CS570 Spring 2018: Analysis of Algorithms Exam I

	Points
Problem 1	20
Problem 2	10
Problem 3	18
Problem 4	20
Problem 5	15
Problem 6	17
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]
If all the edge weights of a given graph are the same, then every spanning tree of that graph is minimum.

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

[TRUE/FALSE]
If all edges in a connected undirected graph have distinct positive weights, the shortest path between any two vertices is unique.

[TRUE/FALSE]
If a connected undirected graph $G(V, E)$ has $n = |V|$ vertices and n +10 edges, we can find the minimum spanning tree of G in $O(n)$ runtime.

[TRUE/FALSE]
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.

[TRUE/FALSE]
An amortized cost of insertion into a binomial heap is constant.

[TRUE/FALSE]
Gale-Shapley algorithm is a greedy algorithm.

[TRUE/FALSE]
For all positive functions $f(n)$, $g(n)$ and $h(n)$, if $f(n) = O(g(n))$ and $f(n) = \Omega(h(n))$, then $g(n) + h(n) = \Omega(f(n))$.

[TRUE/FALSE]
Any function which is $\Omega(\log \log n)$ is also $\Omega(\log n)$.

[TRUE/FALSE]
The depths of any two leaves in a binomial heap differ by at most 1.

2) 10 pts.
Consider a list data structure that has the following operations defined on it:

- *Append*(*x*): Adds the element *x* to the end of the list

- *DeleteFourth*(): Removes every fourth element in the list i.e. removes the first, fifth, ninth, etc., elements of the list.

Assume that *Append*(*x*) has a cost 1, and *DeleteFourth*() has a cost equals to the number of elements in the list. What is the amortized cost of *Append* and *DeleteFourth* operations? Consider the worst sequence of operations. Justify your answer using the accounting method.

Solution: *Append* gets charged 5 tokens. When we call *Append*, we spend 1 token immediately to pay for the cost of the call, we then store the remaining 4 tokens with the item added to the list. When we call *DeleteFourth* (), every element in the list has 4 tokens stored with it. We take the 4 tokens from each item that is deleted to pay for the cost of the call to *DeleteFourth* (). We can do this since n/4 items are deleted and so there are n tokens on these deleted items. Thus, at the end of the call to *DeleteFourth* () all remaining elements in the list still have 4 tokens stored on them. The amortized cost per operation is thus O(1)

3) 18 pts.
For each of the following recurrences, give an expression 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. $T(n) = 16\ T(n/4) + 5\ n^3$ solution: $T(n) = \theta(n^3)$.
2. $T(n) = 4\ T(n/2) + n^2 \log n$ solution: $T(n) = \theta(n^2 \log^2 n)$.
3. $T(n) = 4\ T(n/8) - n^2$ solution: does not apply
4. $T(n) = 2^n\ T(n/2) + n$ solution: does not apply
5. $T(n) = 0.2\ T(n/2) + n \log n$ solution: does not apply
6. $T(n) = 4\ T(n/2) + n/\log n$ solution: $T(n) = \theta(n^2)$.

4) 20 pts

You are given a set $X = \{x_1, x_2, \ldots, x_n\}$ of points on the real line. Your task is to design a greedy algorithm that finds a smallest set of intervals, each of length-2 that contains all the given points. Linked list is a data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of data and a reference (in other words, a link) to the next node in the sequence.

Example: Suppose that $X = \{1.5, 2.0, 2.1, 5.7, 8.8, 9.1, 10.2\}$. Then the three intervals [1.5, 3.5], [4, 6], and [8.7, 10.7] are length-2 intervals such that every $x \in X$ is contained in one of the intervals. Note that 3 is the minimum possible number of intervals because points 1.5, 5.7, and 8.8 are far enough from each other that they have to be covered by 3 distinct intervals. Also, note that the above solution is not unique.

a) Describe the steps of your greedy algorithm in plain English. What is its runtime complexity? (10 pts)

Sort X = $\{x_1, x_2, \ldots, x_n\}$. Let it be S.
Initialize T = {}
while S not empty:
    select the smallest x from S
    add [x, x+2] into T
    remove all elements within [x, x+2] from S
return T

b) Argue that your algorithm correctly finds the smallest set of intervals. (10 pts)

Assume there is an optimal solution O. We will prove that the size of our solution A is the same as the size of the optimal solution O.

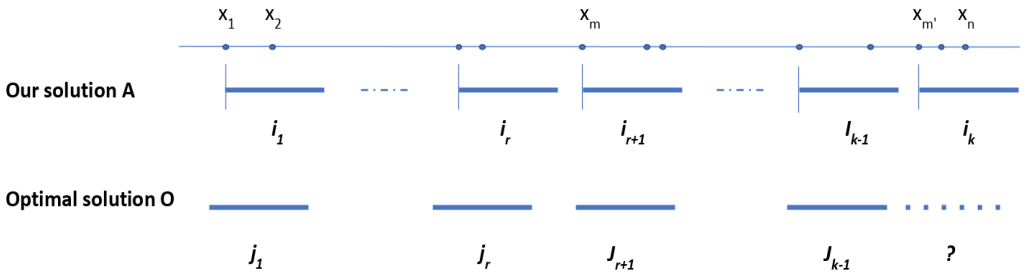
1- We will first prove that intervals in our solution (i1 ,i2 …. , in) are never to the left of the corresponding intervals in the optimal solution (j1 ,j2 …. , ip). We will use mathematical induction:

Base Case: i1 is not to the left of j1 since if it were then point x1 will not be covered by j1

Inductive step: Assume ir is not to the left of jr for r ≥ 1, we will prove that ir+1 is not to the left of jr+1

Proof: In above diagram, xm is the leftmost point covered by interval ir+1. Our solution A uses interval ir+1 to cover this point since interval ir was not covering it. And since ir is not to the left of jr then jr will not be able to cover xm either. So, if jr+1 is any further to right of ir+1 then xm will not be covered in O.

2- Now assume our solution needs k intervals and that the optimal solution O needs fewer intervals. We will prove that this is not possible:
Proof: Our solution needed interval ik because xm' (the leftmost point covered by ik) was not covered by ik-1 and since in step 1 we proved that our intervals are never to the left of the corresponding intervals in the optimal solution, then xm' will not be covered by jk-1 either. So the optimal solution will also need an interval jk to cover xm'. Therefore, the size of our solution is the same as the size of optimal solution.



5) 15 pts.

Given a $n \times n$ matrix where each of the rows and columns are sorted in ascending order, find the k-th smallest element in the matrix using a heap. You may assume that $k < n$. Your algorithm must run in time strictly better than $O(n^2)$.

Solution 1.

1. Build a min heap containing the first element of each row and then run deleteMin to return the smallest element.
2. If the element from the i-th row was deleted, then insert another remaining element in the same row to the min heap. Repeat the procedure k times.
3. The total running time is **O(n + k log n)**. Minheap is built in O(n) average and worst-time, since the input elements are pre-sorted. A deleteMin operation takes Θ(log n), since there are n elements in the . To find the kth smallest elements we require k-1 delete and insert operations.

Solution 2. (Using straightforward pruning: for a given integer k, no row or column in range[k+1,n] can have the kth smallest element)

The algorithm is identical to the one in solution 1 expect for the following changes.
If k<n, for any given k, there is no scenario in which rows k+1 to n can have the kth smallest element. This implies that I can prune these rows entirely. Accordingly, I will need to add only k elements (the first elements of the first k rows) to my min heap (in lieu of step 1), giving a total running time of **O(k + k log k)** since the heap now contains no more than k elements.

Solution 3. (Using same pruning as above but dumbing-down the solution to near brute force)

1. Insert k elements from each of the k rows into a minheap. Total k^2 elements in min heap => O(k^2),
2. and output the kth smallest element after k deletemin operations O(k log k).

Total running time: **O(k^2)**

Solution 4. (another pruning criterion) (-4 marks)

1. Insert k elements of first row into a max-heap. We will maintain this heap of fixed size k.
2. Iteratively visit each following row, reducing the number of elements visited by 1 in each iteration. (e.g. in second row visit k-1 elements in order, then in third row visit k-2 ,...)
3. for each new element encountered, compare with root node of max-heap. If smaller, remote root of max-heap, insert the new element in the heap.

In short, max heap maintains the k smallest elements yet seen.

Total running time: O(k^2 log k) (Not necessarily better than O(n^2))

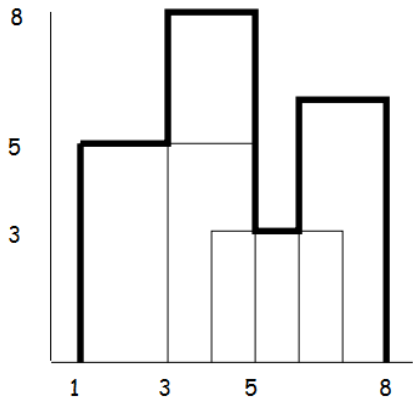
6) 17 pts

Suppose that you are given the exact locations and shapes of several rectangular buildings in a city, and you wish to draw the skyline (in two dimensions) of these buildings, eliminating hidden lines. Assume that the bottoms of all the buildings lie on the x-axis. Each building B_i is represented by a triple (L_i, H_i, R_i), where L_i and R_i denote the left and right x coordinates of the building, respectively, and H_i denotes the building's height. A skyline is a list of x coordinates and the heights connecting them arranged in order from left to right.

For example, the buildings in the figure below correspond to the following input

(1, 5, 5), (4, 3, 7), (3, 8, 5), (6, 6, 8).

The skyline is represented as follows: (1, 5, 3, 8, 5, 3, 6, 6, 8). Notice that in the skyline we alternate the x-coordinates and the heights. Also, the x-coordinates are in sorted order.



a) Given a skyline of n buildings in the form $(x_1, h_1, x_2, h_2, \ldots, x_n)$ and another skyline of m buildings in the form $(x'_1, h'_1, x'_2, h'_2, \ldots, x'_m)$, show how to compute the combined skyline for the $m + n$ buildings in $O(m + n)$ steps. (5 pts)

Merge the "left" list and the "right" list iteratively by keeping track of the current left height (initially 0), the current right height (initially 0), finding the lowest next x-coordinate in either list; we assume it is the left list. We remove the first two elements, x and h, and set the current left height to h, and output x and the maximum of the current left and right heights.

b) Assume that we have correctly built a solution to part a), design a divide and conquer algorithm to compute the skyline of a given set of n buildings. Your algorithm should run in $O(n \log n)$ steps. (12 pts)

If there is one building, output it.
Otherwise, split the buildings into two groups, recursively compute skylines, output the result of merging them using part (a).
The runtime is bounded by the recurrence
$T(n) \leq 2T(n/2) + O(n)$, which implies that $T(n) = O(n \log n)$.