TF Questions

A directed graph has a topological ordering if and only if it contains no cycle.
True

Both BFS and DFS can be used to find shortest path from one node to another node on graphs that are unweighted.
False

If a directed graph has a topological ordering, then this topological ordering is unique.
False

Kruskal's, Prim's and Reverse Delete algorithms are all examples of greedy algorithms
True

In an unweighted strongly connected (directed) graph, the shortest distance from A to B is always the same as that from B to A.
False

In a weighted undirected graph, the shortest distance from A to B is always the same as that from B to A.
True

In an unweighted directed graph, the shortest distance from A to B is always the same as that from B to A.
False

In the stable matching problem with n men and n women, if a man and a woman are each other's last preference, then they will never be matched with each other in any stable matching.
False

We can find the k-th largest element in a binary max heap in $\Omega(1)$ time.
True

A strongly connected (directed) graph cannot be a DAG
True

If the heaviest weight edge e in an undirected connected graph G is unique, then e cannot belong to any minimum spanning tree of G.
False

The height of a complete binary tree with n nodes is O(n)
True

Given a graph G, if there is no negative cost cycles in G, then Dijkstra's algorithm will work correctly on G
False

MC questions: (correct answers are in red)

Which one of the following statements is False about binary heaps.

a) We should use a min-heap instead of a max-heap in Dijkstra's algorithm to find single source shortest path.
b) Inserting a new element into a heap has a worst-cast runtime of O(log n)
c) Checking the max element in a max-heap has a worst-cast runtime of O(1)
d) Finding an arbitrary element in a heap has a worst-cast runtime of O(log n)
e) Popping the maximum element from a max-heap has a worst-cast runtime of O(log n)

Running time of insertion sort on every input is:

a) THETA(n^2)
b) OMEGA(n^2)
c) O(n^2)
d) None of the above

Which of the following are true about the Gale-Shapley algorithm (with men proposing)?

a) Men end up with their worst valid partners
b) Women end up with their worst valid partners
c) Men end up with their best valid partners
d) Women end up with their best valid partners

Select the pair of runtimes that have the same Big-Theta
(a) log n + n log n
(b) 2^n
(c) n + log(n ^ n)
(d) log (2 ^ n)

Select all runtimes that are Ω(n) and O(n^3)
(a) n log n
(b) n^2
(c) n^4
(d) log n

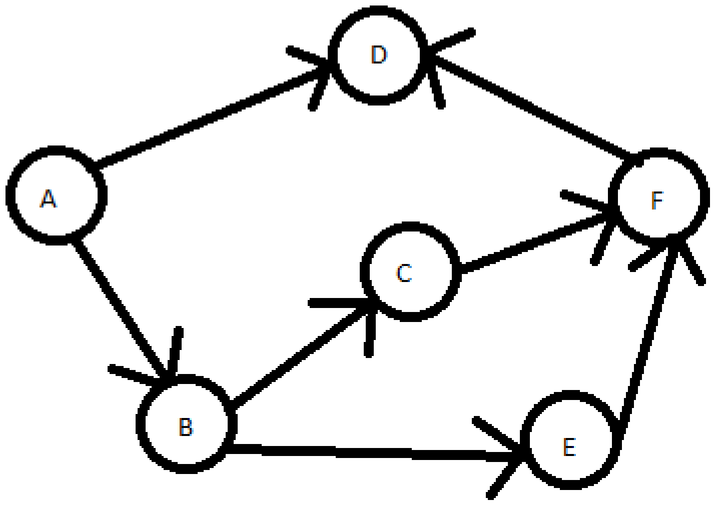
Which of the following bounds on the worst-case runtime complexity of Dijkstra's algorithm (using binary min-heap) is correct?
(A) O(V)
(B) O(E)
(C) O(VlogV)
(D) O(ElogE)
(E) None of the above

Suppose [2, 5, 7, 8, 10, 9, 11] is a binary min-heap. What will the heap look like after running the Extract-Min operation?

(a) [5, 7, 8, 10, 9, 11]
(b) [5, 8, 7, 11, 10, 9]
(c) [5, 7, 8, 9, 10, 11]

(d) [5, 8, 7, 9, 10, 11]

Problem 1



a. Is it possible to get a Topological Sorted order of the graph shown above?
b. If you answered No, justify why it is not possible for the graph.
   If you answered Yes, give all possible topological sorted orders of the graph.

Ans:

a. Yes, it is possible. (2 points)
b. A, B, E, C, F, D and A, B, C, E, F, D (4 points each)

Problem 2A

Suppose you are given an undirected weighted graph with a source node s and a destination node d. Given a particular edge e of the graph, give a polynomial time algorithm to check whether there exists a shortest path from s to d going through e.

Solution: Suppose e=(u, v). Run Dijkstra's from u, and then from v, and another from s as source node. Let dist(a, b) denote cost of shortest path between a and b. Then, answer is yes iff dist(s, d) == dist(u, s) + cost of e + dist(v, d) OR dist(s, d) == dist(v, s) + cost of e + dist(u, d).

10 points for the right answer, -2 points if some details are missing

5 points if performing Dijkstra's algorithm, but the output is not always correct

OR if the output is correct but the algorithm does not run in polynomial time

OR if the algorithm would be correct by replacing BFS (which doesn't work for weighted graph) with Dijkstra's algorithm

Up to 2 points partial credit for any other incorrect attempt

Problem 2B

The diameter of a graph is defined as the maximum of the length of shortest paths between any pair of vertices. Design an algorithm to find the diameter of a connected undirected graph in O(mn).

Solution:

Perform BFS starting from every vertex v of the graph and record the maximum shortest path length from v, and then return the maximum value of all vertices. The total time takes O(n(m+n)) = O(nm) since the graph is connected (so m+n = O(m)).

10 points for the right answer, -2 points if some details are missing

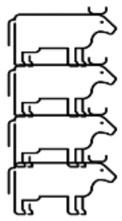
5 points if performing BFS, but the output is not always correct

OR if the output is correct but the algorithm does not run in O(mn)

2 points partial credit for any valid attempt

Problem 3

Farmer John has N cows (1, 2,..., N) who are planning to escape to join the circus. His cows generally lack creativity. The only performance they came up with is the “cow tower”. A “cow tower” is a type of stunt in which every cow (except for the bottom one) stands on another cow’s back and supports all cows above in a column. The cows are trying to find their position in the tower. Cow I (i = 1, 2, … , N) has weight Wi and strength Si. The “risk value” of cow i failing (Ri) is equal to the total weight of all cows on its back minus Si.



We want to design an algorithm to help cows find their positions in the tower such that we minimize the maximum “risk value” of all cows. For each of the following greedy algorithms either prove that the algorithm correctly solves this problem or provide a counter example. Hint: One of the two solutions is correct and the other is not.

a) Sort cows in ascending order of Si from top to bottom
   Counter example

   5 points for valid counterexample

   Partial credit up to 2 points for attempted proof or invalid counter example

b) Sort cows in ascending order of Si+Wi from top to bottom

The proof is similar to the proof we did for the scheduling problem to minimize maximum lateness. We first define an inversion as a cow i with higher (Wi+Si) being higher in the tower compared to cow j with lower (Wj+Sj). We can then show that inversions can be removed without increasing the maximum risk value time. We then show that given an optimal solution with inversions, we can remove inversions one by one without affecting the optimality of the solution until the solution turns into our solution.

1. Inversions can be removed without increasing the risk value.

   Remember that if there is an inversion between two items a and b, we can always find two adjacent items somewhere between a and b so that they have an inversion between them. Now we focus on two adjacent cows (one standing on top of the other) who have an inversion between them, e.g. cow i with higher (Wi+Si) is on top of cow j with lower (Wj+Sj). Now we show that scheduling cow i before cow j is not going to increase the maximum risk value of the two cows i and j. We do this one cow at a time:

   - By moving cow j higher we cannot increase the risk value of cow j
   - By moving cow i lower (below cow j) we will increase the risk value of cow j but since cow i has a higher total weight and strength than cow j, the risk value of cow i (after removing the inversion) will not be worse than the risk value for cow j prior to removing the inversion:
     - Risk value of cow j before removing the inversion: Wi + (weight of remaining cows above cows i and j) - Sj

- Risk value of cow i after removing the inversion: Wj + (weight of remaining cows above cows i and j) - Si
- Since Wi+Si >= Wj+Sj , if we move Si and Sj to the opposite sides we get: Wi-Sj >= Wj-Si. (weight of the remaining cows above the two cows i and j does not change). In other words, the risk value of cow j before removing the inversion is higher than that of cow i after removing the inversion

2. Since we know that removing inversions will not affect the maximum risk value negatively, if we are given an optimal solution that has any inversions in it, we can remove these inversions one by one without affecting the optimality of the solution. When there are no more inversions, this solution will be the same as ours, i.e. cows sorted from top to bottom in ascending order of weight+strength. So our solution is also optimal.

10 points for correct proof by inversion argument or otherwise

Partial credit up to 7 points for proof that omits section 2 or is otherwise incomplete.

Partial credit up to 5 points for proof without a complete mathematical argument in section 1

Partial credit up to 2 points for incorrect proof

0 points for counter example

Problem 4

You are hosting an online auction that's open to millions of people. After your team presents a new item, you will wait for 30 minutes for people to privately send in their bid prices. In other words, during this 30 minutes, you will receive a data stream of integers like: 3100, 200, 4030, 12000, 399... To get a better sense of how people are feeling about this item, the eager analyst on your team wants you to design an efficient algorithm that can output the median bid price you have seen so far at any given time during this 30 minutes. The complexity of one query for current median has to be O(1). And if you are maintaining some internal data structures, the runtime of maintaining that data structure cannot be greater than O(log n) per new bid price. Hint: See if this can be done with two heap data structures.

As the hint suggests, we will maintain two heaps to solve this problem — a min-heap and a max-heap, each stores about half of the bid prices we've seen so far. For each query of median, we simply examine the top elements of two heaps to determine the current median.

Here is the pseudocode:

```
MaxHeap<int> mxhp;
MinHeap<int> mnhp;
void add_new_bid_price(int price){
        if(!mnhp.empty() && price < mnhp.front()){
                mxhp.add(price);
                if(mxhp.size() > mnhp.size() + 1){
                        mnhp.add(mxhp.front()); //.front() functions peeks the top element of the heap
                        mxhp.pop(); //.pop function deletes the heap top element
                }
        }else{

                mnhp.add(price);
                if(mnhp.size() > mxhp.size() + 1){
                        mxhp.add(mnhp.front());'
                        mnhp.pop();
                }
        }
}

double current_median(){
        if(mnhp.size() < mxhp.size() ){
                return mxhp.front();
        }else if(mnhp.size() < mxhp.size()){
                return mnhp.front();
        }else{
                return (mnhp.front() + mxhp.front())/2.0;
        }
}
```

Basically, if we visualize the bid prices we've seen so far at a certain time stamp as a sorted list, the min heap stores larger half of the list while the max heap store the smaller half of the list. And their top elements are the parts in the middle.

For example, if the prices we've seen so far are: 40 10 50 20 70 5 (if sorted: 5 10 20 40 50 70) by the logic of **add_new_bid_price** function

min-heap would be storing: 40 50 70

max-heap would be storing: 20 10 5

If you look at the current_median() function, we would return (mnhp.front() + mxhp.front())/2.0 = (20 + 40)/2.0 = 30, which is the correct median. Of course if the total number of elements we've seen so far is odd, there would be a heap that stores one more element than the other, and the top element of the larger heap would be the correct

(Description of the data structure/main idea: 5 points)

(Description of insertion/keeping balance: 4 points)

(Description of query: 1 point)

median. Since we only peak the top elements of two heaps, the runtime of current_median() is O(1). And when a new bid price comes in, we do at most (2 heap_push + 1 heap_pop) operation, the runtime of add_new_bid_price()is O(logn)