as the last thing you kept

$\boxed{|S| = m}$

A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L'_i is the *trimmed* version of L_i

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i =$ **Trim**(U, δ)
- Output the largest number in L'_m

$|S| = m$

A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L'_i is the *trimmed* version of L_i

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i =$ **Trim**(U, δ)
- Output the largest number in L'_m

$O(|L'_{i-1}|)$ time

$\boxed{|S| = m}$

A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L'_i is the *trimmed* version of L_i

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i =$ **Trim**(U, δ)
- Output the largest number in L'_m

$O(|L'_{i-1}|)$ time

$O(|L'_{i-1}|)$ time

$\boxed{|S| = m}$

A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L'_i is the *trimmed* version of L_i

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i =$ **Trim**(U, δ)
- Output the largest number in L'_m

$O(|L'_{i-1}|)$ time

$O(|L'_{i-1}|)$ time

$O(|L'_i|)$ time

$\boxed{|S| = m}$

A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L'_i is the *trimmed* version of L_i

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \dots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1} — $O(|L'_{i-1}|)$ time
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$ — $O(|L'_{i-1}|)$ time
  - Let $L'_i = \textbf{Trim}(U, \delta)$ — $O(|L'_i|)$ time
- Output the largest number in L'_m

Trim(U, δ): Include $U[j]$ in L'_i iff $U[j] > (1 + \delta) \cdot$ prev

where prev is the previous thing we included in L'_i

$\boxed{|S| = m}$

A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L'_i is the *trimmed* version of L_i

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1} — $O(|L'_{i-1}|)$ time
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$ — $O(|L'_{i-1}|)$ time
  - Let $L'_i = \textbf{Trim}(U, \delta)$ — $O(|L'_i|)$ time
- Output the largest number in L'_m — $O(|L'_m|)$ time

Trim(U, δ): Include $U[j]$ in L'_i iff $U[j] > (1 + \delta) \cdot$ prev

where prev is the previous thing we included in L'_i

$\boxed{|S| = m}$

A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L'_i is the *trimmed* version of L_i

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \dots m$:
    - Compute $(L'_{i-1} + s_i)$ from L'_{i-1} — $O(|L'_{i-1}|)$ time
    - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$ — $O(|L'_{i-1}|)$ time
    - Let $L'_i =$ **Trim**(U, δ) — $O(|L'_i|)$ time
- Output the largest number in L'_m — $O(|L'_m|)$ time

$\boxed{|S| = m}$

A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L_i' is the *trimmed* version of L_i

The algorithm

- Let $L_0' = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L_{i-1}' + s_i)$ from L_{i-1}' — $O(|L_{i-1}'|)$ time
  - Compute $U = L_{i-1}' \cup (L_{i-1}' + s_i)$ — $O(|L_{i-1}'|)$ time
  - Let $L_i' =$ **Trim**(U, δ) — $O(|L_i'|)$ time
- Output the largest number in L_m' — $O(|L_m'|)$ time

This algorithm throws away some possible subsets,

but it always outputs a *valid* subset (but probably not the largest one)

$\boxed{|S| = m}$

A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L_i' is the *trimmed* version of L_i

The algorithm

- Let $L_0' = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L_{i-1}' + s_i)$ from L_{i-1}' — $O(|L_{i-1}'|)$ time
  - Compute $U = L_{i-1}' \cup (L_{i-1}' + s_i)$ — $O(|L_{i-1}'|)$ time
  - Let $L_i' =$ **Trim**(U, δ) — $O(|L_i'|)$ time
- Output the largest number in L_m' — $O(|L_m'|)$ time

This algorithm throws away some possible subsets,

but it always outputs a *valid* subset (but probably not the largest one)

Two questions remain...

$\boxed{|S| = m}$

A PTAS for Subset Sum

Let L_i be the set of sizes of all $S' \subseteq S_i$ which are not larger than t

- L'_i is the *trimmed* version of L_i

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \dots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1} — $O(|L'_{i-1}|)$ time
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$ — $O(|L'_{i-1}|)$ time
  - Let $L'_i = \textbf{Trim}(U, \delta)$ — $O(|L'_i|)$ time
- Output the largest number in L'_m — $O(|L'_m|)$ time

This algorithm throws away some possible subsets,

but it always outputs a *valid* subset (but probably not the largest one)

Two questions remain... How big is $|L'_i|$? How good is the solution given?

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

$|S| = m$

L_i vs. L_i'

Lemma For any $y \in L_i$ there is an $z \in L_i'$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

For any entry in the original set (L_i) ...

there is one in the trimmed set (L_i') ...

of a *'similar'* size (δ is very small)

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

Base Case: $L_0 = L'_0 = \{0\}$

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

Base Case: $L_0 = L'_0 = \{0\}$

Inductive step: Assume that the lemma holds for $(i-1)$

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

Base Case: $L_0 = L'_0 = \{0\}$

Inductive step: Assume that the lemma holds for $(i-1)$

As $y \in L_i$ we have that either $y \in L_{i-1}$ or $(y - s_i) \in L_{i-1}$

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

Base Case: $L_0 = L'_0 = \{0\}$

Inductive step: Assume that the lemma holds for $(i-1)$

As $y \in L_i$ we have that either $y \in L_{i-1}$ or $(y - s_i) \in L_{i-1}$

if $y \in L_{i-1}$ then there is a $x \in L'_{i-1}$ with $\frac{y}{(1+\delta)^{(i-1)}} \leqslant x \leqslant y$

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

by the inductive hypothesis

Base Case: $L_0 = L'_0 = \{0\}$

Inductive step: Assume that the lemma holds for $(i-1)$

As $y \in L_i$ we have that either $y \in L_{i-1}$ or $(y - s_i) \in L_{i-1}$

if $y \in L_{i-1}$ then there is a $x \in L'_{i-1}$ with $\frac{y}{(1+\delta)^{(i-1)}} \leqslant x \leqslant y$

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

Base Case: $L_0 = L'_0 = \{0\}$

Inductive step: Assume that the lemma holds for $(i-1)$

As $y \in L_i$ we have that either $y \in L_{i-1}$ or $(y - s_i) \in L_{i-1}$

if $y \in L_{i-1}$ then there is a $x \in L'_{i-1}$ with $\frac{y}{(1+\delta)^{(i-1)}} \leqslant x \leqslant y$

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

Base Case: $L_0 = L'_0 = \{0\}$

Inductive step: Assume that the lemma holds for $(i-1)$

As $y \in L_i$ we have that either $y \in L_{i-1}$ or $(y - s_i) \in L_{i-1}$

if $y \in L_{i-1}$ then there is a $x \in L'_{i-1}$ with $\frac{y}{(1+\delta)^{(i-1)}} \leqslant x \leqslant y$

By the definition of Trim there is some $z \in L'_i$ with $z \leqslant x \leqslant z \cdot (1+\delta)$

$|S| = m$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

Base Case: $L_0 = L'_0 = \{0\}$

Inductive step: Assume that the lemma holds for $(i-1)$

As $y \in L_i$ we have that either $y \in L_{i-1}$ or $(y - s_i) \in L_{i-1}$

if $y \in L_{i-1}$ then there is a $x \in L'_{i-1}$ with $\frac{y}{(1+\delta)^{(i-1)}} \leqslant x \leqslant y$

By the definition of Trim there is some $z \in L'_i$ with $z \leqslant x \leqslant z \cdot (1+\delta)$

So we have that $z \leqslant x \leqslant y$ and $z \geqslant \frac{x}{1+\delta} \geqslant \frac{y}{(1+\delta)^i}$

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

Base Case: $L_0 = L'_0 = \{0\}$

Inductive step: Assume that the lemma holds for $(i-1)$

As $y \in L_i$ we have that either $y \in L_{i-1}$ or $(y - s_i) \in L_{i-1}$

if $y \in L_{i-1}$ then there is a $x \in L'_{i-1}$ with $\frac{y}{(1+\delta)^{(i-1)}} \leqslant x \leqslant y$

By the definition of Trim there is some $z \in L'_i$ with $z \leqslant x \leqslant z \cdot (1+\delta)$

So we have that $z \leqslant x \leqslant y$ and $z \geqslant \frac{x}{1+\delta} \geqslant \frac{y}{(1+\delta)^i}$

$|S| = m$

University of BRISTOL

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

Base Case: $L_0 = L'_0 = \{0\}$

Inductive step: Assume that the lemma holds for $(i-1)$

As $y \in L_i$ we have that either $y \in L_{i-1}$ or $(y - s_i) \in L_{i-1}$

if $y \in L_{i-1}$ then there is a $x \in L'_{i-1}$ with $\frac{y}{(1+\delta)^{(i-1)}} \leqslant x \leqslant y$

By the definition of Trim there is some $z \in L'_i$ with $z \leqslant x \leqslant z \cdot (1+\delta)$

So we have that $z \leqslant x \leqslant y$ and $z \geqslant \frac{x}{1+\delta} \geqslant \frac{y}{(1+\delta)^i}$

$|S| = m$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

Base Case: $L_0 = L'_0 = \{0\}$

Inductive step: Assume that the lemma holds for $(i-1)$

As $y \in L_i$ we have that either $y \in L_{i-1}$ or $(y - s_i) \in L_{i-1}$

if $y \in L_{i-1}$ then there is a $x \in L'_{i-1}$ with $\frac{y}{(1+\delta)^{(i-1)}} \leqslant x \leqslant y$

By the definition of Trim there is some $z \in L'_i$ with $z \leqslant x \leqslant z \cdot (1+\delta)$

So we have that $z \leqslant x \leqslant y$ and $z \geqslant \frac{x}{1+\delta} \geqslant \frac{y}{(1+\delta)^i}$

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

Base Case: $L_0 = L'_0 = \{0\}$

Inductive step: Assume that the lemma holds for $(i-1)$

As $y \in L_i$ we have that either $y \in L_{i-1}$ or $(y - s_i) \in L_{i-1}$

if $y \in L_{i-1}$ then there is a $x \in L'_{i-1}$ with $\frac{y}{(1+\delta)^{(i-1)}} \leqslant x \leqslant y$

By the definition of Trim there is some $z \in L'_i$ with $z \leqslant x \leqslant z \cdot (1+\delta)$

So we have that $z \leqslant x \leqslant y$ and $z \geqslant \frac{x}{1+\delta} \geqslant \frac{y}{(1+\delta)^i}$

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

Base Case: $L_0 = L'_0 = \{0\}$

Inductive step: Assume that the lemma holds for $(i-1)$

As $y \in L_i$ we have that either $y \in L_{i-1}$ or $(y - s_i) \in L_{i-1}$

if $y \in L_{i-1}$ then there is a $x \in L'_{i-1}$ with $\frac{y}{(1+\delta)^{(i-1)}} \leqslant x \leqslant y$

By the definition of Trim there is some $z \in L'_i$ with $z \leqslant x \leqslant z \cdot (1+\delta)$

So we have that $z \leqslant x \leqslant y$ and $z \geqslant \frac{x}{1+\delta} \geqslant \frac{y}{(1+\delta)^i}$

$\boxed{|S| = m}$

L_i vs. L_i'

Lemma For any $y \in L_i$ there is an $z \in L_i'$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

Base Case: $L_0 = L_0' = \{0\}$

Inductive step: Assume that the lemma holds for $(i-1)$

As $y \in L_i$ we have that either $y \in L_{i-1}$ or $(y - s_i) \in L_{i-1}$

if $y \in L_{i-1}$ then there is a $x \in L_{i-1}'$ with $\frac{y}{(1+\delta)^{(i-1)}} \leqslant x \leqslant y$

By the definition of Trim there is some $z \in L_i'$ with $z \leqslant x \leqslant z \cdot (1+\delta)$

So we have that $z \leqslant x \leqslant y$ and $z \geqslant \frac{x}{1+\delta} \geqslant \frac{y}{(1+\delta)^i}$

I.e. that there is an $z \in L_i'$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$ as required

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

Base Case: $L_0 = L'_0 = \{0\}$

Inductive step: Assume that the lemma holds for $(i-1)$

As $y \in L_i$ we have that either $y \in L_{i-1}$ or $(y - s_i) \in L_{i-1}$

if $y \in L_{i-1}$ then there is a $x \in L'_{i-1}$ with $\frac{y}{(1+\delta)^{(i-1)}} \leqslant x \leqslant y$

By the definition of Trim there is some $z \in L'_i$ with $z \leqslant x \leqslant z \cdot (1+\delta)$

So we have that $z \leqslant x \leqslant y$ and $z \geqslant \frac{x}{1+\delta} \geqslant \frac{y}{(1+\delta)^i}$

I.e. that there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$ as required

The case that $(y - s_i) \in L_{i-1}$ is almost identical

$|S| = m$

L_i vs. L_i'

Lemma For any $y \in L_i$ there is an $z \in L_i'$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

Proof (by induction)

Base Case: $L_0 = L_0' = \{0\}$

Inductive step: Assume that the lemma holds for $(i-1)$

As $y \in L_i$ we have that either $y \in L_{i-1}$ or $(y - s_i) \in L_{i-1}$

if $y \in L_{i-1}$ then there is a $x \in L_{i-1}'$ with $\frac{y}{(1+\delta)^{(i-1)}} \leqslant x \leqslant y$

By the definition of Trim there is some $z \in L_i'$ with $z \leqslant x \leqslant z \cdot (1+\delta)$

So we have that $z \leqslant x \leqslant y$ and $z \geqslant \frac{x}{1+\delta} \geqslant \frac{y}{(1+\delta)^i}$

I.e. that there is an $z \in L_i'$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$ as required

The case that $(y - s_i) \in L_{i-1}$ is almost identical *(we omit it for brevity)*

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

By setting $i = m$ and $\delta = \epsilon/2m$ we have that,

For any $y \in L_m$ there is a $z \in L'_m$ with $\frac{y}{(1 + \frac{\epsilon}{2m})^m} \leqslant z \leqslant y$

$\boxed{|S| = m}$

L_i vs. L_i'

Lemma For any $y \in L_i$ there is an $z \in L_i'$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

By setting $i = m$ and $\delta = \epsilon/2m$ we have that,

For any $y \in L_m$ there is a $z \in L_m'$ with $\frac{y}{(1 + \frac{\epsilon}{2m})^m} \leqslant z \leqslant y$

Further, $\mathrm{Opt} \in L_m$ meaning there is a $z \in L_m'$ with

$$\frac{\mathrm{Opt}}{(1 + \frac{\epsilon}{2m})^m} \leqslant z \leqslant \mathrm{Opt}$$

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

By setting $i = m$ and $\delta = \epsilon/2m$ we have that,

For any $y \in L_m$ there is a $z \in L'_m$ with $$\frac{y}{(1 + \frac{\epsilon}{2m})^m} \leqslant z \leqslant y$$

Further, $\mathrm{Opt} \in L_m$ meaning there is a $z \in L'_m$ with

$$\frac{\mathrm{Opt}}{\left(1 + \frac{\epsilon}{2m}\right)^m} \leqslant z \leqslant \mathrm{Opt}$$

Recall that the output of the algorithm is the largest number in $L'_m \ldots$

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

By setting $i = m$ and $\delta = \epsilon/2m$ we have that,

For any $y \in L_m$ there is a $z \in L'_m$ with $\frac{y}{(1+\frac{\epsilon}{2m})^m} \leqslant z \leqslant y$

Further, $\mathrm{Opt} \in L_m$ meaning there is a $z \in L'_m$ with

$$\frac{\mathrm{Opt}}{\left(1+\frac{\epsilon}{2m}\right)^m} \leqslant z \leqslant \mathrm{Opt}$$

Recall that the output of the algorithm is the largest number in L'_m...

We only need to show that $\left(1+\frac{\epsilon}{2m}\right)^m \leqslant 1+\epsilon \ldots$

$\boxed{|S| = m}$

L_i vs. L'_i

Lemma For any $y \in L_i$ there is an $z \in L'_i$ with $\frac{y}{(1+\delta)^i} \leqslant z \leqslant y$

By setting $i = m$ and $\delta = \epsilon/2m$ we have that,

For any $y \in L_m$ there is a $z \in L'_m$ with $\frac{y}{(1 + \frac{\epsilon}{2m})^m} \leqslant z \leqslant y$

Further, $\mathrm{Opt} \in L_m$ meaning there is a $z \in L'_m$ with

$$\frac{\mathrm{Opt}}{1+\epsilon} \leqslant z \leqslant \mathrm{Opt} \quad \textbf{VS} \quad \frac{\mathrm{Opt}}{(1 + \frac{\epsilon}{2m})^m} \leqslant z \leqslant \mathrm{Opt}$$

Recall that the output of the algorithm is the largest number in $L'_m \ldots$

We only need to show that $\left(1 + \frac{\epsilon}{2m}\right)^m \leqslant 1 + \epsilon \ldots$

$\boxed{|S| = m}$

L_i vs. L'_i

We need to show that $\left(1+\frac{\epsilon}{2m}\right)^m \leqslant 1+\epsilon$ (for $0 < \epsilon \leqslant 1$)

$\boxed{|S| = m}$

L_i vs. L_i'

We need to show that $(1 + \frac{\epsilon}{2m})^m \leqslant 1 + \epsilon$ (for $0 < \epsilon \leqslant 1$)

$$\left(1 + \frac{\epsilon}{2m}\right)^m \leqslant e^{\epsilon/2} \leqslant 1 + \frac{\epsilon}{2} + \left(\frac{\epsilon}{2}\right)^2 \leqslant 1 + \epsilon$$

$\boxed{|S| = m}$

L_i vs. L'_i

We need to show that $(1+\frac{\epsilon}{2m})^m \leqslant 1+\epsilon$ (for $0 < \epsilon \leqslant 1$)

$$\left(1+\frac{\epsilon}{2m}\right)^m \leqslant e^{\epsilon/2} \leqslant 1+\frac{\epsilon}{2}+\left(\frac{\epsilon}{2}\right)^2 \leqslant 1+\epsilon$$

This follows from the following facts:

$e^x \geqslant (1+\frac{x}{m})^m$ for all $x, m > 0$

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} \leqslant 1+x+x^2$$

$\boxed{|S| = m}$

L_i vs. L'_i

We need to show that $(1 + \frac{\epsilon}{2m})^m \leqslant 1 + \epsilon$ (for $0 < \epsilon \leqslant 1$)

$$\left(1 + \frac{\epsilon}{2m}\right)^m \leqslant e^{\epsilon/2} \leqslant 1+\frac{\epsilon}{2}+\left(\frac{\epsilon}{2}\right)^2 \leqslant 1+\epsilon$$

So the output of the algorithm is some z where,

$\boxed{|S| = m}$

L_i vs. L_i'

We need to show that $\left(1 + \frac{\epsilon}{2m}\right)^m \leqslant 1 + \epsilon$ (for $0 < \epsilon \leqslant 1$)

$$\left(1 + \frac{\epsilon}{2m}\right)^m \leqslant e^{\epsilon/2} \leqslant 1+\frac{\epsilon}{2}+\left(\frac{\epsilon}{2}\right)^2 \leqslant 1+\epsilon$$

So the output of the algorithm is some z where,

$$\frac{\mathrm{Opt}}{1+\epsilon} \leqslant \frac{\mathrm{Opt}}{\left(1 + \frac{\epsilon}{2m}\right)^m} \leqslant z \leqslant \mathrm{Opt}$$

$\boxed{|S| = m}$

L_i vs. L'_i

We need to show that $(1 + \frac{\epsilon}{2m})^m \leqslant 1 + \epsilon$ (for $0 < \epsilon \leqslant 1$)

$$\left(1 + \frac{\epsilon}{2m}\right)^m \leqslant e^{\epsilon/2} \leqslant 1+\frac{\epsilon}{2}+\left(\frac{\epsilon}{2}\right)^2 \leqslant 1+\epsilon$$

So the output of the algorithm is some z where,

$$\frac{\text{Opt}}{1+\epsilon} \leqslant z \leqslant \text{Opt}$$

$\boxed{|S| = m}$

L_i vs. L'_i

We need to show that $(1 + \frac{\epsilon}{2m})^m \leqslant 1 + \epsilon$ (for $0 < \epsilon \leqslant 1$)

$$\left(1 + \frac{\epsilon}{2m}\right)^m \leqslant e^{\epsilon/2} \leqslant 1 + \frac{\epsilon}{2} + \left(\frac{\epsilon}{2}\right)^2 \leqslant 1 + \epsilon$$

So the output of the algorithm is some z where,

$$\frac{\mathrm{Opt}}{1 + \epsilon} \leqslant z \leqslant \mathrm{Opt}$$

But how long does it take to run?

$\boxed{|S| = m}$

How big is L'_i?

The time complexity depends on $|L'_i|$...

$\boxed{|S| = m}$

How big is L'_i?

The time complexity depends on $|L'_i| \ldots$

By the definition of Trim we have that,

any two successive elements, z, z' of L'_i have

$$\frac{z'}{z} \geqslant 1 + \delta = 1 + \frac{\epsilon}{2m}$$

$\boxed{|S| = m}$

How big is L_i'?

The time complexity depends on $|L_i'| \ldots$

By the definition of Trim we have that,

any two successive elements, z, z' of L_i' have

$$\frac{z'}{z} \geqslant 1 + \delta = 1 + \frac{\epsilon}{2m}$$

Further, all elements are no greater than t

$\boxed{|S| = m}$

How big is L_i'?

The time complexity depends on $|L_i'|\ldots$

By the definition of Trim we have that,

any two successive elements, z, z' of L_i' have

$$\frac{z'}{z} \geqslant 1 + \delta = 1 + \frac{\epsilon}{2m}$$

Further, all elements are no greater than t

So L_i' contains at most $O(\log_{(1+\delta)} t)$ elements

$\boxed{|S| = m}$

How big is L_i'?

The time complexity depends on $|L_i'|\ldots$

By the definition of Trim we have that,

any two successive elements, z, z' of L_i' have

$$\frac{z'}{z} \geqslant 1 + \delta = 1 + \frac{\epsilon}{2m}$$

Further, all elements are no greater than t

So L_i' contains at most $O(\log_{(1+\delta)} t)$ elements

$$\log_{(1+\delta)} t = \frac{\ln t}{\ln(1 + (\epsilon/2m))} \leqslant \frac{2m(1 + (\epsilon/2m)) \ln t}{\epsilon} = O\left(\frac{m \log t}{\epsilon}\right)$$

$\boxed{|S| = m}$

How big is L_i'?

The time complexity depends on $|L_i'|\ldots$

By the definition of Trim we have that,

any two successive elements, z, z' of L_i' have

$$\frac{z'}{z} \geqslant 1 + \delta = 1 + \frac{\epsilon}{2m}$$

Further, all elements are no greater than t

another fact:

$$\ln(1 + x) > \frac{x}{x+1}$$

(here $x = \epsilon/2m$)

So L_i' contains at most $O(\log_{(1+\delta)} t)$ elements

$$\log_{(1+\delta)} t = \frac{\ln t}{\ln(1 + (\epsilon/2m))} \leqslant \frac{2m(1 + (\epsilon/2m)) \ln t}{\epsilon} = O\left(\frac{m \log t}{\epsilon}\right)$$

A PTAS for Subset Sum

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i =$ **Trim**(U, δ)
- Output the largest number in L'_m

$O(|L'_{i-1}|)$ time

$O(|L'_i|)$ time

$O(|L'_i|)$ time

$O(|L'_m|)$ time

$\boxed{|S| = m}$

A PTAS for Subset Sum

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \dots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i = \textbf{Trim}(U, \delta)$
- Output the largest number in L'_m

$O(|L'_{i-1}|)$ time

$O(|L'_i|)$ time

$O(|L'_i|)$ time

$O(|L'_m|)$ time

As $|L'_i| = O(m \log t/\epsilon)$, the algorithm runs in

$$O(m^2 \log t/\epsilon) = O(n^3 \log n/\epsilon) \text{ time}$$

$\boxed{|S| = m}$

A PTAS for Subset Sum

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i =$ **Trim**(U, δ)
- Output the largest number in L'_m

$O(|L'_{i-1}|)$ time

$O(|L'_i|)$ time

$O(|L'_i|)$ time

$O(|L'_m|)$ time

As $|L'_i| = O(m \log t/\epsilon)$, the algorithm runs in

$$O(m^2 \log t/\epsilon) = O(n^3 \log n/\epsilon) \text{ time}$$

$m \leqslant n$

$\log t = O(n \log n)$

Recall that n is the *length of the input* (measured in words)

$\boxed{|S| = m}$

A PTAS for Subset Sum

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i =$ **Trim**(U, δ)
- Output the largest number in L'_m

$O(|L'_{i-1}|)$ time

$O(|L'_i|)$ time

$O(|L'_i|)$ time

$O(|L'_m|)$ time

As $|L'_i| = O(m \log t/\epsilon)$, the algorithm runs in

$$O(m^2 \log t/\epsilon) = O(n^3 \log n/\epsilon) \text{ time}$$

$\boxed{|S| = m}$

A PTAS for Subset Sum

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i =$ **Trim**(U, δ)
- Output the largest number in L'_m

$O(|L'_{i-1}|)$ time

$O(|L'_i|)$ time

$O(|L'_i|)$ time

$O(|L'_m|)$ time

As $|L'_i| = O(m \log t/\epsilon)$, the algorithm runs in

$$O(m^2 \log t/\epsilon) = O(n^3 \log n/\epsilon) \text{ time}$$

The output z is such that $\dfrac{\text{Opt}}{1+\epsilon} \leqslant z \leqslant \text{Opt}$

$\boxed{|S| = m}$

A PTAS for Subset Sum

The algorithm

- Let $L'_0 = \{0\}$, $\delta = \epsilon/(2m)$
- For $i = 1 \ldots m$:
  - Compute $(L'_{i-1} + s_i)$ from L'_{i-1}
  - Compute $U = L'_{i-1} \cup (L'_{i-1} + s_i)$
  - Let $L'_i =$ **Trim**(U, δ)
- Output the largest number in L'_m

$O(|L'_{i-1}|)$ time

$O(|L'_i|)$ time

$O(|L'_i|)$ time

$O(|L'_m|)$ time

As $|L'_i| = O(m \log t/\epsilon)$, the algorithm runs in

$$O(m^2 \log t/\epsilon) = O(n^3 \log n/\epsilon) \text{ time}$$

The output z is such that $\dfrac{\text{Opt}}{1+\epsilon} \leqslant z \leqslant \text{Opt}$

So this is in fact an FPTAS for Subset Sum

Polynomial time approximation schemes

A Polynomial Time Approximation Scheme (PTAS) for problem P is a family of algorithms:

For any constant $\epsilon > 0$ there is an algorithm in the family, A_ϵ

such that A_ϵ is a $(1+\epsilon)$-approximation algorithm for P

We have seen an FPTAS for Subset Sum

which runs in $O(n^3 \log n/\epsilon)$ time

The output z is such that $\dfrac{\mathrm{Opt}}{1+\epsilon} \leqslant z \leqslant \mathrm{Opt}$

A PTAS does not have to have a time complexity which is polynomial in $1/\epsilon$

e.g. the time complexity could be $O(n^{\frac{c}{\epsilon}})$ (for example)

A fully PTAS (FPTAS) has a time complexity which is polynomial in $1/\epsilon$ (as well as polynomial in n)

i.e. the time complexity is $O((n/\epsilon)^c)$ for some constant c