# CS570 Summer2019: Analysis of Algorithms Exam I

| | Points | | Points |
|---|---|---|---|
| Problem 1 | 20 | Problem 5 | 15 |
| Problem 2 | 9 | Problem 6 | 15 |
| Problem 3 | 20 | Problem 7 | 10 |
| Problem 4 | 11 | | |
| | **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 ]**
Suppose $f(n) = f\left(\frac{n}{2}\right) + 5log\ n$, then $f(n) = \theta\ (log\ n)$

**[ TRUE/FALSE ]**
The array [20 15 18 7 9 5 12 3 6 2] forms a binary max-heap.

**[ TRUE/FALSE ]**
A graph with an odd number of vertices cannot be bipartite.

**[ TRUE/FALSE ]**
Any stable matching is a perfect matching.

**[ TRUE/FALSE ]**
Binary search is $O(n)$.

**[ TRUE/FALSE ]**
The key value for the element at the last index of a binary min-heap will be the highest.

**[ TRUE/FALSE ]**
In the aggregate analysis different operations may have different amortized costs, while in the accounting method all operations have the same amortized cost.

**[ TRUE/FALSE ]**
A weighted graph has a unique MST only if its edge weights are distinct.

**[ TRUE/FALSE ]**
Suppose we have a weighted graph $G = (V, E, w)G = (V, E, w)$ and let S be a shortest $s - ts - t$ path for $s, t \in V$. If the weight of every edge in G is doubled (i.e., $w'(e) = 2w(e)w'(e) = 2w(e)$ for each $e \in Ee \in E$), then S will still be a shortest $s - t\ s - t$ path in $(V, E, w')(V, E, w')$.

**[ 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.

2) 9 pts

We have k >= 1, $\epsilon > 0$, c > 1 in the table below. Provide Yes/No answers in the table below (3 answers per row). No justification required.

| f(n) | $\log^k n$ | $n^k$ | $\sqrt{n}$ |
|---|---|---|---|
| g(n) | $n^\epsilon$ | $c^n$ | $n^{\text{sinn}}$ |
| Is $f(n)$ $O(g(n))$? | | | |
| Is $f(n)$ $\Omega(g(n))$? | | | |
| Is $f(n)$ $\Theta(g(n))$? | | | |

Solution:

| f(n) | $\log^k n$ | $n^k$ | $\sqrt{n}$ |
|---|---|---|---|
| g(n) | $n^\epsilon$ | $c^n$ | $n^{\text{sinn}}$ |
| Is $f(n)$ $O(g(n))$? | Yes | Yes | No |
| Is $f(n)$ $\Omega(g(n))$? | No | No | No |
| Is $f(n)$ $\Theta(g(n))$? | No | No | No |

3) 20 pts

Suppose that you are organizing an event. Given that you have a row of $n$ chairs and that there are two types of guests who will come to the event: Economists (E) and Politicians (P). You have learned from the previous event you organized that when two politicians sit next to each other, they tend to not get along and may disturb other guests. Your task is to assign one guest to each seat, but you can never seat two politicians together. For example, if there is a row of $n = 4$ chairs, some of the seating assignments you can do are EEEE, EPEP, PEPE, PEEP, etc., but you can never do PPEE, EPPE, etc. We are trying to find the number of valid seating assignments when there is a row of n chairs and we have p politicians and (n-p) economists to seat. Describe a dynamic programming solution to solve this problem.

a) Define (in plain English) subproblems to be explored. (5 pts)

Count(n,p) = number of possible seating arrangements when we have n chairs and p politicians (and n-p economists)

Rubrics: max number of possible assignments is also fine but not needed.
-2.5 if miss one of the variables in a way that it would not work.
-3 missing the number of the seats and guests in the subproblem definition.

b) Write the recurrence relation to come up with this count. (7 pts)

Count(n,p) = Count(n-2,p-1) + Count(n-1,p)

Rubrics: -3 if take max instead of sum or make it multiple case (will get grade if it works).
-2 if use Count(n-1, p-1) instead of Count(n-2, p-1)
-2 if use Count(n-1,p-2)
-3 if use count(n,p) = count(n-2,p-1) + count(n-1,p) +1 or similar
**Wrong subproblems**: -2 if use wrong subproblems like OPT(n)=OPT(n-1) + OPT(n-2) or similar but understood that it is a sum.

c) Using the recurrence formula in part b, write pseudocode (using iteration) to compute the count. (5 pts)
Make sure you have initial values properly assigned. (3 pts).
Initialize count(0,p) to 0 for p>0 (no seats)
Initialize count(0,0) to 1 (nothing)
Initialize count(1,1) to 1 (one seat and one politician)
Initialize count(1,p) to 0 for p >1 (one seat and more than one politician)

```
Initialize count(n,0) to 1 for all n (no politicians)

For i=2 to n
        For j=1 to p
                Count(i, j) =  Count(i-2, j-1) + Count(i-1, j)
        End for
End for

Return Count(n,p)
```

Rubric: -0.5 for missing any of the initializations, -3 if miss all

-2, Missing one of the loops or use one loop
-2, if use j=1 to i/2 or something similar which leaves a part of the subproblems unsolved.
-1 if use wrong recurrence formula

4) 11 pts

Given a connected undirected weighted graph with *n* nodes and *n-1* edges, describe a shortest path algorithm that runs in linear time with respect to *n*. (edge weights may be positive or negative)

Solution:

There are two cases:

1. One or more negative edge weights exist
2. No negative edge weights exist

In the first case, since this is a connected and undirected graph all paths can be made negative infinity long, since we can simply traverse one negative weight path forever.

In the second case, we can recognize that the graph is a tree, and that therefore the first path to any node is also the shortest. Therefore we can use BFS or DFS to find the shortest path.

Checking for negative weights takes O(n-1) = O(n), since we will have to look at each edge weight to see if it's negative.

BFS or DFS both take O(n+n-1) = O(n). Therefore, if we add both the checking for negative weights and BFS/DFS together, we get O(2n) = O(n), which is linear time.

It's possible to combine negative edge search and BFS/DFS by simply ending BFS/DFS once we hit a negative edge.

Rubric

Recognizing that graph is tree or equivalent consideration: 3 points
Describing how one or more negative weights lead to -INF: 3 points
Using BFS or DFS to deal with the case of only positive weights: 3 points
Runtime O(n) for finding negative weight(s): 1 point
Runtime O(n) for BFS/DFS: 1 point
Modifying BFS/DFS or adding different algorithms: -1 point

Not necessary: Pseudocode, proofs (since we can simply refer to BFS or DFS)

5) 15 pts

Let $v$ be a vertex in a connected network $G = (V, E)$. Describe an algorithm for finding, among all minimum spanning trees of $G$, one that minimizes the degree of $v$, in other words, the MST that has the minimum number of edges incident on $v$.

If all edges have distinct weights, only one MST exists. In this case, using Kruskal algorithm for example, only one order of edges is possible.

If edges with same weight exist, we must make sure we select those that are not connected to v first. This can be done in different ways, for example:

- **Solution 1**: Add a small enough value ε to every edge incident to $v$. The cost of any MST of this new network is (cost of original MST)+ ε (# edges from $v$ used).
- **Solution 2**: Modified version of Kruskal: sort the edges in non-decreasing order, making edges that contain v to be the last ones in the sequence of edges with the same weight.
- **Solution 3**: Modified version of Prim's: when adding a new edge to the tree, if there is more than one safe edge, select the one that is not adjacent to v.

Rubrics:

1- Figuring out that this problem can be solved by using Kruskal or Prim's algorithm (5 points).
2- Figuring out that the algorithm needs to consider edges with the same weight, i.e. non-unique MSTs (5 points).
3- The algorithm is complete and works for all cases (5 points).

Common Mistakes:

- Modify Kruskal or Prim's Algorithm in a way that it does not result in a MST for G, e. g. removing v, and adding it after running the algorithm without checking the weights.
- Use Kruskal or Prim's to get all different MST.

6) 15 pts

You are given n pairs of numbers. In every pair, the first number is always smaller than the second number. A pair (c , d) can follow another pair (a , b) in a chain if and only if b < c . Different chains of pairs can be formed in this fashion.

**This question is exactly the same as the Interval Scheduling problem in the book.**

a) Given a set of pairs, describe a greedy algorithm that finds the length of the longest chain which can be formed. You do not need to use up all the given pairs.

**Solution:**

- Sort pairs by the second number in a non-decreasing/increasing order. That is (x0,y0),(x1,y1)…(xn,yn) so that y0 <= y1 <= y2 <=…<= yn.
- Select the first pair, Then, iteratively select a pair from the above to form a chain only if the pair is compatible with the already chosen pairs (can compare with the last selected pair in the chain).

Psuedocode:

```
S <- Sort pairs of points by the second number in an increasing order.
A <- empty set (A is set of pairs selected)
A <- first pair
For j = 2 to n in S{
    If (pair j compatible with last pair in A):
            A <- A U (j)
}
Return |A|
```

Rubrics:

- Correct description of algorithm containing all information – 5 pts.
- Lack of crucial information in the algorithm description or do it other way would not yield an optimal solution or not proposing a greedy algorithm -2.5pts.

b) Prove that your algorithm is correct.

**Solution:** Need to show

- 1). **The algorithm gives non-overlapping pairs** (argue that the algorithm selects only pairs that are compatible to form a chain). – Here you have to clearly state that it is obvious from the algorithm in a) that the algo chooses a pair which does not overlap with the previously chosen pairs.
- 2). **The algorithm stays ahead of the optimal solution**.

- Let A = {i1,…ik} be the set of pairs selected by our greedy algo in order they are added. Let O = {j1,…jm} be the optimal solution, ordered by their second number. Let S(x) be a function that returns the second number of pair x.

- Base case n = 1: since our algo always pick the smallest second number. It is easy to see that S(i1) <= S(j1)

- Now, assume that it is true for r-1 and we will prove it for r. Our induction hypothesis states that S(ir-1) <= S(jr-1). So, any pair that is valid (compatible) to add to the optimal solution are also valid to add to our algorithm. Therefore, it must be the case for S(ir) <= S(jr).

- Now let's assume that our A is not optimal, that is, there is a pair r+1 in O that is not in A. We know that this pair's first number must be after the second number of r-th pair (S(jr)), as per the constraint. However, this pair would have also been compatible in A since S(ir) <= S(jr), and so our algorithm would have added this pair to A as well, a contradiction. Therefore, |O| = |A|.

**Rubrics:**

For 1).

- State clearly that the algorithm always selects compatible pairs to form a chain-proof of compatibility (2.5pts)
- State indirectly – 1 point.

For 2).

- Correct proof of optimality without omitting crucial information (2.5pts)
- At most 1 point will be given if the answer is somewhat correct.

c) What is the worst-case running time of your solution?

**Solution:** O(nlogn)

**Rubrics:**

- Correct answer. Only accept tight upper bound (i.e., O(nlogn))
- Partial Credit: 2.5pts for any time complexity that is upperbound, but not tight.
- Partial Credit: 1pt – for solution that is not tight and upperbound.

7) 10 pts
Consider the following divide and conquer algorithm:
The algorithm divides a problem of size n into 9 problems of size n/3. It then combines the subproblems in $O(n^2)$ time.

a) Does this algorithm run in $O(n^2 \log^2 n)$ time? Justify your answer.

Yes, in the worst case combine takes $\theta(n^2)$ time. So, this falls into case 2 which gives us a $\theta(n^2 \log n)$ for the run time which is also $O(n^2 \log^2 n)$ [notice that O( ) does not indicate a tight upper bound].

Rubrics:
+2 for Yes and +3 for proper justification.
Please note if the answer just uses master method to solve and leaves with a $\theta(n^2 \log n)$ without any explanation or a specific answer, it does not get any credits. It should solve the recurrence correctly and show understanding of the fact that it is covered under the loose bound of O().

b) Could this algorithm run in $O(n^2)$ time? Justify your answer.

Yes, the combining step takes $O(n^2)$ time which could really be much less than $\theta(n^2)$ say O(n) [notice that O( ) does not indicate a tight upper bound]. So, this could fall into case 1 which gives us a $\theta(n^2)$ for the run time which is also $O(n^2)$.

+2 for yes and +3 for justifying it.
If the justification involved modifying the number of sub-problems, or any assumptions that involved modifying the problem which is not given in the question, it is not a correct approach for this question. Saying that it is a yes when n=2 or less, i.e., for special cases only is not eligible as well. The answer should express the understanding that O(n^2) can include linear or constant time functions f(n) for which it will be considered as a candidate for case 1 of master theorem.

A no for either part (a or b), it will not get any partial credits.

Additional Space

Additional Space