CONCORDIA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

COMP 6651: Algorithm Design Techniques

Fall 2019 - Quiz # 3

Question 1.

The following result can be proved using a reasoning by induction.
Theorem: Given any unreduced schedule S, it can transform it into a reduced schedule S' with no more evictions.
State the base case and the inductive step.
What are the different cases you need to consider in the inductive step.

Base case **2 points**
Induction is on the number of unreduced items.
Base case: prove that the statement holds for the first natural number $j = 1$.

Inductive Step. **2 points**
Induction is on the number of unreduced items. Prove that for every $j \geq 1$, if the statement holds for j, then it holds for $j + 1$. In other words, assume the statement holds for some arbitrary natural number $j \geq 1$, and prove that then the statement holds for $j + 1$.
Statement: Given any unreduced schedule S with j unreduced items, it can transform it into a reduced schedule S' with no more evictions.

Cases to be considered in the inductive step **6 points**
Consider a schedule S with $j + 1$ unreduced items, and focus on the first unreduced item.
Suppose S brings d into the cache at time t, without a request. Let c be the item S evicts when it brings d into the cache.

Case 1: d evicted at time t', before next request for d.

Case 2: d requested at time t' before d is evicted.

Question 2. Row of seats. Walmart Labs Interview Question for Senior Software Development Engineers

Consider N seats adjacent to each other. There is a group of people who are already seated in that row randomly, i.e., some are sitting together & some are scattered.
An occupied seat is marked with a character 'x' and an unoccupied seat is marked with a dot ('.')
Your target is to make the whole group sit together, i.e., next to each other, without having any vacant seat between them in such a way that the total number of jumps to move them is minimum.

Example
15 seats represented by the String (0, 1, 2, 3, ... , 14):
Input: . . . . x . . x x . . . x . .
Output: . . . . . x x x x . . . . .
$\rightsquigarrow$ 5 jumps

One of your friends suggest the following idea:
Make an array and store (index of 1st x $\rightsquigarrow$ ARRAY[0]) , (index of 2nd x $\rightsquigarrow$ ARRAY[1]) , (index of kth x $\rightsquigarrow$ ARRAY$[k-1]$).
Take the median of this array and then take sum of |median − ith element of ARRAY | where i starts from 0 to the size of array.

Do you think it is a good idea?
If yes, write an algorithm, apply it to the above example, and provide its complexity.
If no, provide your own algorithm, apply it to the above example and analyze its complexity.

Bonus Question (5 points): can you prove that the greedy algorithm, whether it is the one of your friend or yours is exact?

Algorithm **7 points**

number_x ← 0
For $i = 0$ **to** $N - 1$ **do** **3 points. Idea of** ARRAY
  If (A[i] = 'x') **then**
    ARRAY[number_x] ← i
    number_x ← number_x + 1
  EndIf
EndFor
MID ← $\lfloor (N+1)/2 \rfloor - 1$
#JUMPS ← 0
For $i = 0$ **to** number_x −1 **do** **4 points. Second pass**
  START ← ARRAY[i]
  DEST ← ARRAY[MID] − (MID − i)
  #JUMPS ← #JUMPS + |DEST − START|
EndFor
return #JUMPS

ARRAY			
0	1	2	3
4	7	8	12

number_x = 4
MID = 1

#JUMPS = 0

i = 0
START = 4
DEST = 7 - (1-0) = 6
#JUMPS = 6 - 4 = 2

i = 1
START = 7
DEST = 7 - (1-1) = 7
#JUMPS = 2

i = 2
START = 8
DEST = 7 - (1-2) = 8
#JUMPS = 2

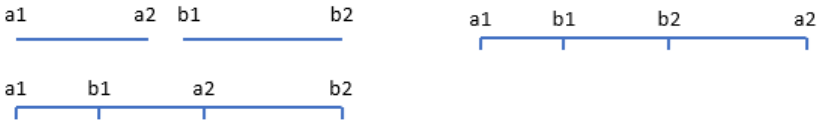
i = 3
START = 12
DEST = 7 - (1-3) = 9
#JUMPS = 2 + 3 = 5

Complexity: $O(N)$ **3 points**

Optimality Proof **5 points**

Lemma 1. Given 4 numbers a_1, a_2, b_1, b_2, and $a_1 < a_2, b_1 < b_2$, then $|a_1 - b_1| + |a_2 - b_2| \leq |a_1 - b_2| + |a_2 - b_1|$.

This Lemma can be proved by listing out all possible combinations and calculating the geographical distances, as in the following illustration.



From Lemma 1, we can deduce the following result. Suppose we have 2 people currently in position a_1, a_2, and with final positions b_1, b_2 respectively. Assuming $a_1 < a_2$, we then have $b_1 < b_2$. It means that people sit in their original order, just in different seats Assume now that we have 5 people, and the original positions are: ...1..2.3...4..5, they can start with 12345....., and move the window until finding the minimum result.

We need to distinguish 2 cases. Let n be the number of people, then n can be odd or even. Call $a_1, ...a_n$ the original positions of the people, and $b_1, ...b_n$ be their final positions.

Case 1: n is odd.

Call m the median position. So to the left and right of m there are k people, with $k = (n-1)/2$. We start with the solution that satisfies $a_m = b_m$, meaning that we keep the position of m, and move the others closer to m.

If we move the window $b_1, ...b_n$ to the left, then all people from the right of m will have #JUMPS increased by 1, so #TOTAL_#JUMPS increases by k. For all the people from the left, their distance either decrease or increase by 1, depending on their original position a_i and current position b_i. If $a_i < b_i$ then by moving to the left, we decrease the distance, if $a_i > b_i$, we increase the distance, so #TOTAL_#JUMPS decreases by $k' \leq k$. So in total, #NEW_TOTAL_#JUMPS $\geq$ #TOTAL_#JUMPS $+k - k'$, the total distance increases.

The same reason when we move the window $b_1, ...b_n$ to the right.

Case 2: n is even.

Let m be the lower median, then $m+1$ is the upper median. It follows that there are k people to the left of m, k people to the right of $m+1$. If we move the window linearly between 2 states that satisfy: $a_m = b_m$, or $a_{m+1} = b_{m+1}$, then the total distances of $2k$ persons from the left and right of $m, m+1$ do not change, because if one side increase total distance by k, the other side decrease total distance by k. Let consider the distance of $m, m+1$. By moving the window between these 2 states, by moving left, distance of m will decrease by 1 while distance of $m+1$ will increase by 1, same result if moving right. Conclusion, by moving between these 2 states, the total number of distances does not change.

If we move the window outside the 2 previous states, the total distance will increase, using the same reasoning as in **Case 1**.

So from the above reasoning, the state in which $a_m = b_m$ defines an optimal solution.