CSCI 570

Exam 1 Solutions

Problem 1 (Questions 1-10**)** (Random Questions)[Any 10 from the below 14 Questions]
Rubric: 2 points each correct Answer. Max 20 points

1) Height of any complete binary tree with n nodes is O(n) – True

2) Any instance of the stable matching problem will have at least two different stable matches. – False

3) $n^2 * (log n)^{10} = O(n^{2.1})$ – True

4) Given a binary max heap with k levels (with root at level 1), the smallest element of the heap will be on level k. – False

5) Given a binary min heap of size n, we can construct a sorted list of the n elements in O(n) time. – False

6) Function $10n^{10}2^n + 3^n\log(n)$ is $O(n^{10}2^{n+1})$ – False

7) Given a directed graph G with n vertices, any BFS tree corresponding to graph G will have n-1 edges – False

8) For the recurrence equation $T(n) = 4T(n/2) + cn^2$, and for a positive constant c, $T(n) = O(n^3)$ – True

9) In the Gale-Shapley algorithm we will always have to do the same number of iterations whether men are proposing, or women are proposing – False

10) Dijkstra's algorithm works correctly on a directed acyclic graph even when there are negative-weight edges – False

11) Amortized cost of an operation in a data structure is always lower than its worst-case cost – False

12) The first edge added by Kruskal's algorithm can be the last edge added by Prim's Algorithm – True

13) In a binary max-heap, the kth largest element cannot appear anywhere below the kth level. (root is at level 1) – True

14) If f(n) is O(g(n)), if n is large enough, we always have f(n)<=g(n) – False

Problem 2 (Question 11) (Random Questions)[Any 1 from the below 2 Questions]

Rubric: 5 points each correct Answer. Max 5 points

1) The function below has a worst-case run time performance of $\Omega(n*\log n)$.

```
function(n):
cnt = 0
i = n
while i > 1
for j=1 to i
cnt = cnt + 1
        endfor
i = floor(i / 2)
    endwhile
    return cnt
```

False

2) The function below has a worst-case run time performance of Θ (n).

```
function(n):
cnt = 0
i = n
while i > 1
for j=1 to i
cnt = cnt + 1
        endfor
i = floor(i / 2)
    endwhile
return cnt
```

True

Problem 3(Question 12-14):
Farmer John has N cows (1, 2,…, N) who are planning to escape to join the circus. His cows generally lack creativity. The only performance they came up with is the "cow tower". A "cow tower" is a type of stunt in which every cow (except for the bottom one) stands on another cow's back and supports all cows above in a column. The cows are trying to find their position in the tower. Cow i(i = 1, 2, … , N) has weight Wi and strength Si. The risk of cow i failing (Ri) is equal to the total weight of all cows on its back minus Si.

Question 12: Design an algorithm to find the positions of all cows in the tower such that we minimize the maximum "risk value" of all cows. 10 pts

Solution:

Sort all the cows by key (Wi+Si) in ascending order and arrange them from top to ground (the top is the cow with the smallest 'key'). This ordering of cows will minimize the maximum risk value.

Rubrics:

(3 pts**)** Using greedy algorithm
(4 pts) Having the correct key(Wi+Si)
(3 pts) Having the correct order: sorting in ascending order, smallest 'key' on the top

Question 13 Prove the correctness of the solution. 8 pts

Solution:
The proof is similar to the proof we did for the scheduling problem to minimize maximum lateness. We first define an inversion as a cow i with higher (Wi+Si) being higher in the tower compared to cow j with lower (Wj+Sj). We can then show that inversions can be removed without increasing the maximum risk value time. We then show that given an optimal solution with inversions, we can remove inversions one by one without affecting the optimality of the solution until the solution turns into our solution.

1. Inversions can be removed without increasing the risk value.
   Remember that if there is an inversion between two items a and b, we can always find two adjacent items somewhere between a and b so that they have an inversion between them. Now we focus on two adjacent cows (one standing on top of the other) who have an inversion between them, e.g. cow i with higher (Wi+Si) is on top of cow j with lower (Wj+Sj). Now we show that scheduling cow i before cow j is not going to increase the maximum risk value of the two cows i and j. We do this one cow at a time:

- By moving cow j higher we cannot increase the risk value of cow j

- By moving cow i lower (below cow j) we will increase the risk value of cow j but since the since cow i has a higher total weight and strength than cow j, the risk value of cow i (after removing the inversion) will not be worse than the risk value for cow j prior to removing the inversion:

  Risk value of cow j before removing the inversion: Wi + weight of remaining cows above i&j - Sj

  Risk value of cow i after removing the inversion: Wj + weight of remaining cows above i&j - Si

  Since Wi+Si >= Wj+Sj , if we move Si and Sj to the opposite sides we get: Wi-Sj >= Wj-Si. (weights of the remaining cows above the two cows i and j does not change). In other words, the risk value of cow j before removing the inversion is higher than that of cow i after removing the inversion

2. Since we know that removing inversions will not affect the maximum risk value negatively, if we are given an optimal solution that has any inversions in it, we can remove these inversions one by one without affecting the optimality of the solution. When there are no more inversions, this solution will be the same as ours, i.e. cows sorted from top to bottom in ascending order of weight+strength. So our solution is also optimal.

Rubrics:

(3 pts) The answer uses inversions (switching the positions) of two adjacent cows to prove the correctness of greedy algorithm

(5 pts) It proves that using the our rule, the maximum risk of two adjacent cows after switching the position is no smaller than before, and this rule can be extended to the whole group of cows

Question 14 What is the worst-case time complexity of your algorithm? 2 pts

Solution:

O(nlogn)
sort take O(nlogn), then we could get the maximum risk in one pass

Rubrics:

Give 2 pts when the answer is O(nlogn) and question 13 is correct, or the complexity is correct for the given algorithm but question 13 is not correct

Give 0 pts when the given time complexity is not correct for the given algorithm, or there is no algorithm

Problem 4 (Question 15-16):

Given an array of integers A consisting of N integers, the task is to minimize the sum of the given array by halving the value of at most K elements, halving the value of an element involves reducing element Ai to floor(Ai/2). We know that K >= 0.001N. We want to achieve a worst case run time complexity of O(KlogK).

Question 15:

Present your solution (8 pts)

Answer: The main idea is to reduce the largest K elements in the array.

Solution 1:

- Create a Max-Heap of size N for the input array A
- Pop the root r of the Max-Heap (using `ExtractMax`), halve the value of r and place it in position "end-of-heap+1" in the array
- After Repeating the above step K times, sum up all the terms in A
- Return the sum

Solution 2:

- Sort the array A
- Halve the values of the largest K elements in A and sum up all the terms in A
- Return the sum

Solution 3:

- Create a Min-Heap of size K in array A using the first K elements of the array
- Compare the remaining N-K elements of A with the root r of the Min-Heap. If we find a new element > r, then replace the root r with new element x (extract_min r, insert x, place r in x's old position in A)
- After repeating the above step (N-K) times, halve all first K elements in A, and sum up all the terms in A
- Return the sum

Rubric:

- -2 pts if for any minor mistake (e.g. only find the largest K elements but not the sum)
- Partial credit: 2 pts if the basic idea is correct (halving the largest K elements), but the output is suboptimal or the algorithm is unclear; or the run time complexity is not in `O(KlogK)`
- 0 if the algorithm is wrong or incomplete

Question 16:

Explain why the complexity of your solution is O(KlogK) (2 pts)

Answer:

Solution 1:

Max-Heap requires O(N); Repeat ExtractMax K times require O(K log N). So the total run time is O(N) + O(K log N) = O(K log K) since N = O(K).

Solution 2:

Sorting requires O(N log N) = O(K log K) since N = O(K).

Solution 3:

Min-Heap requires O(K); Repeat ExtractMin and Insert requires (N-K) log K. So the total run time is O(K) + O((N-K) log K) = O(K log K) since N = O(K).

Rubric: 2 pts for correct explanation, 0 pt for wrong or incomplete analysis.

Problem 5(Question 17-18):

Given the x coordinates of a set of points in 1D, use divide and conquer, we want to find the closest pair of points in the set.

Question 17

a - Design a divide and conquer algorithm to find the closest pair of points. (8 pts)

Solution:

Step 1: First sort all points ascendingly according to the x coordinates.

Step 2: Divide into two equal sets of points, one set P1 is the left half and the other set P2 is the right half.

Step 3: Look at 3 cases similar to 2D problem

In case 3 the 2 pts should be the leftmost point in P2 and rightmost point in P1. So there is no searching involved.

Step 4: Accept the minimal distance among the 3 cases.

Rubrics:
8 points for a correct algorithm, 0 point for wrong, incomplete or illegible answers.

Partial credit: 2 points for a correct but suboptimal algorithm.

In the grading feedback, "comb" means "combination" or "merge"

Question 18
b - What is the run time complexity of your algorithm? (2 pts)

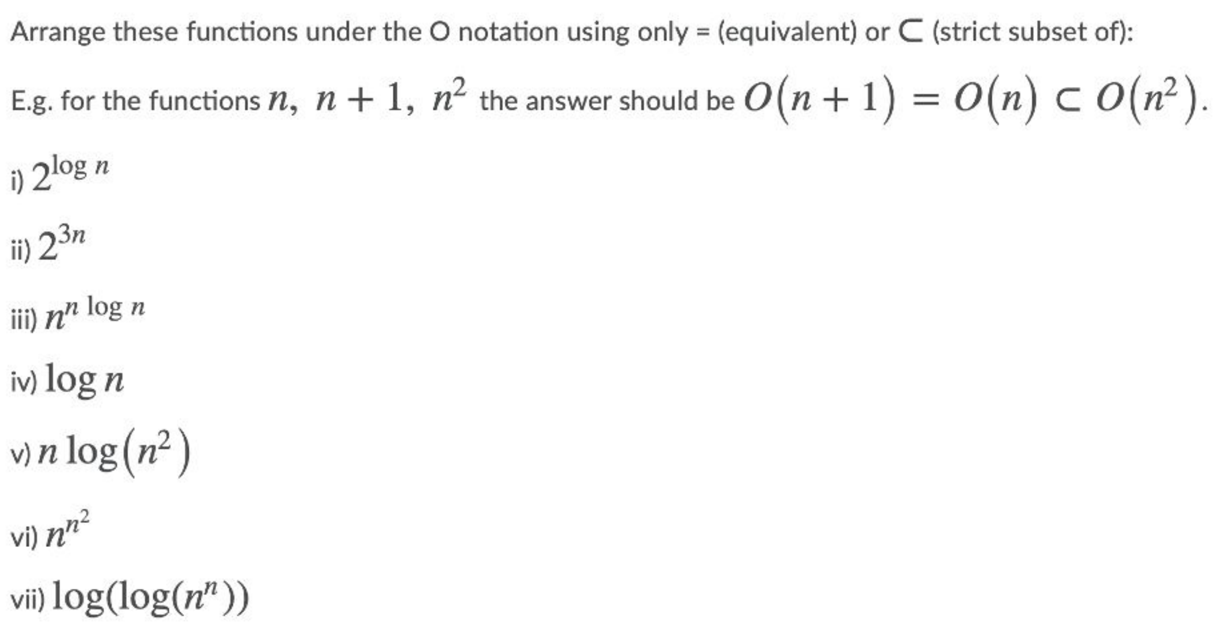
Solution:

Divide and conquer part takes O(n) (case 1 of master theorem) but overall complexity will be O(n log n) because of sort

Rubrics:
2 points for answering O(n log n), 0 point for wrong, illegible or not tight enough answers.

Partial credit: 1 point for a correct and tight answer, but not a simplified expression. For example, 1 point for answering O(n log (n^2)) or O(n + n log n).

Problem 6 (Question 19):



Arrange these functions under the O notation using only = (equivalent) or $\subset$ (strict subset of):

E.g. for the functions $n,\ n+1,\ n^2$ the answer should be $O(n+1) = O(n) \subset O(n^2)$.

i) $2^{\log n}$

ii) 2^{3n}

iii) $n^{n \log n}$

iv) $\log n$

v) $n \log(n^2)$

vi) n^{n^2}

vii) $\log(\log(n^n))$

Solution:

logn = log(log(n^n)) < 2^(logn) < nlog(n^2) < 2^(3n) < n ^ (nlogn) < n ^ (n ^ 2)

4 = 7 < 1 < 5 < 2 < 3 < 6

Rubrics:

10 points for everything correct (including correct notations)
8 points for order correct but not equated 4 and 7
6 points if one "inversion" in the order
4 points if one inversion and not equated 4 and 7
3 points if two inversions in the order
0 for incorrect ordering

Problem 7 (Question 20):

Suppose in a stable matching problem, there are n men and n women. Prove or disprove: It is possible that one man has 2 valid partners and each of the other n-1 men have 1 valid partner.

Solution:

Using Proof by Contradiction:

If there is a man m who has two valid partners, there must be at least two stable matchings S and S' for this set of preference lists. Suppose man m is paired with one of his valid partners w in S.

And man m is paired with the other valid partner w' in S'. So, w' must be a valid partner of another man m' in S. This leads to a contradiction since we are assuming that every man m' (i.e. other than m) only has one valid partner. So if w' is m' 's only valid partner, then m and w' cannot be paired together in S'.

Or using Direct Proof:
If every man other than m only has one valid partner, then it means that n-1 of the men are paired with exactly the same partner in every stable matching. This only leaves one possible partner for m since the other n-1 women are each the only valid partners of one of the n-1 men. So m cannot have more than 1 valid partner either.

Rubrics:

5 points for getting disproving statement correctly
0 points otherwise (No Partial Credit)

Problem - 8 (Question 21-22)

We have a book written in English and we want to translate it into n other languages. We know the translation cost of this book from any language to any other language.

Question 21:

Find a polynomial time algorithm to compute the minimum total cost of translating the book into all n other languages. (8 pts)

Solution: We can construct an undirected weighted graph (V, E) with each language being a node and the translation costs being the edge weights. Then we can run the MST algorithm on this graph and take the total weights of MST as the translation cost.

Rubrics:
For MST algorithm:

3 points for constructing the graph.

3 points for running MST.

2 points for returning the weights of MST as the translation cost.

If you use other algorithms:

Full points if the algorithm is different but correct.

No point if the algorithm is wrong.

Question 22:

What is the worst case run time complexity of your solution? (2 pts)

Solution: The time complexity is O(V+E+ElogV)=O(ElogV).

Rubrics: The running time can be one of the following values:

1. O(ElogV). Prim's algorithm with a min-heap priority queue.
2. O(ElogE) or O(ElogV). Kruskal's algorithm.
3. O(E^2). Reverse-delete.
4. O(V^2). Prim's algorithm. Implement the priority queue as array [1, …, V] where element i stores the cheapest edge from tree to node i; or use Fibonacci heap.

You still get full points if your section (a) is wrong but (b) is the correct running time of (a).

No point if the time complexity is better than O(V^2) or O(E) as reading in the edges already takes O(E) time.

Problem - 9 (Question 23 - 24)

Given a directed graph G=(V, E) with |V|=n, |E|=m and edge weights take only two values, 5 and 10. Let s, t be two vertices in G.

Question 23:

Design an O(n+m) algorithm that finds the shortest path from s to t (if it exists). (8 pts)

Solution:
Algorithm:
Split all edges of weight 10 (u, v) into two edges (u, w) and (w, v) with weight 5 for each.
Run BFS algorithm to find the shortest path.
Remove all intermediate nodes in the output path and multiply the distances found by BFS by 5 to get actual distances.

Rubric:
BFS: 1 point
Split edges: 4 points
Interpret BFS result: 3 points

Any other correct algorithm with time complexity greater than O(n+m): 2 points in total and no credit for part b).

Using Dijkstra's algorithm will exceed O(n+m). There are many answers mentioning using two sets to store edges, but it still needs a min-heap to store tentative distances of all unvisited nodes, e.g. [5, 10, 15, inf, inf], which increases the time complexity.
Using Prim's/Kruskal's algorithm to build MST can't guarantee the shortest path from s to t.
Using topological sorting only works on directed acyclic graph(DAG).

Question 24:

b) Show that the run time complexity of your solution is O(n+m). (2 pts)

Solution:
Split edges: O(m) -- 1 point
BFS: O(m+n) -- 1 point
Total complexity: O(m+n)