CSOR 4231 Midterm 2

Hongmin Zhu

TOTAL POINTS

25.5 / 60

QUESTION 1

20 pts

1.1 **0 / 10**

- **- 0 pts** Correct
- ✓ **- 10 pts greedy algorithm incorrect**
- **- 3 pts** greedy algorithm incorrect but also gave correct DP algorithm
- **- 4 pts** mostly correct
- **- 10 pts** incorrect

1.2 **3 / 3**

- ✓ **- 0 pts Correct**
- **- 3 pts** incorrect for given algorithm
- **- 3 pts** empty

1.3 **0 / 7**

- **- 0 pts** Correct
- ✓ **- 7 pts incorrect**
- **- 3 pts** needs elaboration
- **- 2 pts** mostly correct

QUESTION 2

2 **10 / 20**

- **- 10 pts** Incorrect Formula
- **- 5 pts** Incorrect or missing Optimal Substructure Proof
- **- 5 pts** Incorrect or missing Running Time analysis
- ✓ **- 5 pts Partially correct Formula**
- **- 3 pts** Partially Correct Optimal Substructure Proof
- **- 3 pts** Partially Correct Running Time analysis
- **- 15 pts** Using Sk-1 plus the max of the two values in the next card to get Sk
- **- 15 pts** Incorrect Algorithm
- ✓ **- 5 pts Partially correct analysis**
- **- 0 pts** correct
- **- 3 pts** small mistakes (detailed in comments)
- **- 18 pts** Click here to replace this description.
- **- 5 pts** n^3 is not optimal algorithm here
- **- 10 pts** Incorrect analysis
- **- 0 pts** n2 is correct.
- **- 13 pts** Click here to replace this description.

Where are we starting building up the recursion and the recursion formula is wrong .

QUESTION 3

20 pts

3.1 **5 / 5**

- ✓ **- 0 pts Correct**
- **- 5 pts** Wrong
- **- 2.5 pts** Wrong Justification

3.2 **0 / 5**

- **- 0 pts** Correct
- ✓ **- 5 pts Wrong**
- **- 2.5 pts** Wrong justification

3.3 **2.5 / 5**

- **- 0 pts** Correct
- **- 5 pts** Wrong
- ✓ **- 2.5 pts Wrong justification**

3.4 **5 / 5**

- ✓ **- 0 pts Correct**
- **- 5 pts** Empty
- **- 1 pts** Minor mistake
- **- 5 pts** No (w/o explanation)
- **- 0.5 pts** Analyzed total running time instead of amortized running time.
- **- 1 pts** Wrong/missing running time but correct algorithm

- 5 pts Wrong answer

CSOR 4231 Midterm Exam 2
November 21, 2019, 10:10 AM

Rules; Answer each question completely and concisely. When you give an algorithm, be sure to give the most efficient one you can, explain why it is correct, and to analyze its running time. You do not need to repeat the details or pseudocode for any algorithm used in class, you can use it as a black-box.

Answer in the pages given. You can use the fronts and backs. Please make clear what you expect the graders to read and what is scratch work.

Question	Points	Total
1		20
2		20
3		20
Total		60

UNI: hz2637

Name: Hongmin Zhu

Please read and sign the following:
HONOR PLEDGE: I pledge that I have neither given nor received unauthorized aid during this examination.

Signature: Hongmin Zhu

n k

$p_1 \dots p_n$

k

Problem 1 [20 POINTS] Based on your passion for greedy algorithms, you decide to set up a franchise of stores where people can come and bring their greedy algorithms to be solved. As a start you will locate them along Interstate 80, a road that travels the length of the United States. For this problem, assume that Interstate 80 is n miles long, and you are given, at each mile marker i, a profit p_i associated with building a store at that mile marker. You want to make sure that you don't build too many stores, because then they would compete with each other, so you decide that each one must be at least k miles away from any other stores.

a) **[10 POINTS]** Give an algorithm that determines the set of stores you can build to obtain the maximum profit.

the problem can be described as using intervals that have length of k to cover as many mile marker as possible. So we can use a greedy algorithm to solve it. Suppose the road and its mile markers can be treated as an array, the array has n items, each one is a mile marker i.e. A with n entries, we use a set R to store the location of each store, which is the start point of the interval. we start with the first entry and draw an interval of length k, then we add this interval to our result set R and remove the points within the interval from array A, then we start from the first point of A again, until array A is empty.

```
~~function getStore(A, k)~~
~~    R ← ∅~~
~~    while (A is not empty) do~~
~~        start ← A[0]~~
~~        interval ← ∅~~
~~        size ← A.length-1~~
~~        for i ← 0 to size do~~
~~            if (A[i] >= start && A[i] ≤ start+k) then~~
~~                interval ← A[i]~~
~~        R ← interval~~
```

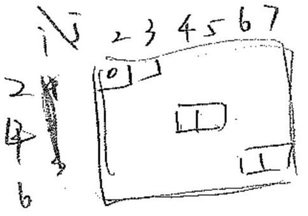
```
function getStore(A, k)
    R ← ∅
    while A is not empty do
        start ← A[0]
        interval ← ∅
        for each item in A do
            if item >= start && item ≤ start+k
                interval ← item
                delete(A, item)
        R ← interval
    return R
```

b) **[3 POINTS]** Analyze the running time of your algorithm

(b) the algorithm will go over each item in array A (each mile marker) for only 1 time, So the running time is O(n)

c) [**7 POINTS**] Prove that your algorithm is correct.

the proposed algorithm tries to cover as many mile markers as possible in each interval, so assume for ~~[illegible]~~ a step, the ~~[illegible]~~ algorithm gives us the optimal Solution of $[x, x+k]$. assume the ~~[illegible]~~ there exists a optimal Solution $[p, p+k]$ for this step, and we know ~~[illegible]~~ that $p \le x \le p+k$ since $[p, p+k]$ is the optimal solution. Recall that after each interval is found, the algorithm will remove the items from array A and starts again from the first item. So we know x should be the first item, and since $p \le x$ then there is nothing inside $[p, x)$, which means ~~[illegible]~~ p equals to x, So we can just replace p with x, we get $[x, x+k]$ for this step and its the optimal solution. So we can say the algorithm gives us optimal Solution and it is correct.



$i \rightarrow$ take ① $\longrightarrow e_j$

$\rightarrow$ take ②

2 3 4 5 6

envelope 2 envelope 4

card 2 ~~[illegible]~~ $\rightarrow (V_2, e_2)$ card 4 $\rightarrow (V_4, e_4)$

$i / i+1$ / i card 3 $\rightarrow (V_3, e_3)$ card 5 $\rightarrow (V_5, e_5)$

$(i, j) \rightarrow x$

cost

Problem 2. [20 POINTS] You are given a sequence of n envelopes, numbered with even numbers from 2 through $2n$. Each envelope i has two cards in it, numbered i and $i + 1$. Each card j has a pair of numbers (v_j, e_j) where v_j a value and e_j is the number of another envelope to take. You are promised that $e_j > j$, for $j < 2n$, and that e_j is even. When you open an envelope i , you choose to keep exactly one of the cards $j \in \{i, i + 1\}$ in the envelope, keep that card, and then you must open the envelope e_j next. When you open envelope $2n$, you stop. (Note that the rules guarantee that you will eventually open envelope $2n$). You begin by opening envelope 2 and your payoff is the sum of the v_j values for the cards you kept.

This game sounds like a game of guessing, but suppose that you can cheat and obtain all the data about every card in advance.

Give a dynamic programming algorithm to optimally play the game. That is, you should give an algorithm that chooses the set of cards with the largest total value. You should prove optimal substructure, give a recurrence and explain what the running time will be and why. You do not need to give pseudocode. Be sure to explain in words what the variables in your recurrence stand for.

0. 0.

~~[illegible]~~ i

$V(2,2) = V_2$

$V(2,3) = V_3$

$V(4,2) = V$

$(i,i) \rightarrow \text{cost} + V_i$

$(i,i+1) \rightarrow \text{cost} + V_{i+1}$

cost 1

First, we will build a look-up table so that we know for choosing card j we know the next envelope we're looking for is e_j, ~~so the table is~~ also ~~T(i,j) =~~ we can know the value for each card. The table $T(i,j) = (V_j, e_j)$ $(j = i$ or $j = i+1)$ ~~T(i,j) =~~

for ~~each~~ the values of ~~at~~ current step, we also have a table cost to store the values. Suppose, we're ~~[illegible]~~ now coming to envelop i, we have two choices to choose card i ~~[illegible]~~ or card $i+1$, but no matter which one we chose, the values we get before envelope i are fixed, which is the optimal values for the subproblem. So for current envelop i we have ~~cost~~

$\text{cost}(i) = \max($ ~~cost(backtrace(i))~~ $V_i, V_{i+1}) + \text{cost}(\text{backtrace}(i))$

we ~~use~~ find the current optimal value, and use ~~return~~ recursion to get the next target envelope and pass the current values as well so after the recursion, we'll have the optimal solution.

Since we build look-up table to find corresponding envelope, the worst case is we check every envelope for once, so the running time is $O(n)$

(For more of the answer to problem 2)

~~for i~~

Problem 3. **[20 POINTS]** You are given an array A with n positive numbers, and the array is initially in an arbitrary order.

You are given a sequence of calls SELECTATION(i), where SELECTATION(i) returns the ith smallest number. As you know, one call to SELECTATION can be implemented in $O(n)$ time, using the linear time SELECT algorithm studied in class.

a) **[5 POINTS]** Suppose that you keep an array $B[1 \dots n]$ with each $B[i] = -1$ initially. When you get a call to SELECTATION(i), you check if $B[i] = -1$. If it does, you run linear time SELECT(i) on array A and store the result in $B[i]$. If $B[i] \neq -1$, then you return $B[i]$. What is the worst case running time of this operation?

the worst case is that for i, we need to go through all items in B to find if $B[i] == -1$ or $B[i] \neq -1$, which takes $O(n)$ time, and then we find $B[i] == -1$, so we have to go through array A to find the smallest which takes another $O(n)$ time, So the total running time is $O(n)$

b) **[5 POINTS]** Suppose that you have a total of n^2 calls to SELECTATION. What is the amortized running time of the algorithm described in part a?

~~for the if $B[i] \neq -1$, we can say the~~

for ~~a~~ real cost, we have c_i, for amortized time we have $\hat{c}_i$

$$\sum_{i=1}^{n^2} c_i \leq \sum_{i=1}^{n^2} \hat{c}_i$$

assume $B[i] == -1$, we assign amortized time $O(~~[illegible]~~\ n)$ to it

if $B[i] \neq -1$, the amortized time is 0.

c) [**5 POINTS**] Suppose that you have a total of n calls to SELECTATION. What is the amortize running time of the algorithm described in part a?

Suppose the real cost is c_i, the amortized time is $\hat{c}_i$

$\sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i$

total = $O(n^2)$

d) [**5 POINTS**] Can you implement SELECTATION in a way that the n calls have a lower amortized running time than they do in your answer to part c? If so, please give the implementation and analyze the running time.

we can use a min-heap to store the value, since ~~for~~ getting the minimum value takes only $O(1)$ time, and sorting the heap is $O(\log n)$ time.