CS570 Fall 2018: Analysis of Algorithms Exam I

	Points
Problem 1	20
Problem 2	16
Problem 3	16
Problem 4	10
Problem 5	16
Problem 6	10
Problem 7	12
Total	100

Instructions:

1. This is a 2-hr exam. Closed book and notes

2. If a description to an algorithm or a proof is required please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.
5. Do not detach any sheets from the booklet. Detached sheets will not be scanned.
6. If using a pencil to write the answers, make sure you apply enough pressure so your answers are readable in the scanned copy of your exam.
7. Do not write your answers in cursive scripts.

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/FALSE]
Assume that no two men have the same highest-ranking woman. If the women carried out the proposal to men, then the Gale-Shapley algorithm will contain a matching set where every man gets their highest-ranking woman.

[TRUE/FALSE]
Consider a binary heap with n nodes stored as an array. The parent, left child and right child of the node with index 11 are at indices 6, 22, and 23 respectively.

[TRUE/FALSE]
Inserting an element into a binary min-heap takes O(1) time if the new element is greater than all the existing elements in the min heap.

[TRUE/FALSE]
Using the master's theorem the asymptotic bounds for the recurrence 2T(n/4) + n is $\Theta(n)$.

[TRUE/FALSE]
Consider a version of the interval scheduling problem where all intervals are of the same size. A greedy algorithm based on earliest start time will always select the maximum number of non-overlapping intervals.

[TRUE/FALSE]
For any tree with *n* vertices and *m* edges we can say that O(m+n) = O(m).

[TRUE/FALSE]
A directed graph G is strongly connected if and only if G with its edge directions reversed is strongly connected.

[TRUE/FALSE]
The number of binomial trees in a binomial heap with n elements is at most O(log n).

[TRUE/FALSE]
A minimum spanning tree of a bipartite graph is not necessarily a bipartite graph.

[TRUE/FALSE]
In Fibonacci heaps, the decrease-key operation has an amortized cost of (1).

2) 16 pts.
A Maximum Spanning Tree is a spanning tree with maximum total weight.

a) Present an efficient algorithm to find a Maximum Spanning Tree of an undirected graph G. (8 pts)

```
W.l.o.g., we assume the input graph is G = (V, E)
Generalized Kruskal's algorithm
F = ∅
Sort all edges by their lengths in nonincreasing order
for (each edge (u, v) ∈ E) do
    if (length(u, v) > 0 and (u, v) does not create a cycle in F) then
        Add (u, v) to F
    end if
end for
```

Time complexity of generalized Kruskal's algorithm is O(|E| log |V |) which is the same as normal Kruskal's algorithm.

b) Prove the correctness of your solution in part a. (8 pts)

In order to show F is a maximum spanning tree, we need to show two things: (1) F contains no cycle and (2) the total weight of F is maximum.

(1) is naturally true since when picking edges, we always avoid creating a cycle.

(2) For (2), we can apply the similar argument for minimal spanning tree. Clearly, edges with negative cost should be abandoned. Let F∗ be one maximum forest of G, if F and F∗ are the same, then we are done. Otherwise, there is an edge e ∈ F∗ that is not contained in F. Consider F + e, there must be a cycle C. Here we should observe that:

a. Every other edges in C has weight no less than e. It is true by the rule we construct F, edges with longer length are selected first.

b. There is some edge f in C that is not included in F∗.

Let F' = F + e − f, we have a new forest and have more common edges with F∗ than F. We can continue this process until we get F∗. Noting that, each time we replace f by e. we do not increase the weight, it is true by (a). Therefore, we have the following relation

$w(F) \geq w(F') \geq \ldots \geq w(F*)$. Since $F*$ is one maximum forest, then F must also be a maximum forest.

3) 16 pts.

In an instance of the Stable Matching problem, man m has claimed “I’m gonna marry w, anyway!” and by that, he means in every stable matching, m must marry w — or equivalently, if in a matching, m and w do not pair up, then the matching is definitely unstable. It is a strong claim and you want to figure out if it is true. For example, if w is the topmost choice of m and m is also the topmost choice of w, they must pair up in every stable matching and thus, the claim is true in this case. Design an O(n^2) algorithm that, given an instance of Stable Matching problem as well as man m and woman w, checks whether m marries w in every stable matching or not.

Solution.
We run Gale-Shapley algorithm twice: once with men being proposing and then, with women proposing. In the first execution of the algorithm we learn whom is the best woman that m can hope to marry in a stable matching.In the second execution, we see whom is the worst woman he may ever marry in a stable matching. If they are both equal to w, it means that m must marry w. Otherwise, there exists a stable matching (basically, one of the two matchings we have found) in which m marries someone other than w. The complexity of the Gale-Shapely algorithm is O(n^2) therefore running it twice with men proposing and women proposing will result in O(2*n^2) = O(n^2)

Rubric.

Providing a correct algorithm(+9)

The most straightforward solution:
Run the Gale-Shapely algorithm twice, with men proposing and women proposing. If both algorithms result in the matching of m & w, the claim is true - otherwise false (+9/+9)

Other answers:
Incomplete or inexplicit description of the above algorithm(+6/+9)
Looking for specific patterns in the preference lists to evaluate the claim(+3/+9)

Running the Gale-Shapely algorithm only once(+0/+9)
Explaining the Gale-Shapely algorithm(+0/+9)
Proving m and w will always be matched in all conditions(+0/+9)
Forcing m and w to be matched through the Gale-Shapely algorithm(+0/+9)
Any solution that indicates a misunderstanding of the Gale-Shapely algorithm(+0/+9)

**Note: If your provided algorithm is incorrect or incomplete, you will not receive points on the ~~explanation~~explanation of algorithm correctness or time complexity.*

Explaining why the proposed algorithm works(+4)

For the mentioned solution:
If m and w are matched in both pairings, it means they're each other's only valid partners and will exist in every possible matching; because:
If men propose they end up with their best valid partners, but women end up with their worst valid partner(+2)
If women propose they end up with their best valid partners, but men end up with their worst valid partners(+2)
Since m & w are each other's best and worst match and vice versa, they will always be paired together.

Other answers:
For noting the proposing side gets their best valid partners(not mentioning worst for the other side)(+2/+4)
For noting they're each other's only valid partners(+1/+4)
For noting that the matchings are distinct(but not saying best/worst valid partners with respect to the proposing side): (+1/+4)
Not providing any explanation of why the provided algorithm works(+0/+4)

Explaining the proposed algorithm's asymptotic complexity(+3)

For the known solution:
Since Gale-Shapely is O(n^2), the above algorithm will be of O(2*n^2) = O(n^2) (+3/+3)

Other answers:
Not providing any explanation of time complexity(+0/+3)

4) 10 pts

Rank the following functions in order from smallest asymptotic complexity to largest. No justification needed.

(lg is log base 2 by convention)

$lg\ n^{10}$, 3^n, $lg\ n^{2n}$, $3n^2$, $lg\ n^{lg\ n}$, $10^{lg\ n}$, $n^{lg\ n}$, $n\ lg\ n$

Solt.

$\lg n^{10} < \lg n^{\lg n} < \lg n^{2n} = n \lg n < 3n^2 < 10^{\lg n} < n^{\lg n} < 3^n$

Rubric: -2 points for every literal in the incorrect order

5) The Fractional Knapsack problem is described as follows. We have a resource (knapsack) with capacity W, i.e. it can hold items of total weight at most W. There are n items, each of weight w1, w2, . . . , wn. Each item also has a value v1, v2, . . . , vn. You are allowed to put all or any fraction of an item (say, 1/3 of an item) into the knapsack. The goal is to select a set of fractions p1, p2, . . . , pn for all items in order to maximize the total value p1v1 + p2v2 + . . . pnvn subject to the capacity constraint p1w1 + p2w2 + . . . pnwn $\leq$ W.

a) Present an efficient solution to solve the Fractional Knapsack problem

Solution: Choose items in decreasing order of vi/wi and choose as much of each item as possible without exceeding total capacity W. (6 pts)

Worng answers:

- Sort by vi (3 pts)
- Sort by wi (1 pt)
- Other solutions (0 pts)
- Mistake in greedy algorithm (-1 pts)

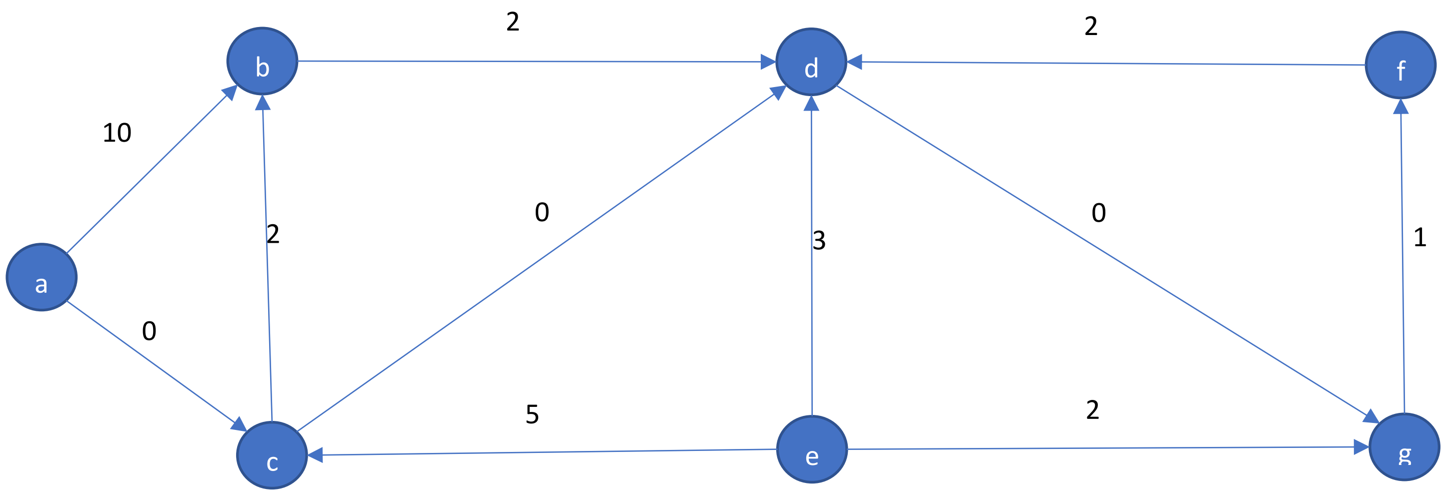
b) Prove that the solution provided in part a is correct.

Solution: Assume that a better solution O exists. Order the items in O in the same order as in our solution, and let's say the fraction of item k used in O is qk. Let the i^{th} item be the first item where pi is not equal to qi. So, the total capacity used by items 1 through i-1 is the same for both solutions, and the remaining capacity is also the same.
Since our solution uses as much of item i as possible, it must be that pi > qi. We can now improve O by increasing the fraction of item i chosen. We can achieve this by increasing this fraction to pi and removing an equivalent amount of weight from objects i+1 and higher (Since objects i+1 or higher have lower vi/wi ratios) (10 pts)

- Discussing the inversions between greedy and an assumed optimal solution
  - Proving the difference (10 pts)
  - Not proving (9 pts)
- Proving that greedy stays ahead:
  - Not mentioning inversions (7 pts)
  - Mentioning inversion and not proving the difference (9 pts)
- Proving by induction (4 pts) (it only proves that greedy stays ahead)
- Prove by contradiction, without mentioning greedy stays ahead or inversions (3 pts)
- Vaguely mentioning inversions (3 pts)
- Talking about inversion but not resolving (5 pts)
- Assuming constraints for the proof (-1 pt)

6) 10 pts

Below figure shows graph G along with its edge costs marked on each edge. Use Dijkstra's algorithm to find the shortest paths from node *a* to all nodes that can be reached from *a*. You should list all shortest paths in the order they are found.



Sol.
a->c
a->c->d
a->c->-d->g
a->c->d->g->f
a->c->b

If there is an incorrect shortest path to a node then -2
If the shortest paths are not in order, then -3

Other acceptable solution in a different format: -

(a,c)
(a,c)->(c,d)
(a,c)->(c,d)->(d,g)
(a,c)->(c,d)->(d,g)->(g,f)
(a,c)->(c,b)

12 pts

Given an array of n distinct integers sorted in ascending order, we are interested in finding out if there is a Fixed Point in the array. Fixed Point in an array is an index i such that arr[i] is equal to i. Note that integers in array can be negative.

Example:

Input: arr[] = {-10, -5, 0, 3, 7}

Output: 3 // since arr[3] == 3

a) Present an algorithm that returns a Fixed Point if there are any present in array, else returns -1. Your algorithm should run in O(log n) in the worst case. (6 pts)

Solt. First check whether middle element is Fixed Point or not. If it is, then return it; otherwise check whether index of middle element is greater than value at the index. If index is greater, then Fixed Point(s) lies on the right side of the middle point (obviously only if there is a Fixed Point). Else the Fixed Point(s) lies on left side.
If subproblem size equals one and Fixed is not found, return -1.

-3, if you prune the incorrect half of the array
-2 if you forget to return -1 when subproblem size = 1

b) Use the Master Method to verify that your solutions to part a) runs in O(log n) time. (3 pts)

a=1, b=2, f(n) = O(1).
n**(log a base b) = n**(log 1 base 2) = n**0 = O(1)
Case 2 $\rightarrow$ t(n) = θ (lg n)

-not graded if part (a) is incorrect.
-not graded if part (a) it runs in greater than O(log n) time.
-1 if the master theorem case is incorrect
-1 if n**(log a base b) is incorrectly computed

c) Let's say you have found a Fixed Point P. Provide an algorithm that determines whether P is a unique Fixed Point. Your algorithm should run in O(1) in the worst case. (3 pts)

Say a fixed point is found at index i. Check indices i+1 and i-1. If we don't find fixed points at these two indices, then there cannot be any other fixed points since the array is sorted and elements are all distinct, so for all other elements j above i, we must have arr[j] > j. And for all elements j below i, we must have arr[j] < j.

-1 if you omit to check either i+1 or i-1.