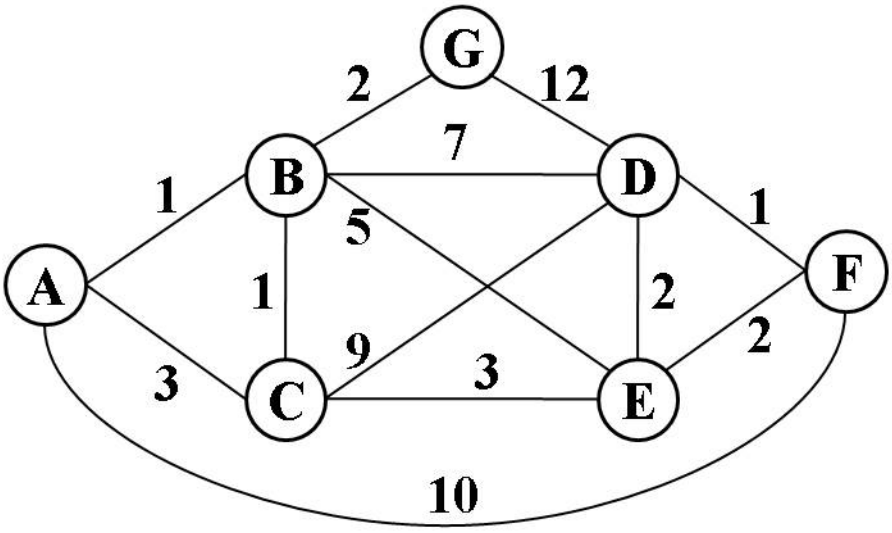
CSE373 Fall 2013

Some Additional Example Questions for the Final Exam

Name:_____

1. Consider the following undirected, weighted graph:



Step through Dijkstra's algorithm to calculate the single-source shortest paths from A to every other vertex. Show your steps in the table below. Cross out old values and write in new ones, from left to right within each cell, as the algorithm proceeds. Also list the vertices in the order which you marked them known. Finally, indicate the lowest-cast path from node A to node F.

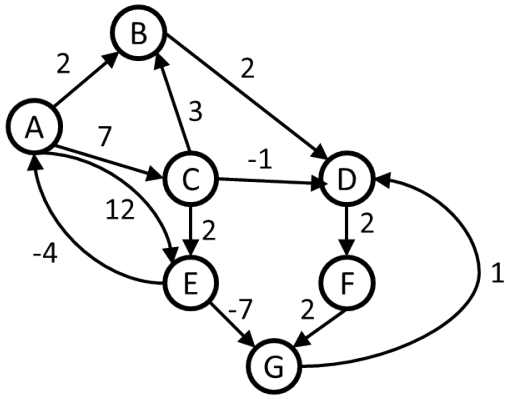
Solution:
Known vertices (in order marked known): A B C G E D or F D or F

Vertex	Known	Cost	Path
A	Y	0	
B	Y	1	A
C	Y	3 2	A B
D	Y	8 7	B E
E	Y	6 5	B C
F	Y	10 7	A E
G	Y	3	B

Lowest-cost path from A to F: A to B to C to E to F

Name:_____

2. Consider the following directed, weighted graph:



(a) Even though the graph has negative weight edges, step through Dijkstra's algorithm to calculate *supposedly* shortest paths from A to every other vertex. Show your steps in the table below. Cross out old values and write in new ones, from left to right within each cell, as the algorithm proceeds. Also list the vertices in the order which you marked them known.

Solution:

Known vertices (in order marked known): A B D F C G E

Vertex	Known	Distance	Path
A	Y	0	
B	Y	2	A
C	Y	7	A
D	Y	4	B
E	Y	12 9	A C
F	Y	6	D
G	Y	8	F

(b) Dijkstra's algorithm found the wrong path to some of the vertices. For just the vertices where the wrong path was computed, indicate *both* the path that was computed and the correct path.

(c) What *single* edge could be removed from the graph such that Dijkstra's algorithm would happen to compute correct answers for all vertices in the remaining graph?

Solution:

(b) Computed path to G is A,B,D,F,G but shortest path is A,C,E,G.
Computed path to D is A,B,D but shortest path is A,C,E,G,D.
Computed path to F is A,B,D,F but shortest path is A,C,E,G,D,F.

(c) The edge from E to G.

Name:_____

3. Suppose we define a different kind of graph where we have weights on the vertices and not the edges. Does the *shortest-paths problem* make sense for this kind of graph? If so, give a precise and formal description of the *problem*. If not, explain why not. Note we are not asking for an algorithm, just what the problem is or that it makes no sense.

   Solution:
   Yes, this problem makes sense: Given a starting vertex v find the lowest-cost path from v to every other vertex. The cost of a path is the sum of the weights of the vertices on the path.

Name:_____

4. Consider using a simple linked list as a dictionary. Assume the client will never provide duplicate elements, so we can just insert elements at the beginning of the list. Now assume the peculiar situation that the client may perform any number of insert operations but will only ever perform at most one lookup operation.

   (a) What is the worst-case running-time of the operations performed on this data structure under the assumptions above? Briefly justify your answer.

   (b) What is the worst-case amortized running-time of the operations performed on this data structure under the assumptions above? Briefly justify your answer.

   Solution:

   (a) inserts are $O(1)$ (push on the front of a linked list), but the lookup is $O(n)$ where n is the number of inserted items since the lookup may be last and be for one of the earliest inserted items

   (b) amortized all operations are now $O(1)$. Inserts are still $O(1)$. And the lookup can take at most time $O(n)$ where n is the number of previously inserted items. So the total cost of any n operations is at most $O(n)$, which is amortized $O(1)$.

Name:_____

5. In class we studied algorithms DFS for depth-first traversal of a graph and BFS for breadth-first traversal of a graph. Also recall $\Theta(f(n))$ just means $O(f(n))$ and $\Omega(f(n))$.

   Although DFS and BFS are for arbitrary graphs, we can use them for traversals starting from the root of a large, rooted binary tree with n nodes.

   (a) Use a picture to describe such a binary tree for which DFS from the root needs $\Theta(n)$ extra *space*.

   (b) Use a picture to describe such a binary tree for which BFS from the root needs $\Theta(n)$ extra *space*.

   (c) Use a picture to describe such a binary tree for which DFS from the root needs $\Theta(1)$ extra *space*.

   (d) Use a picture to describe such a binary tree for which BFS from the root needs $\Theta(1)$ extra *space*.

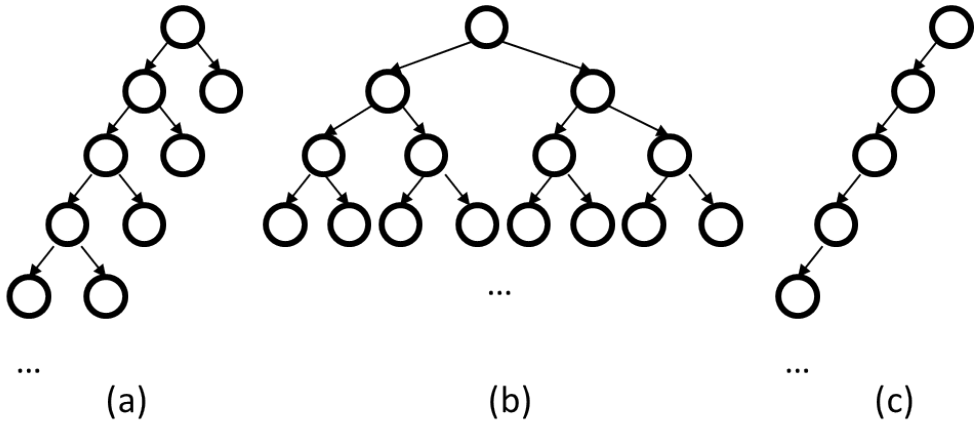
   Solution:
   There are various possible answers for each, but it is important the tree not have too many nodes for the bounds to hold.

   (a) See (a) below. We need a tall tree where we push two nodes as we descend each level of the tree. The tree must be unbalanced — a balanced tree would need only a stack of logarithmic size.

   (b) See (b) below. We need a balanced tree. For a complete tree, for example, when the traversal gets to the last full level of the tree, all the nodes in this level are in the tree and there are $O(n)$ such nodes (between roughly 1/4 and 1/2 of all the nodes).

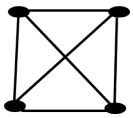
   (c) See (c) below: A binary tree does not require two children at internal nodes, so the simplest answer is a tree with one leaf despite many internal nodes.

   (d) The answer to (a) and the answer to (c) work fine, as done any graph where the number of nodes at any height in the tree is a constant.

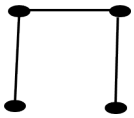


Name:_____

6. Here is a picture of an undirected graph with 4 nodes that contains every non-self edge:

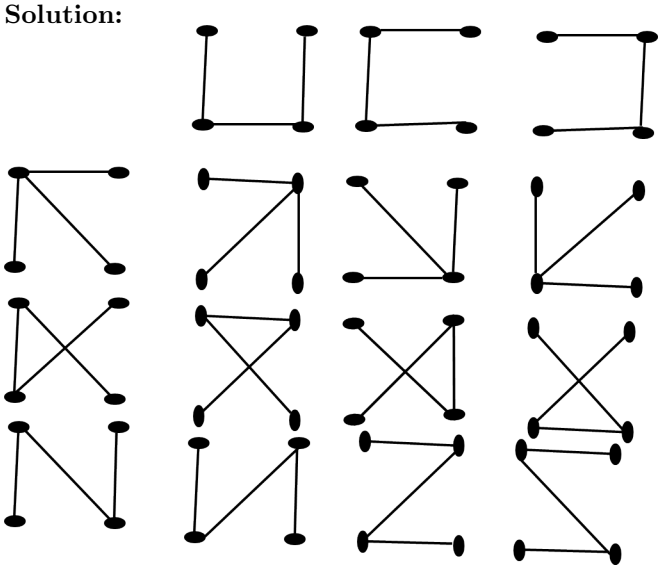


Here is a picture of just one possible *spanning tree* for this graph:



Draw fifteen more spanning trees for this same graph.

Solution:



Name:_____

7. Given this code, complete the client code below such that the call to `length` goes into an infinite loop. Then explain in 1-2 English sentences how to rewrite one of the methods in `Bag` to avoid this problem.

```
public class Node {
  public int x;
  public Node next;
}
public class Bag {
   private Node elements = null;
   void add(int y) {
      Node n = new Node();
      n.x = y;
      n.next = elements;
      elements = n;
   }
   int length() {
      Node p = elements;
      int i = 0;
      while(p != null) {
        p = p.next;
        i++;
      }
      return i;
   }
   Node allElements() {
      return elements;
   }
}

// client code:
Bag b = new Bag();
// YOUR "EVIL" CODE HERE
b.length();
```

Solution:

```
b.add(42); // number irrelevant
Node n = b.allElements();
n.next = n; // cycle!
```

The method `allElements` should "copy-out" by returning a copy of the list held in `elements`.

Name:_____

8. For each of the following situations, name the best sorting algorithm we studied. (For one or two questions, there may be more than one answer deserving full credit, but you only need to give one answer for each.)

   (a) The array is mostly sorted already (a few elements are in the wrong place).

   (b) You need an $O(n \log n)$ sort even in the worst case *and* you cannot use any extra space except for a few local variables.

   (c) The data to be sorted is too big to fit in memory, so most of it is on disk.

   (d) You have many data sets to sort *separately*, and each one has only around 10 elements.

   (e) You have a large data set, but all the data has only one of about 10 values for sorting purposes (e.g., the data is records of elementary-school students and the sort is by age in years).

   (f) Instead of sorting the entire data set, you only need the k smallest elements where k is an input to the algorithm but is likely to be much smaller than the size of the entire data set.

   Solution:

   (a) insertion sort

   (b) heap sort

   (c) merge sort

   (d) insertion sort (or selection sort is probably okay too)

   (e) bucket sort

   (f) selection sort is the simplest most natural choice and fine if k is truly small – this is the expected answer. If k is not a small constant, we might prefer heap sort or a variant of quicksort with a cut-off like we used on a homework problem. So full credit for any of these answers.

Name:_____

9. Suppose we sort an array of numbers, but it turns out every element of the array is the same, e.g., `{17, 17, 17, ..., 17}`. (So, in hindsight, the sorting is useless.)

   (a) What is the asymptotic running time of insertion sort in this case?

   (b) What is the asymptotic running time of selection sort in this case?

   (c) What is the asymptotic running time of merge sort in this case?

   (d) What is the asymptotic running time of quick sort in this case?

   Solution:

   (a) $O(n)$

   (b) $O(n^2)$

   (c) $O(n \log n)$

   (d) $O(n^2)$

Name:_____

10. (a) What is the asymptotic running time for summing an array in parallel using divide-and-conquer assuming an infinite number of processors?

    (b) Suppose we changed mergesort to do the two recursive sorts in parallel.

    - What is the recurrence relation that now describes the asymptotic run-time of mergesort over an n-element array range?
    - What is the solution to this recurrence (i.e., what is the worst-case running time of this algorithm)?
    - In 1–2 English sentences, describe the meaning (what the method does) of calling `t.join()` where `t` is an instance of a subclass of `java.lang.Thread`.
    - In 1–2 English sentences, describe the purpose (why you would do it) of calling `t.join()` where `t` is an instance of a subclass of `java.lang.Thread`.

Solution:

(a) $O(\log n)$

(b) $T(n) = 1T(n/2) + O(n)$ and $O(n)$

(c) The method does not return until and unless the thread created by calling `start` on the object referred to by `t` has finished executing.

(d) By waiting until the thread finishes executing any subsequent accesses of memory written to by the thread will see the full result of the thread's execution.

Name:_____

11. You are at a summer internship working on a program that currently takes 10 minutes to run on a 4-processor machine. Half the execution time (5 minutes) is spent running sequential code on one processor and the other half is spent running parallel code on all 4 processors. Assume the parallel code enjoys perfectly linear speedup for any number of processors.

    Note/hint/warning: This does *not* mean half the work is sequential. Half the *running time* is spent on sequential code.

    Your manager has a budget of $6,000 to speed up the program. She figures that is enough money to do only one of the following:

    - Buy a 16-processor machine.
    - Hire a CSE373 graduate to parallelize more of the program under the highly dubious assumptions that:
      - Doing so introduces no additional overhead
      - Afterwards, there will be 1 minute of sequential work to do and the rest will enjoy perfect linear speedup.

    Which approach will produce a faster program? Show your calculations, including the total *work* done by the program and the expected running time for both approaches.

    Solution:
    This is an application of Amdahl's Law. Let S be the running time for the sequential portion, L be the running time for the parallel portion on 1 processor and P be the number of processors. Then $T_p = S + L/P$. Initially T_P is 10 minutes, S is 5 minutes and $P = 4$, which means L is 20 minutes. So the total work is 20+5=25 minutes.

    Therefore, $T_{16} = 5 + 20/16$ is 6.25 minutes and that's the "buy a 16-processor" option.

    Under the "hire an intern" option S becomes 1 and L becomes 24 minutes. So the total time is $1+24/4$ which is 7 minutes.

    So the better choice is to buy the computer — apologies to those of you hoping for the job.