Data Structures and Algorithms

(**CS210A**)

Semester I – **2014-15**

Lecture 1:

- An **overview** and **motivation** for the course
- some **concrete** examples.

The website of the course

moodle.cse.iitk.ac.in

→Courses

→CS210: Data Structures and Algorithms

(guest login allowed)

Close your notebook. (Slides will be provided for each lecture)

Let us start the course with fresh mind.

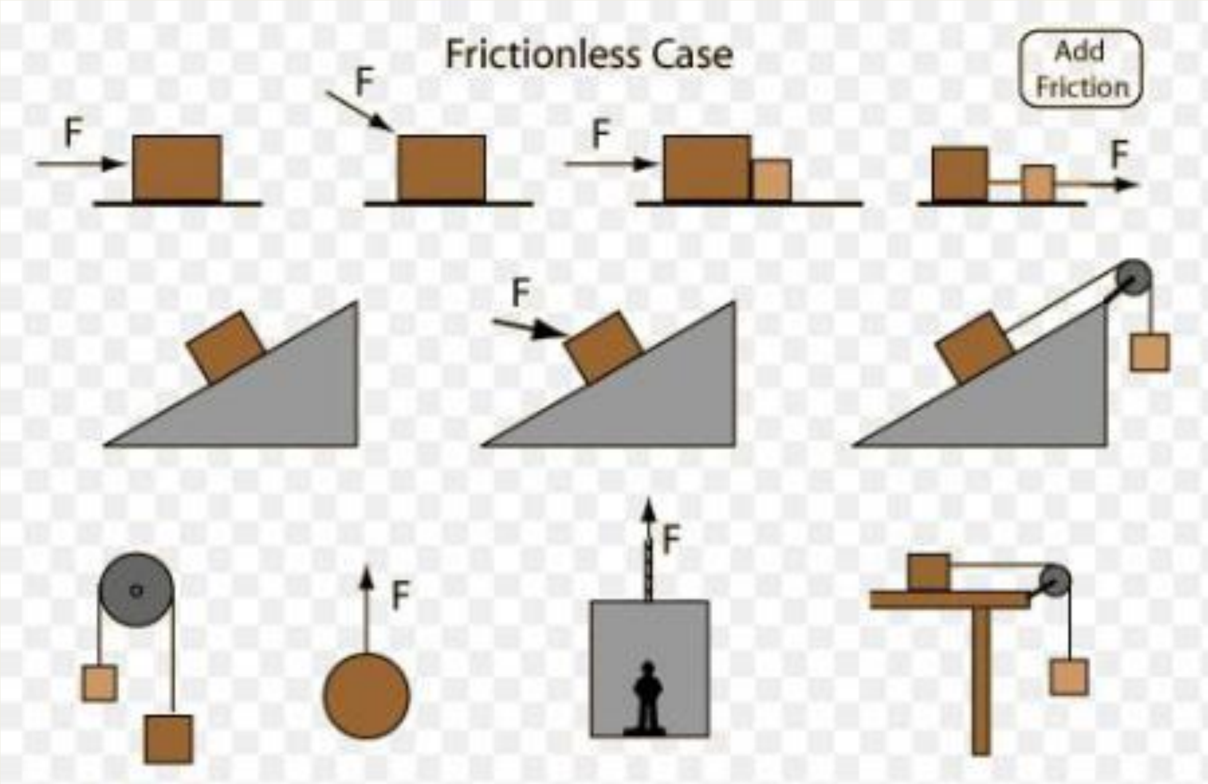
Prerequisite of this course

- A good command on Programming in C
  - Programs involving arrays
  - Recursion
  - Linked lists (preferred)

- **Fascination for solving Puzzles**

This course will be taught **differently.**

Recall your JEE preparation days



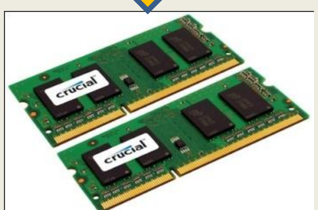
What is this difference ?

- **We** shall **re-invent** every concept in the class itself.
- We shall **solve each problem in the class** through discussion.
- You will realize that solution will emerge naturally if we ask **right set of questions** and then try to find their **answers**.
- Most importantly we shall so everything **together**.

… so that finally it is a concept/solution derived by yourself and not concept from some scientist/book/teacher.

Isn’t that nice ☺

Let us open your desktop



A processor (CPU)

speed = few GHz

(a few **nanoseconds** to execute an instruction)

Internal memory (RAM)

size = a few GB (Stores few million bytes/words)

speed = a few GHz(a few **nanoseconds** to read a byte/word)

External Memory (Hard Disk Drive)

size = a few tera bytes

speed : seek time = **miliseconds**

transfer rate= around **billion** bits per second

A simplifying assumption for the rest of the lecture

It takes around a few **nanoseconds** to execute an instruction.

*(This assumption is **well supported** by the modern day computers)*

EFFICIENT ALGORITHMS

What is an algorithm ?

Definition: A **finite sequence** of **well defined instructions** required to solve a given computational problem.

(We shall see more precise definition soon)

WHY DO WE CARE FOR EFFICIENT ALGORITHMS WHEN WE HAVE PROCESSORS RUNNING AT GIGAHERTZ?

Revisiting problems from ESC101

Problem 1:
Bit-sum-prime numbers

Definition: A positive integer is said to be **bit-sum-prime** if the sum of its bits is a prime number.

Examples: **6** (110) — is **bit-sum prime.**

7 (111) — is **bit-sum prime.**

29 (11101) — is **not bit-sum prime.**

Algorithmic problem:

Input: positive integer n,

Output: the count of all bit-sum-prime numbers less than n.

Homework 1: Write a **C** program for **bit-sum prime** problem with n as **long long int** (64 bit integer), and execute it for some large value of n.

For example, execute the program for n **=123456789123456789.**

Problem 2:
Fibonacci numbers

Fibonacci numbers

$F(0) = 0;$

$F(1) = 1;$

$F(n) = F(n-1) + F(n-2)$ for all $n > 1;$

Exercise : Using induction or otherwise, show that $F(n) > 2^{\frac{n-2}{2}}$

Algorithms you must have implemented for computing $F(n)$:

- **Iterative**
- **recursive**

Iterative Algorithm for F(n)

IFib(n)

```
if n=0 return 0;
    else if n=1 return 1;
        else {      a ← 0;  b← 1;
                    For(i=2 to n) do
                    {    temp ← b;
                         b← a+b;
                         a← temp;
                     }
              }
  return b;
```

Recursive algorithm for F(n)

```
Rfib(n)
{   if n=0 return 0;
      else if n=1 return 1;
          else return(Rfib(n-1) + Rfib(n-2))
}
```

Problem 2:
Fibonacci numbers

Homework 2: Write a program for the following problem:

Input: Two numbers n,m (**long long int** (64 bit integer)),

Output: **F(n) mod m**

Experimentally find the range of numbers n and m for which your (iterative and recursive) programs takes less than a minute. The results will be very disappointing ☹

Interestingly, there is an algorithm that will run in **a few micro seconds** for the whole range of n and m. We shall discuss it soon in some class.

Problem 3:
Subset-sum problem

Input: An array **A** storing **n** numbers, and a number **s**

Output: Determine if there is a subset of numbers from **A** whose sum is **s**.

The fastest existing algorithm has to execute $2^{n/2}$ instructions.
Hence, on the fastest existing computer, it will take

- **At least an year** for **n=100**
- **At least 1000 years** for **n=120**

Problem 4:
Sorting

Input: An array **A** storing **n** numbers.

Output: Sorted **A**

A fact:

A significant fraction of the code of all the software is for sorting or searching only.

To sort **10 million** numbers on the present day computers

- **Selection sort** will take at least a few hours.
- **Merge sort** will take only a few seconds.

How to design efficient algorithm for a problem ?

Design of algorithms and data structures is also an Art

Requires:

- **Creativity**
- **Hard work**
- **Practice**
- **Perseverance** (most important)

Summary of Algorithms

- There are many practically relevant problems for which there does not exist any efficient algorithm till date.

- Efficient algorithms are important for theoretical as well as practical purposes.

- Algorithm design is an art which demands a lot of creativity, intuition, and perseverance.

- More and more applications in real life require efficient algorithms
  - Search engines like **Google** exploits many clever algorithms.

THE DATA STRUCTURES

An Example

Given: a telephone directory storing telephone no. of **hundred million** persons.

Aim: to answer a sequence of **queries** of the form

"what is the phone number of a given person ?".

Solution 1 :

Keep the directory in an array.

do **sequential search** for each query.

Time per query: around **1/10th** of a **second**

Solution 2:

Keep the directory in an array, and **sort it** according to names,

do **binary search** for each query.

Time per query: less than **100 nanoseconds**

Aim of data structure ?

To store/organize a given data in the memory of computer so that each subsequent operation (query/update) can be performed quickly ?

Range-Minima Problem

A Motivating example

to realize the importance of data structures

Range-Minima Problem

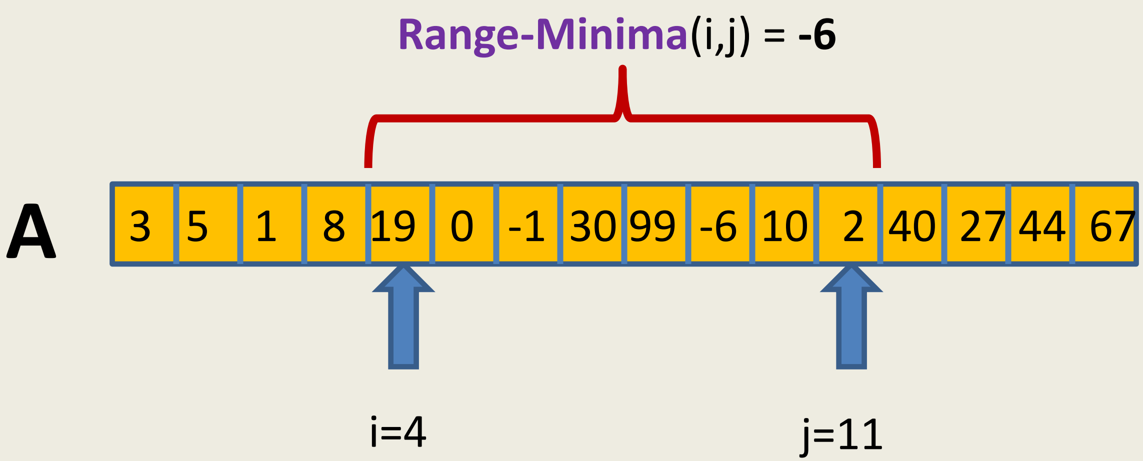
Given: an array **A** storing n numbers,

Aim: a data structure to answer a sequence of queries of the following type

Range-minima(i,j) : report the smallest element from **A**[i],…,**A**[j]

Let **A** store one **million** numbers

Let the number of queries be **10 millions**



Range-Minima Problem

Applications:

- **Computational geometry**
- **String matching**
- **As an efficient subroutine in a variety of algorithms**

(we shall discuss these problems sometime in this course or the next level course CS345)

Range-Minima Problem

Solution 1: Answer each query in a brute force manner using **A** itself.

```
Range-minima-trivial(i,j)
{    temp ← i+1;
     min ← A[i];
     While(temp <= j)
     {   if (min > A[temp])
             min ← A[temp];
         temp ← temp+1;
     }
     return min
}
```

Time for answering **all** queries:
a few hours

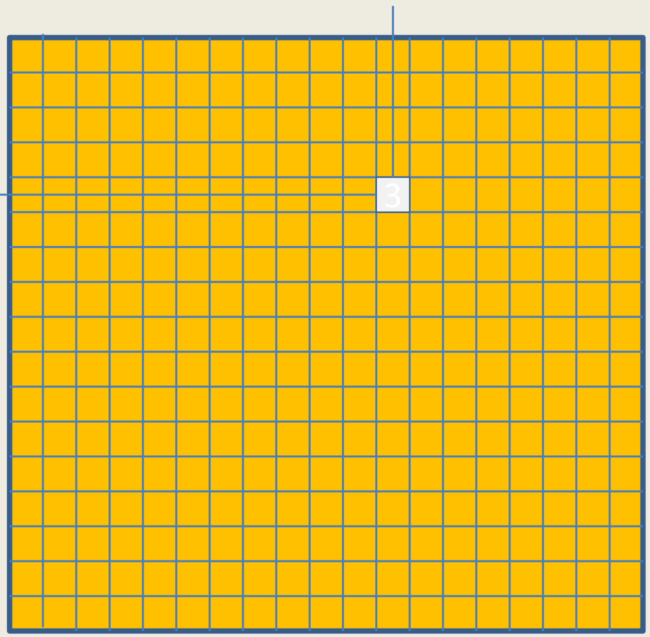
Time taken to answer a query: few **milliseconds**

Range-Minima Problem

Solution 2: Compute and store answer for each possible query in a $n \times n$ matrix **B**.

j

i



B

B[i][j] stores the smallest element from **A**[i],…,**A**[j]

Space : O(n^2)

Solution 2 is Theoretically efficient but practically impossible

Size of **B** is **too large** to be kept in RAM. So we shall have to ke t of it in the **Hard disk** Hence it will take a few **milliseconds per query**.

Range-Minima Problem

Question: Does there exist a data structure for Range-minima which is

- **Compact**

  (nearly **the same size** as the input array A)
- **Can answer each query efficiently ?**

  (a few **nanoseconds** per query)

Homework 3: Ponder over the above question.

(we shall solve it soon)

Data structures to be covered in this course

Elementary Data Structures

- Array
- List
- Stack
- Queue

Hierarchical Data Structures

- Binary Heap
- Binary Search Trees

Augmented Data Structures

Most fascinating and powerful data structures