1) 20 pts

Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE]
If f(n) = O(g(n)), then g(n) = Ω(f(n)).

[TRUE]
In the interval scheduling problem, if all intervals are of equal size, a greedy algorithm based on earliest start time will always select the maximum number of compatible intervals.

[FALSE]
There are at most 2 distinct solutions to the stable matching problem: one that is preferred by men and one that is preferred by women.

[FALSE]
The divide and conquer algorithm to solve the closest pair of points in 2D runs in O(n log n). But if the two lists of points, one sorted by X-coordinate and the other sorted by Y-coordinate are given to us as input, the rest of the algorithm (skipping the sorting steps) runs in O(n) time.

[TRUE].
In a graph, if one raises the lengths of all edges to the power 3, the minimum spanning tree will stay the same.

[TRUE]
The first edge added by Kruskal's algorithm can be the last edge added by Prim's algorithm.

[TRUE]
The shortest path of a graph could change if the weight of each edge is increased by an identical number.

[TRUE]
The shortest path in a weighted DAG can be found in linear time.

[FALSE]
If an operation takes O(1) amortized time, then that operation takes O(1) worst case time.

[FALSE]
In Binomial heaps, the decrease-key operation takes O(1) worst case time.

2) 15 pts

You are given a set S of n points, labeled 1 to n, on a line. You are also given a set of k intervals $I_1, \ldots, I_k$, where each interval I_i, is of the form $[s_i, e_i]$, $1 \leq s_i \leq e_i \leq m$. Present an efficient algorithm to find a smallest subset XÍ S of points such that each interval contains at least one point from X. Prove that your solution is optimal.

Solution: Sort the intervals in increasing order of ei. Select the first point as the right end-point of the first interval in this order. Remove all intervals which intersect with this point, and repeat. Proof of correctness is similar to the interval scheduling problem discussed in class. If the algorithm picks points p1 < p2 < · · · < pk, and optimum solution picks points q1 < q2 < · · · , qs, then prove by induction that pi ≥ qi, and so k ≤ s.

Grading:

1. Find the correct solution (earliest end time): 9 pt
2. Prove the correctness: 6 pt
3. If the solution is wrong (not using earliest end time):
    a. Student uses a reasonable heuristic, and show his reasoning in the proof: 4 – 5 pt
    b. Otherwise: 0 – 3 pt
4. Common mistakes:
    a. The most common misunderstanding of the problem is that students assume the locations of points are given in advance. In this case: -2 pt;
    b. If the student didn't remove intervals after adding a new point: -3 pt

Many students failed to use the proper heuristic (earliest end time). If this is the case, then the algorithm as well as the following proof would be wrong. So they will get 0 – 5 pt.

3) 10 pts

Arrange the following functions in increasing order of growth rate with g(n) following f(n) in your list if and only if $f(n) = O(g(n))$

$$\log n^n,\ n^2,\ n^{\log n},\ n \log n \log \log n,\ 2^{\log n},\ n^{\sqrt{2}}$$

Solution: $2^{\log n}$, $\log n^n$, $n \log n \log \log n$, $n^{\sqrt{2}}$, n^2, $n^{\log n}$

Grading:

There are 10 points and 6 functions to compare. With 6 functions you can have a total of 15 pairwise comparisons. Count the number of incorrect pairwise comparisons (x) and deduct half a point for every incorrect comparison. For example, if there were only three functions: n, n^2 and n^3 then the order n, n^3, n^2 only has one incorrect pairwise comparison but n^3, n, n^2 has two incorrect pairwise comparisons. You get the point!

4) 10 pts
An ***ordered stack*** is a singly linked list data structure that stores a sequence of items in increasing order. The head of the list always contains the smallest item in the list. The ordered stack supports the following two operations. POP() deletes and returns the head (NULL if there are no elements in the list). PUSH(x) removes all items smaller than x from the beginning of the list, adds x and then adds back all previously removed items. PUSH and POP can only access items in the list starting with the head. What would be the amortized cost of PUSH operation, if we start with an empty list?? Use the aggregate method.

Solution: The worst sequence of operations is pushing in order. Assume we push 1,2,3, …, n, starting with an empty list. The first push costs 1. The second push costs 3 (pop, push(2), push(1)). The last push costs 2n-1 (n-1 pops followed by push(n), followed by n-1 pushes). The total cost is given by

1 + 3 + 5 + … +(2n-1) = O(n2)
The amortized cost is O(n).

Grading:
Partial credit (which adds up)

1. Right derivation of total cost of worst case scenario O(n2) – 5 points
2. Right approach for worst case for nth element (2n-1 or 2n+1, O(n)+O(1)+O(n), 3n-2 for having compare as step) – 3 points
3. Have right concept of amortized cost by aggregate method (# of total operation in worst case/#of operation, not necessarily should explicitly write the definition) – 2 points

Partial credit (final score)

1. Right case of answer, O(n) with – 8 points
   - wrong approach (wrong worst case scenario)
   - use average case, not average of worst case
   - wrong math
   - wrong statement of amortized cost (e.g. using average cases instead of worst cases)
2. Similar answers (when the solution is correct)
   - didn't finalize math – 8 points
   - theta(n), n – 8 points
   - O(n+1) for correct solution – points 10 (used total n+1 cases for amortized cost)

3. Similar answer with wrong approach - 6 points (e.g. O(n/2), n, linear, or wrong math)

5) 12 pts

a) 5 pts

You are given a weighted graph G, two designated vertices s and t, and a number W. Your goal is to find a path from s to t in which every edge weight is at least W. Describe an efficient algorithm to solve this problem and show its complexity. Your algorithm should work even if the edge weights are negative.

Solution: Run BFS, ignoring any edges of weight less than W. This will take O(V + E) time. Or Remove all edges with weight less than W and run BFS from s.

Common Mistakes:

1. Running Kruskal's, Prim's or Dijkstra's algorithms while ignoring edges of weight less than W. This is correct but not efficient (3.5 points)

2. Giving the correct algorithm but not analyzing the complexity correct (4.5 points)

3. Giving the correct algorithm but sorting the edges first. (The sort is unnecessary and increases the complexity) (4.5 points)

4. Choosing one edge going out of s with weight higher than W to get to a new node, v. Then choosing a new edge going out of v with weight higher than W and so on until you reach t. If you don't mention at some point you might need to backtrack to a previous node and continue the search the algorithm is incorrect (2.5 points)

5. If your algorithm finds a path where the cost of the entire path is less than W you're solving the wrong problem but since there is some similarity you get some points (2.5 points)

b) 7 pts

You are given a weighted graph G, two designated vertices s and t. Your goal is to find a path from s to t in which the minimum edge weight is maximized i.e. if there are two paths with weights 10->1->10 and 2->2->2 then the second path is considered better since the minimum weight (2) is greater than the minimum weight of the first (1). Describe an efficient algorithm to solve this problem and show its complexity. You may use the algorithm from Part (a). For full credit, your algorithm must run in
$O((V + E) \log E)$.

Solution: Sort all edges $e_1, e_2, \ldots, e_m$. Then run the Part a) algorithm with $W = e_k$, testing for each W whether you can find a path in the graph using only edges of weight greater than W. The largest W is the solution. How to choose W among $e_1, e_2, \ldots, e_m$? We do it in a binary search fashion. Runtime $O((V + E) \log E)$.

You can also run Kruskal's or Prim's algorithm and instead of going through the edges from smallest weight to the largest weight, you go the other way around and check for cycles. At the end run DFS or BFS to find the path between s and t and find the edge with minimum weight on this path. That will be the answer

Common Mistakes:

1. Don't use binary search and go over different edge weights one by one (5 points)
2. If your algorithm finds "all paths from s to t" and you don't mention how you find those paths this is not correct. BFS, DFS and Dijkstra find only one path. (3 points)
3. Finding the most expensive path is not correct. (2 points)
4. Finding the most expensive edge going out of s to a new node 'v', and then finding the most expensive edge going out of 'v' until you find 't' is not correct. (2 points)
5. In some cases on of the two correct algorithms were explained but it wasn't explained how you find the final answer. Either 1 or 2 points were deducted.

6) 10 pts

Given an instance of the original Stable Marriage problem with n couples (so that every man ranks every woman and vice versa), where there is a man M who is last on each woman's list and there is a woman W who is last on every man's list. Prove or disprove the following statement: if we run the Gale-Shapley algorithm on this instance, then M and W will be paired with each other.

Solution: True. Note that no man will propose to W unless he has been rejected by all n-1 other women. Let X be the first man to propose to W. At the time this happens, all the other women must be engaged to men they prefer to X. This can only happen if X is M, as otherwise M would have been preferred to X by some woman. So M proposes to W, she accepts and then everyone is engaged and the algorithm stops

Grading rubric: Points awarded for some of the common mistakes are:
1) 4 points, if the proof argument is discussed for an example but not for general case.
2) 7 points, a women (other than W) may not get a better proposal other than M and she may not reject M. Why should this case not happen? Need to argue about this.
3) 6 points, assuming that Galey-shapely running with n-1 men and women and then adding m and w, give the same result as Galey-shapely running with n men and women.

7) 13 pts

Suppose we define a new kind of directed graph where we have (positive) weights on the vertices and not the edges. If the length of a path is defined by the total weight of all nodes on the path, describe an algorithm that finds the shortest path between two given points within this graph.

Solution 1. Remove the vertex weight by slicing every vertex a into two vertices a1 and a2 with an edge from a1 to a2 with the given weight for original vertex. All edges going into a will go into a1 and all edges going out of a will go out of a2. Then, to find shortest path from x to y in the original graph, we can run Dijkstra's to find the shortest path from x1 to y2. (Suppose vertex x is sliced into x1, x2, and y is sliced into y1 and y2)

Solution 2. The graph is directed. For each vertex, you can assign the weight of the vertex to its outgoing (or incoming) edges. Then run Dijkstra algorithm. Since the questions only asks for the shortest path (not the length of shortest path) this algorithm gives us a correct answer

Solution 3. For each edge like e = (u, v) assign the weight : w(v) + w(u)

Solution 4. Modify Dijkstra algorithm and the cost function.

- If you don't mention how to use weight of a node to update distance: - 5 point
- Sorting vertices by their weight and adding them to a heap based on their weight and them picking up the smallest one doesn't work. The distance from start point is important: (-7 point)

Wrong solutions that get 0 point:

- BFS, DFS and algorithms for finding MST doesn't work as they don't work for general shortest path problem.
- Convert graph G to G' with replacing nodes as edges and vise versa won't give you an efficient algorithm. Because the size of G' may not be linear.

8) 10 pts

Use the Master Theorem to solve each of the following recurrences by giving tight Θ-notation bounds, or describe why the Master Theorem does not apply to the particular recurrence formula(s).

(a) $T(n) = 2^n T(n/2) + n^n$

Solution:2^n is not constant, so master theorem does not apply

(b) $T(n) = 16T(n/4) + n!$ **Solution**: Θ(n!)

(c) $T(n) = 4T(n/2) + c\,n$ **Solution**: Θ(n^2)

(d) $T(n) = 2T(n/2) + n \log n$ **Solution**: $\Theta(n \log^2 n)$

(e) $T(n) = 64T(n/8) - n^2 \log n$ **Solution**: -n^2 log n is not asymptotically positive function, so master theorem does not apply

Additional Space

Additional Space