CS570 Fall 2018: Analysis of Algorithms Exam I

	Points
Problem 1	20
Problem 2	13
Problem 3	17
Problem 4	15
Problem 5	15
Problem 6	20
Total	100

SOLUTIONS

Instructions:
1. This is a 2-hr exam. Closed book and notes
2. Any kind of cheating will lead to **score 0** for the entire exam and be reported to SJACS.
3. 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.
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.
8. You may not leave your seat for any reason unless you submit your exam at that point.

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

[TRUE]
Given an adjacency-list representation of a directed graph G = (V, E), it takes O(V+E) time to compute the in-degree of all vertices.

[FALSE]
An in-order traversal of a min-heap outputs the values in sorted order.

[TRUE]
$n^{\log_2 n} = \Theta(2^{(\log_2 n)^2})$.

[FALSE]
In a dynamic programming solution, the space requirement is always at least as big as the number of unique sub problems.

[TRUE]
For a binomial heap with the min-heap property, upon delete-Min, the order of the largest binomial tree in the binomial heap will never increase.

[TRUE]
The amortized cost of insertion into a binomial heap is constant.

[FALSE]
Dijkstra's algorithm may not terminate if the graph contains negative-weight edges.

[FALSE]
The longest simple path in a graph can be computed by negating the cost of all the edges in the graph and then running the Bellman-Ford algorithm.

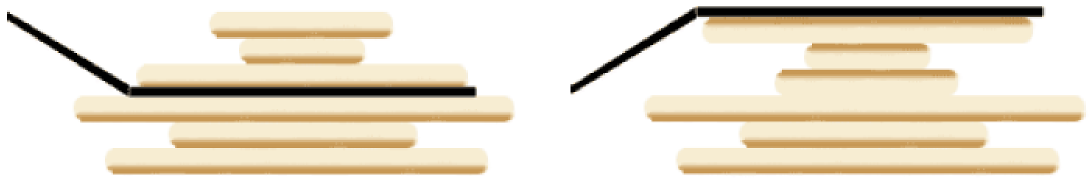
[TRUE]
If path *P* is the shortest path from *u* to *v* and *w* is a node on the path, then the part of path *P* from *u* to *w* is also the shortest path.

[FALSE]
The Floyd-Warshall algorithm always fails to find the shortest path between two nodes in a graph with a negative cycle.

Rubric: No partial credit – all or nothing.

2) 13 pts

Suppose you have a stack of pancakes of size *n* that you would like to sort from smallest to largest such that largest pancake is at the bottom of the stack after sorting. However, the only operation allowed on this data structure is flipping (aka making a sub stack up side down). Only the sub stacks that *contain the top element* can be flipped and we assume flip is an O(1) operation. As an example, here is what happens when the top 3 pancakes are flipped upside down. The left image shows the original pancake stack before flipping the top 3 pancakes. The right image shows the pancake stack after flipping the top 3 pancakes.



a) Design an algorithm to sort a stack of pancakes. Write the steps of your algorithm, either in plain English or in pseudo-code. (10 pts

Solution:

```
Let A[0 .. n] be an array of pancakes with A[0] on the top.
while (n >1)
     // finds the index of the max size pancake in A[0..n)
     int max = findmax(A,0,n);

    // flip the array A[0..max+1). The largest pancake will be on the top.
    flip(A,0,max);

    // flip the array A[0..n). The largest pancake will be on the bottom.
    flip(A,0,n-1);
    n = n-1;
```

b) Compute the runtime of your algorithm in terms of *n*. (3 pts)

Solution:

$O(n^2)$ The while loop will be run n times and findmax will be equivalent to linear search.

Other correct solutions:

1) Instead of finding the maximum size pancake, find the minimum size pancake, then flip. This method will need an extra flip operation in the end.

2) Bubble sort method:
say we want to swap s[i] and s[i+1], we can do
flip(0 to i+1)
flip(0 to 1)
flip(0 to i+1)
time complexity: O(n^2)

3) Insertion sort method:
say s[0] to s[i] has been sorted, we found a position k that s[i+1] > s[k] and s[i+1] < s[k+1],
we want to insert s[i+1] in front of s[k+1], we can do
flip(0 to i+1)
flip(0 to i+1-k)
flip(0 to i-k)
flip(0 to i+1)
Finding position k cost O(logn) with binary search, insertion cost O(1). Repeat n times, so total complexity is O(nlogn)

(Partial) Rubric:

Part a)
10 points - If the logic is correct. For pseudocode, if the logic is correct small code errors (range or value errors) will be ignored.

0 points- If using extra memory (max-heap or sorted array) to find the maximum element, and not updating during each iteration. After each flip, indices of pancakes will change, so using the initial sorted list won't work.
5 points - for giving a partially correct answer i.e for explaining how to find maximum size pancake and flipping the sub-stack to move the maximum size pancake to the top.

Part b)
3 points - for the correct complexity.
0 points - if algorithm in part a) isn't correct.

3) 17 pts.

Let us consider California coast with its many beautiful islands. We can picture the coast as a straight line: the mainland is on one side while the sea with islands is on the other side. Despite the remote setting, the residents of these islands are avid cell phone users. You need to place cell phone base stations at certain points along the coast so that every islands is within *D* miles of one of the base stations. Give an efficient algorithm that achieves this goal and uses as few base stations as possible. You may assume that each island has *xy*-coordinates (x_1, y_1), (x_2, y_2), …, (x_n, y_n).

Design a greedy algorithm to find the minimum number of stations required to cover all islands. (6 pts)

Solution:

1. Draw circle of radius D with each island as center. Calculate the left L_i and right R_i point of intersection of the *x*-axis, i = 1, 2, …., *n*.

2. Sort the intervals (L_i, R_i) by the right end.

3. Starting from the smallest R_i, install the radar at R_i and delete all intervals that overlaps with it. To delete we need to find R_j such that $L_j > R_i$ but $L_{j-1} < R_i$

4. Repeat until all intervals are deleted

(Partial) Rubric:

3 points - for drawing the circle and calculate the interval of x-axis. (step 1)

1 point - for sorting intervals. (step 2)

2 points - for greedy point placement. (step 3 and 4)

Note:

5 point - student only considers the right end point of the interval and forget to consider the left point to remove the islands already covered.

3 point - student sort by the x coordinate of the island. (start from the left island)

3 point - student put the radar station at the center of the inrerval.

No point deduction - student does not check for whether the circle intersects with coast/student use y-axis as the coast.

0 point - student does not transform the problem to interval-point greedy algorithm.

0 point - student transforms the problem to the wrong greedy algorithm.

b) Compute the runtime complexity of your algorithm in terms of *n*. Explain your answer by analyzing the runtime complexity of each step in the algorithm description. (5 pts)

Solution:

1. Complexity O(n)
2. Complexity O(n log n)
3. Complexity O(n)
4. Complexity O(n)

Total : O(n log n)

Rubric:

1 point - for step 1.

2 point - for step 2 (sort).

1 point - for step 3 and 4 (greedy).

1 point - for total.

c) Prove the correctness of your algorithm. (6 pts)

Solution:

Accept either a full proof (with induction) or a short one. The second one is below.

Let (I_1 , I_2 …. , I_m) be our solution and (J_1, J_2 …. , J_p) the optimal solution.

Consider the first stations I_1. The optimal solution J_1 cannot be on the left of our stations because its cover area will be less than D. Also J_1 cannot be on the right of I_1 because the first island will not be covered. Thus, $I_1 = J_1$.

We can apply the same argument to all other stations to prove that our solution size is equal to the optimal size.

(Partial) Rubric:

4 points - if mention the keywords: the proposed point is never to the left of the optimal solution. (Or in other words, the optimal solution is never to the right of the proposed solution.)

2 points - for proving the argument with contradiction.

3 points - in the first question, student sort by the x coordinate of the island. (start from the left island), student puts the radar station at the center of the interval.

No point deduction:

1) Student uses exchange to prove optimality.

2) Student only uses contradiction but does not use the complete form of induction.

3) Student does not describe the complete algorithm and get partial points in the first question. But they prove the optimality with contradiction.

4) 15 pts.
For each of the following recurrences, give an expression in the Theta notation for the runtime T(n) if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply.

1. $T(n) = 16\,T(n/4) + n / \log n$
2. $T(n) = 4\,T(n/2) + n^2 \log n - n$
3. $T(n) = 9\,T(n/3) - n^2 \log n + n$
4. $T(n) = T(n/2) + n! + 3n^4$
5. $T(n) = 1.2\,T(n/2) + n$

SOLUTIOIN

1. $T(n) = \Theta(n^2)$
2. $T(n) = \Theta(n^2 \log^2 n)$
3. does not apply
4. $T(n) = \Theta(n!)$
5. $T(n) = \Theta(n)$

(Partial) Rubric:

No partial credit – all or nothing. Accept either bigO or bigTheta notations.

5) 15 pts
Given a graph G = (V, E) where a set of cities V is connected by a network of roads E. Each road/edge has a positive weight, w(*u*, *v*) between cities *u* and *v*. There is a proposal to add a new road to the network. The proposal suggests a list C of candidate pairs of cities between which the new road may be built. Your task is to choose the road that would result in the maximum decrease in the driving distance between given city *s* and city *t*. Design an efficient algorithm for solving this problem, and prove its complexity in V, E and C.

Solution:

Algorithm:

1. Run Dijkstra algorithm from s to calculate shortest distances from s to all other cities
2. Run Dijkstra algorithm from t to calculate shortest distance from t to all other cities
3. For every candidate pair of cities {u,v}, the shortest path distance between s and t which covers road u → v is min(dist(s,u) + dist(t,v) + length(u,v), dist(s,v) + dist(t,u) + length(u,v)).
4. Choose the shortest distances from loop-3.
If the selected distance is longer than original dist(s,t), any candidate road cannot decrease distance between s and t.
Else, choose the {u,v} pair that produces shortest new distance. In the case of tie, choose one arbitrarily.
Complexity: Complexity of running Dijkstra's algorithm is O(E log V), the complexity of running step-3 is O(C) thus the total complexity is O(C + E log V).

(Partial) Rubric:

2 points - choosing the correct shortest path algorithm:
- 2 points for using Dijkstra
- Replacing Dijkstra with Bellman-Ford or Floyd-Warshall will afford partial credit

4 points - running the shortest-path algorithm twice:
- Should only run from "s" and from "t"
- no credit - running shortest-path for every node or Floyd-Warshall

6 points - goes through list "C" to determine best new edge to add:
Need to see something like this:
u → v, is min(dist(s,u) + dist(t,v) + length(u,v),dist(s,v) + dist(t,u) + length(u,v)).

3 points - correct running time:

  O(C + ElogV)

  Alternative running times for Dijkstra are acceptable

A NOTE ON SLOWER ALGORITHMS:

  An alternative solution that CORRECTLY solves the problem,
  but has a WORSE running time, you can get 4 pts, to a maximum of 6
  (on top of any correct parts they got from above)

  example: Using Dijkstra for every node and finding the distance with
  "u → v, is min(dist(s,u) + dist(t,v) + length(u,v),dist(s,v) + dist(t,u) +
  length(u,v))" will get you 6 points.

6) 20 pts

You are in Downtown of a city where all the streets are one-way streets. At any point, you may go right one block, down one block, or diagonally down and right one block. However, at each city block (i, j) you have to pay the entrance fees *fee*(i, j). The fees are arranged on a grid as shown below:

$$\begin{array}{c|c|c|c|c|c|c|}
 & 0 & 1 & 2 & 3 & & n \\
0 & fee_{(0,0)} & fee_{(0,1)} & fee_{(0,2)} & fee_{(0,3)} & \cdots & fee_{(0,n)} \\
1 & fee_{(1,0)} & fee_{(1,1)} & fee_{(1,2)} & fee_{(1,3)} & \cdots & fee_{(1,n)} \\
2 & fee_{(2,0)} & fee_{(2,1)} & fee_{(2,2)} & fee_{(2,3)} & \cdots & fee_{(2,n)} \\
3 & fee_{(3,0)} & fee_{(3,1)} & fee_{(3,2)} & fee_{(3,3)} & \cdots & fee_{(3,n)} \\
 & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
n & fee_{(n,0)} & fee_{(n,1)} & fee_{(n,2)} & fee_{(n,3)} & \cdots & fee_{(n,n)}
\end{array}$$

You start at (0,0) block and must go to (n, n) block. You would like to get to your destination with the least possible cost. Formulate the solution to this problem using dynamic programming.

a) Define (in plain English) subproblems to be solved. (3 pts)

Solution:

Let OPT(i, j) be the minimum cost to get from (0,0) to (i, j).

(Partial) Rubric:

3 points - Let OPT(i, j) be the minimum cost to get from (0,0) to (i, j).
3 points - Also correct: Let OPT(i, j) be the minimum cost to get from (i,j) to (n, n).
1 point - if they do not explicitly mention OPT(i, j) is the "minimum" cost.

full credit - If they define OPT(i, j) the minimum path from (0, 0) to (n, n), but it seems from their answers to the other parts that they have understood the problem, but this mistake was a matter of negligence, you can ignore and give them full credit for this part.

full credit - some students combine the subproblem definition and the recursive relation. Please comment on this, but if the answer is correct, it gets full credit.

b) Write the recurrence relation for subproblems. (7 pts)

Solution:

OPT (i, j) = min (OPT (i − 1, j) + fee(i, j), OPT (i, j − 1) + fee(i, j),
OPT (i − 1, j − 1) + fee(i, j))

OPT(-1, j) = OPT(i, -1) = BIG_CONSTANT
where $1 \leq i \leq n$ and $1 \leq j \leq n$

OPT(-1, -1) = OPT(0, -1) = OPT(-1, 0) = 0

(Partial) Rubric:

1) The recursive relation (4 points in total)
   Each term gets a credit:
   0) OPT (2.0 point) e.g., they identify the three cases but instead of OPT() they use fee()
   1) OPT (i − 1, j) (0.5 point)
   2) OPT (i, j − 1) (0.5 point)
   3) OPT (i − 1, j − 1) (0.5 point)
   4) fee(i, j) (0.5 point)

2) The base case (3 points in total)
1) OPT(-1, j) = OPT(i, -1) = OPT(-1, -1) = 0, [1<= i,j < n] (4*0.5 = 2 point) -
   There are four terms, each have 0.5.

2) OPT(i, -1) = BIG_CONSTANE (0.5 point)
3) OPT(-1, j) = BIG_CONSTANT (0.5 point)

Notes: Other correct base cases gets full credit.

Total: 7 points

c) Compute the runtime of the above DP algorithm in terms of *n*. (4 pts)

$O(n^2)$ all or nothing.

d) This problem can be represented as a graph problem and therefore solved using one of the shortest path algorithms. Which algorithm will you use and what is it runtime complexity in terms of *n*? (6 pts)

Solution:

The problem can be represented as a DAG and solved by topological sort.
The graph contains V = $O(n^2)$, E = 3 V. The runtime is $O(n^2)$. Both algorithms have the same run-time complexity.

(Partial) Rubric:

2 points - creating the graph, i.e., the set of nodes (1 point) and edges (1 point)
2 points - topological sorting

1 point - other correct algorithms such as Dijkstra/Bellman Ford
2 points - O(n2) Time complexity
1 point - if they use other algorithms, they may not get this time complexity. If the time complexity is correct but different than O(n2)