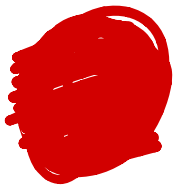
CSCI-570 Spring 2023

Midterm 1



INSTRUCTIONS

- The duration of the exam is 140 minutes, closed book and notes.
- No space other than the pages on the exam booklet will be scanned for grading! Do not write your solutions on the back of the pages.
- If you require an additional page for a question, you can use the extra

page provided at the end of this booklet. However please indicate clearly that you are continuing the solution on the additional page.

1. True/False Questions (10 points)

Mark the following statements as **T** or **F**. No need to provide any justification.

a) **(T/F)**. Prim's algorithm cannot be applied to directed weighted graphs.

True. Greedy approach.

b) **(T/F)**. For a dynamic programming algorithm, computing all values in a bottom-up fashion is asymptotically faster than using recursion and memoization.

False. A bottom-up implementation must go through all of the subproblems and spend the time per subproblem for each. Using recursion and memoization only spends time on the subproblems that it needs. In fact, the reverse may be true: using recursion and memoization may be asymptotically faster than a bottom-up implementation.

c) **(T/F)**. In an undirected weighted graph, the shortest path between two nodes always lies on some minimum spanning tree.

False. One can provide a simple example of three nodes with weights 2, 3 and 4.

d) **(T/F)**. For a graph with nonnegative edge weights, when a node u is removed from the priority queue in Dijkstra's algorithm its distance label is correct (i.e., equal to the length of the shortest su path.)

True. This is exactly the invariant maintained by Dijkstra's algorithm.

e) **(T/F)**. We have n operations, each of which takes amortized $O(1)$. Then, the worst-case running time for any single operation can be $\Theta(n^2)$.

False. Since the n operations have amortized running time O(1), their total running time is at most $O(n)$.

f) **(T/F)**. The spanning tree of maximum weight in G is the minimum spanning tree in a copy of G with all edge weights negated.

True. Minimum spanning algorithms only consider relative edge order so they still, work with negative edge weights to produce a minimum spanning tree. The spanning tree T minimizes $w(T)$ maximizes $w(T)$.

g) **(T/F)**. Suppose G is a weighted undirected graph with positive edge weights, where each edge $e \in E$ has weight w_e, and let G' be a graph that is identical to G except that every edge e has weight w_e^2. Any MST of G' is an MST of G.

True. One way to see this is that the ordering of edges by weights remains the same, so the set of MSTs that can be output by Kruskal's is the same.

h) **(T/F)**. Let $T(n) = 3T(\frac{n}{3}) + O(\log n)$ be a recurrence equation. Then we can conclude that $T(n) = \Theta(n \log n)$ by the Master theorem.

False. Case 1 of the master theorem: $a = 3, b = 3$.

i) **(T/F)**. Any function which is $\Omega(\log(\log n)))$ is also $\Omega(\log n)$.

False. If a function is $\Omega(\log n)$, it can not be $\Omega(\log(\log n)))$.

j) **(T/F)**. $\sqrt{\log_2 n} = \Omega(n^{\frac{1}{\log_2 n}})$.

True. the right-hand side is a constant.

2. Multiple Choice Questions (10 points)

Please select the most appropriate choice. Each multiple choice question has a single correct answer.

a) Suppose that a binomial heap H has a total of n nodes. The maximum number of binomial trees H can contain is the following:

a) $floor(\log n) + 1$
b) $floor(n \log n)$
c) $floor(\log n) - 1$
d) $floor(\log n)$

a. $floor(log(n)) + 1$. Since the binary representation of n has at most $floor(log(n)) + 1$ digits and each digit can represent a binomial tree.

b) In the lecture we discussed the runtime complexity of Dijkstra's algorithm when it's implemented using a binary heap. What would be the algorithm runtime complexity if we replace a binary heap by a Fibonacci heap? Select the tightest upper bound.

a) $O((E + V) \log V))$
b) $O(E + V)$
c) $O(V + E \log V)$
d) $O(E + V \log V)$

d.

c) The solution to the recurrence relation $T(n) = 8T(n/4)+O(n \log n)$ by the Master theorem is:

a) $\Theta(n^2)$
b) $\Theta(n^2 \log n)$
c) $\Theta(n \log n)$
d) $\Theta(n \log^2 n)$

a.

d) Analyze the worst-case complexity of the following code snippet and select the tightest upper bound.

```
for(int i = 1; i < n; i++)
    for(int j = 1; j < i; j++)
        j = j * 2;
```

a) $O(n)$
b) $O(\log n)$
c) $O(n^2)$
d) $O(n \log(n))$
d.

e) Analyze the worst-case complexity of the following code snippet and select the tightest upper bound.

```
while(n > 0){
   for(int i = 0; i<n; i++){
       System.out.println("#");
   }
   n = n/2;
}
```

a) $O(n)$
b) $O(\log n)$
c) $O(n^2)$
d) $O(n \log(n))$
a.
Starting from n, the inner for loop runs n times. Second time, it runs $n/2$ times, and then $n/4$ times and so on. $n + n/2 + n/4 + ... + 2 + 1 = 2n - 1 = O(n)$

3. Short Question (15 points)

Suppose there are two algorithms T_1 and T_2. The first algorithm has a runtime complexity defined by $T_1(n) = 5T_1(n/2) + O(n^2)$. The second algorithm has a runtime complexity defined by $T_2(n) = x\,T_2(n/4) + O(n^2 \log n)$, where $x > 0$ is an unknown positive real number.

What are the values of x such that T_2 is asymptotically faster than T_1? You may express your answer using inequalities for x. Explain your answer.

When $0 < x < 25$. This actually is broken into three cases: $0 < x < 1$, $1 \leq x < 16$ and $16 \leq x < 25$, since each of them is solved differently by the Master theorem

- An answer with range : $0 < x < 25$ is worth full credits (15 pts). If the answer is different, allocate the points based on the following criteria:
- Range: $1 <= x < 16$ (+6.5 pts)
- Range: $16 <= x < 25$ (+6.5 pts)
- Range: $0 < x < 1$ (+2 pts)
- Eg: An answer with range $0 < x < 16$ will receive 6.5+2 = 8.5 pts

$$n^{\log_2 5} = n^{\log_4 x} = n^{\frac{\log_2 x}{\log_2 4}} = n^{\frac{\log_2 x}{2}}$$

$$\log_2 5 = \frac{\log_2 x}{2}$$

$$2\log_2 5 = \log_2 x$$

$$\log_2 5^2 = \log_2 x$$

$$x = 5^2 < 25$$

4. Dynamic Programming Algorithm (20 points)

You are planning to establish a chain of electric vehicle charging stations along the Pacific Coast Highway. There are n potential locations along the highway, and the distance between the starting point and location k is $d_k \geq 0$, where $k = 1, 2, \ldots, n$. It is assumed that $d_i < d_k$ for all $i < k$. These are important constraints:

- At each location k you can open only one charging station with the expected profit p_k.
- You must open at least one charging station along the whole highway.
- Any two stations should be at least M miles apart.

Design a dynamic programming algorithm to maximize the profit by following these steps:

a) Define (in plain English) the sub-problems to be solved (3 points).

b) Write a recurrence relation for the sub-problems. Make sure you specify base cases. (10 points).

c) Using the recurrence formula in part b, write an iterative pseudocode to find the solution. (3 points).

d) What is the complexity of your solution? Explain your answer. (4 points)

OPT[k] ⇒ max profit

2 choices

Case 1 : $\max\{OPT[k-1], P_k + OPT[f(j)]$ where $f(j)$ is used to find the index j such that $d_j \leq d_{k-M}\}$

Case 2 : If there exists no such j - i.e $f(j)=0$

$\max\{OPT[k-1], P_k + OPT[0]\}$

Basecase :

$OPT[0] = 0$

$OPT[i] = P[i]$

Subproblem: Let $OPT(k)$ be the maximum expected profit that you can obtain from locations $1, 2, ..., k$.

Recurrence Relation:
Case 1: $OPT(k) = \max\{OPT(k-1),\ p_k + OPT(f(j))\}$, where $f(j)$ finds the largest index j such that $d_j \leq d_k - M$;
Case 2: When such j doesn't exist, $f(j) = 0$, then :
$OPT(k) = \max\{OPT(k-1),\ p_k\}$.

Base cases: $OPT(0) = 0, OPT(1) = p_1$.

Pseudocode:
Input: n locations; $P[1...., n]$ where $P[k]$ denotes profit at location k
Output: Maximum expected profit P_{max}
Create an array of maximum expected profit $OPT[0, 1..., n]$:
$OPT[k]$ denotes maximum expected profit at location k
$OPT[0] = 0$, $OPT[1] = P[1]$

for $k = 2 \rightarrow n$:
  $j\_exists = false$
  for $j = 1 \rightarrow k$–1:
    if $(d_j \leq d_k – M)$
      $j\_exists = true$
      $OPT[k] = max(OPT[k-1], P[k] + OPT[j])$
  if $(!j\_exists)$ #when the stations are not atleast M miles apart
    $OPT[k] = max(OPT[k-1], P[k] + OPT[0])$
return $OPT[n]$

Runtime: The index $f(k)$ can also be found using linear search, which takes $O(n)$ time. Thus, the total run time can also be $O(n^2)$. Additionally, the algorithm solves n subproblems; each subproblem

requires finding an index $f(k)$ which can be done in time $O(\log n)$ by binary search (as distances are provided in increasing order). Hence, the running time is $O(n \log n)$.
Further, using a queue data structure while solving the subproblem, we can entirely compute the maximum profit in linear time. The enqueue and dequeue operations are at most $O(2n)$. In this case, total runtime is $O(n) + O(2n) = O(n)$.

- Correct subproblem definition (3pts)
- Correct Recurrence Relation for case 1 along with the feasible condition (5pts)
- Correct Recurrence Relation for case 2 along with the feasible condition (5pts)
- Initialization of base cases for the pseudo code and progressing the iteration from the valid indices of k and j (3pts)
- Any of the following provided runtimes with corresponding valid explanation [Must include linear search or binary search or using queue data structure respectively for award] (4pts)
- For a 2 State Subproblem definition $OPT[k][d_n]$, with a pseudo polynomial runtime of $O(n * d_n)$ award partial score (not an ideal formulation of subproblem)(10pts)

5. Amortised Cost Analysis (15 points)

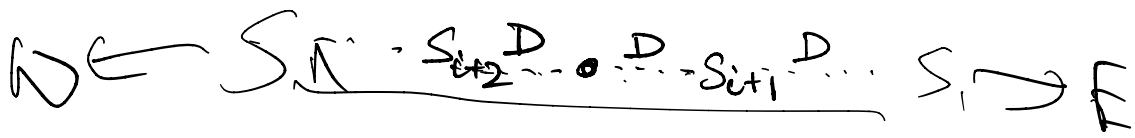
In a web caching software, there are n URLs stored in a doubly linked list with 'head' and 'tail' pointers, pointing to the start and the end of the list respectively. Whenever a new URL is cached, it is stored (or inserted) at the end of the list in constant time using the tail pointer. Whenever any URL (say 'www.usc.edu') is searched in the cache, the search starts from the head of the list and then the list is traversed until the target URL is found. If the target URL is found at a location i, where i≤n, the search is stopped (Call it the 'search' operation) and the target URL is moved towards the head of the list by 1 index (Call it the 'move' operation). The cost for each 'search' operation would be i (i.e., original index of the target URL) and the cost for each 'move' operation is c (where c is a positive constant).

Given that 'www.usc.edu' is present in our cache, calculate the amortized time complexity for searching 'www.usc.edu' considering the worst-case scenario. You can assume that the number of times the URL 'www.usc.edu' is searched would be less than or equal to the total number of URLs already cached in the list, and you can also assume that you don't search for any websites other than 'www.usc.edu'.

Taking the worst-case scenario, we will consider that 'www.usc.edu' is cached at the end of the doubly linked list (i.e., at index n). Let's say that 'www.usc.edu' is searched n times. When the URL 'www.usc.edu' is searched for the first time, it's search cost will be n. Next time it will be (n-1) and so on.
Additionally, cost to move the target element would be c every time (except when the target URL is at position 1). So, for n-1 moves, it will be (n-1)*c
Approximately after n searches, the URL 'www.usc.edu' would be placed at index 1.
T(n) = n + (n-1) + (n-2) + ... 1 + (n-1)*c = (n*(n+1))/2 + (n-1)*c

Therefore, T(n)/n = O(n)

- An answer with cost O(n) with proper steps is worth full credits (15 pts). If the answer is different (or missing steps/ explaination), allocate the points based on the following criteria:
- Correct identification of worst case scenario (+6 pts)
- Correct T(n) calculation (+2pts)
- Final answer is worth 7 points, so an answer with incorrect cost may be get upto 8 points based on the above criteria.



6. Greedy Algorithm (15 points)

In a lawn, there are n lawn sprinklers S_1,...,S_n installed on the center line of the lawn. Every sprinkler sprays D meters long in the same fixed direction from east to west. The sprinkler S_1 is at the east borderline of the lawn. Each sprinkler S_i $(2 \leq i \leq n)$ is $d_i \leq D$ meters away from the previous sprinkler S_{i-1}. The sprinkler S_n is at the west borderline of the lawn. What is the minimum number of sprinklers you must open to cover the lawn from its east side to its west side?

a) Design an algorithm (you can provide a pseudo-code) that solves the above optimization problem. (6 points)

b) What is the asymptotic running time of your algorithm in terms of n? (2 points)

c) Prove the correctness of your algorithm. (7 points)

Our greedy strategy is to always open the next sprinkler as late as possible.

Pseudo-Codes

```
1  d = 0, k = 1
2  for i = 1 to n − 1
3        if d + d_{i+1} > D
4              d = 0, k = k + 1
5        d = d + d_{i+1}
6  return k
```

Proof of why our algorithm is optimal:

Induction on $D_{Ours}(m)$, the maximum total distance that can be covered by m sprinklers selected by our algorithm. Let $D_{OPT}(m)$ denote

the maximum total distance that can be covered by m sprinklers in the optimal solution.

Base Case: The first sprinkler will always be opened and $D_{Ours}(1) = D_{OPT}(1) = D$. $D_{Ours}(2) \geq D_{OPT}(2)$ because our algorithm opens the second sprinkler that is as far as possible from the first sprinkler.

Induction Hypothesis: Assume for $m = k$, $D_{Ours}(k) \geq D_{OPT}(k)$.

The sprinkler opened at kth step in optimal solution and our algorithm could cover at most D meters. For $(k+1)$th stop, our algorithm will select the last sprinkler before $D_{Ours}(k) + D$. The $(k+1)$th sprinkler in the optimal solution is before $D_{OPT}(k) + D$. Since $D_{Ours}(k) + D \geq D_{OPT}(k) + D$, the $(k+1)$th sprinkler selected by the optimal solution will be no further than the $(k+1)$th sprinkler selected by our algorithm. Thus, $D_{Ours}(k+1) \geq D_{OPT}(k+1)$.

Since our algorithm can always cover at least as much distance as the optimal solution with m sprinklers, then our algorithm is guaranteed to use no more stops than OPT on any fixed distance. Hence, our algorithm returns the optimal solution.

The running time of the algorithm is $O(n)$.

Rubrics.
Part a.

- 4 points: The students show that the greedy strategy is to always open the next sprinkler as late as possible in their algorithm.
- 2 points: The student's algorithm is fully correct.

Part b.

- 2 points: The running time of the algorithm is O(n).

Part c.

- 2 points: Any proof of Induction
- 1 points: Induction on $D_{Ours}(m)$, the maximum total distance that can be covered by m sprinklers selected by our algorithm. Let $D_{OPT}(m)$ denote the maximum total distance that can be covered by m sprinklers in the optimal solution.
- 1 points: Base Case: The first sprinkler will always be opened and $D_{Ours}(1) = D_{OPT}(1) = D$. $D_{Ours}(2) \geq D_{OPT}(2)$ because our algorithm opens the second sprinkler that is as far as possible from the first sprinkler.
- 1 points: Induction Hypothesis: Assume for $m = k$, $D_{Ours}(k) \geq D_{OPT}(k)$.
- 2 points: Induction Step: The sprinkler opened at kth step in optimal solution and our algorithm could cover at most D meters. For $(k + 1)$th stop, our algorithm will select the last sprinkler before $D_{Ours}(k) + D$. The $(k + 1)$th sprinkler in the optimal solution is before $D_{OPT}(k)+D$. Since $D_{Ours}(k)+D \geq D_{OPT}(k)+D$, the $(k+1)$th sprinkler selected by the optimal solution will be no further than the (k + 1)th sprinkler selected by our algorithm. Thus, $D_{Ours}(k + 1) \geq D_{OPT}(k + 1)$. Since our algorithm can always cover at least as much distance as the optimal solution with m sprinklers, then our algorithm is guaranteed to

7. Divide and Conquer (15 points)

Suppose you have a set of n identical coins, one of which is a counterfeit. The coins all look the same, however the counterfeit coin is lighter by an unknown amount. You have a rudimentary weighing scale which you can use to compare the weight of two sets of objects. The scale tells you if its two sides weigh equal or tells you which side is lighter.

a) Design a divide-and-conquer algorithm (either in plain English or in pseudo-code) to determine the counterfeit coin. (10 points)

b) Write down the recurrence relation for the runtime complexity and solve it by the master theorem. (5 points)

Algorithm

- Split the coins up into 3 stacks of equal size $n/3$. If n cannot be divided by 3 evenly, split the n coins up 3-way so that two stacks contain an equal number of coins and the number in the third differs by 1 from the two others.
- Put two stacks with an equal number of coins on the scale.
  a) If one stack weighs less than the other, remove the two other stacks and go to step 3.
  b) If both stacks weigh the same, remove them.
- If the remaining stack has exactly 1 coin in it, it will be the counterfeit one. Stop.
- Otherwise recursively go to step 1.

Time complexity: $T(n) = T(n/3) + O(1)$

$T(n) = \Theta(\log_3 n)$

Rubrics. Part a - 10 points

- Division of array to x a constant number partitions (2 points)
- Handling the cases when n is not divisible by x (2 points)
- Continuing the search on the lighter partition (2 points)
- Mentioning when the fake coin is found ie in the last comparison of two coins or during the division (3 points).
- Their proposed algorithm runs in logarithmic time ie considering comparison takes O(1).

Part b - 5 points

- Correct recurrence equation based on their introduced algorithm - depends on x 2 points $\rightarrow T(n/x) + O(1)$
- Computation of c (1 point)
- Mentioning the case of master theorem (1 point)
- Final time complexity (1 point)

Additional space

Additional space

Additional space