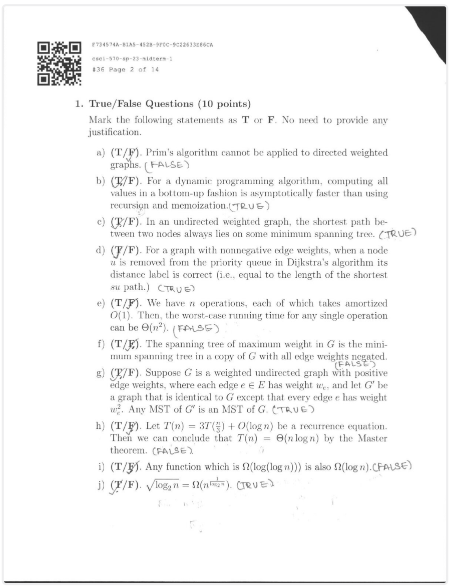
My grades for csci-570-sp-23-midterm-1

Q1 6


F734574A-B1A5-452B-9F0C-9C22633E86CA

csci-570-sp-23-midterm-1

#36 Page 2 of 14

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. (FALSE)

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

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

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)

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)

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. (FALSE)

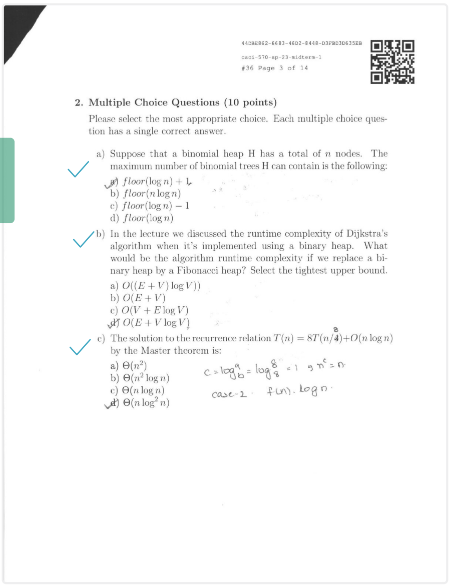
g) ([✓ **T**]/**F**). Suppose G is a weighted undirected graph with positive edge weights, where each edge $e \in E$ has weight w_e, and let G' be a graph that is identical to G except that every edge e has weight w_e^2. Any MST of G' is an MST of G. (TRUE)

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)

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

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

Q2 10



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) $floor(\log n)$

✓ 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.

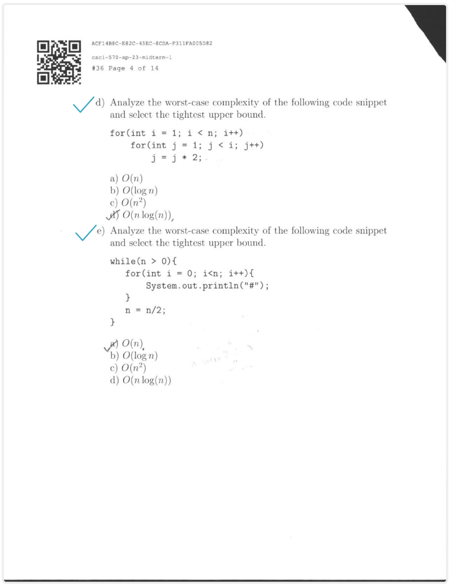
a) $O((E + V) \log V))$
b) $O(E + V)$
c) $O(V + E \log V)$
[✓ d) $O(E + V \log V)$]

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

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

$c = \log_b^a = \log_8^8 = 1 \Rightarrow n^c = n.$

case-2. $f(n).\log n.$



ACF14B8C-E82C-45EC-8CDA-F311FA005D82

csci-570-sp-23-midterm-1

#36 Page 4 of 14

✓ 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;
```

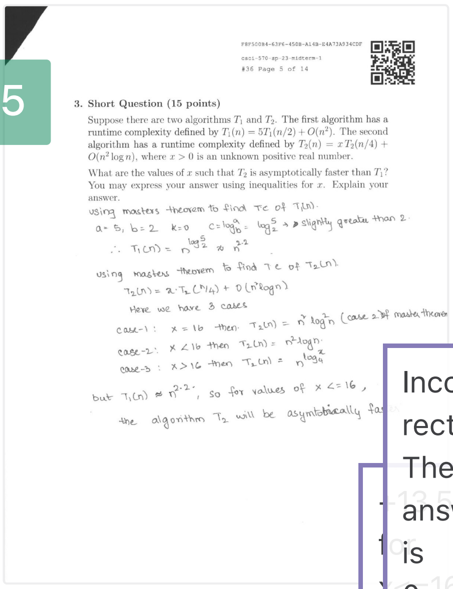
a) $O(n)$
b) $O(\log n)$
c) $O(n^2)$
[✓ d) $O(n \log(n))$]

✓ 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) $O(n \log(n))$

Q3 13.5



F8F500B4-63F6-450B-A14B-E4A73A934CDF

csci-570-sp-23-midterm-1

#36 Page 5 of 14

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.

using masters theorem to find Tc of $T_1(n)$.

$a = 5, b = 2 \quad k = 0 \quad c = \log_b^a = \log_2^5 \rightarrow$ slightly greater than 2.

$\therefore T_1(n) = n^{\log_2^5} \approx n^{2.2}$

using masters theorem to find T c of $T_2(n)$.

$T_2(n) = x \cdot T_2(n/4) + O(n^2 \log n)$

Here we have 3 cases

case-1 : $x = 16$ then $T_2(n) = n^2 \log^2 n$ (case 2) of master theorem

case-2 : $x < 16$ then $T_2(n) = n^2 \log n$.

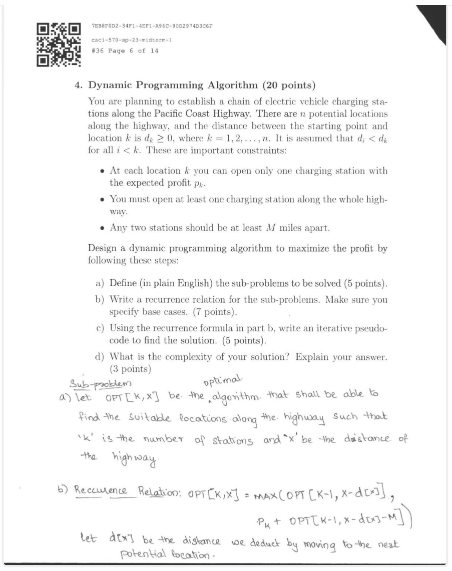
case-3 : $x > 16$ then $T_2(n) = n^{\log_4^x}$

but $T_1(n) \approx n^{2.2}$, so for values of $x <= 16$,

the algorithm T_2 will be asym~~to~~tically fas

Incorrect, The answer is 0<x<25 Refer posted solution.

Q4 6



7EB8F0D2-34F1-4EF1-A96C-90D2974D3C6F
csci-570-sp-23-midterm-1
#36 Page 6 of 14

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.

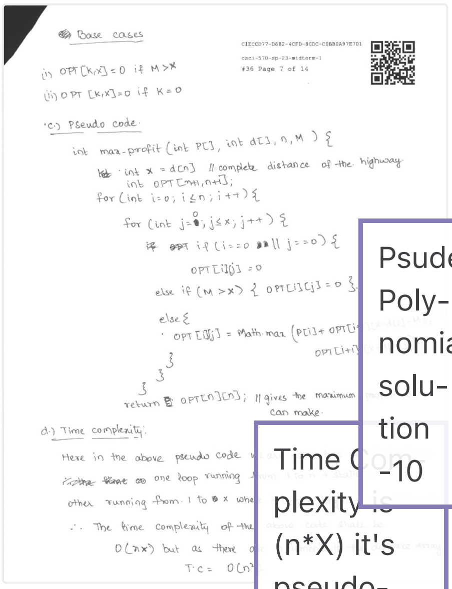
Design a dynamic programming algorithm to maximize the profit by following these steps:

a) Define (in plain English) the sub-problems to be solved (5 points).
b) Write a recurrence relation for the sub-problems. Make sure you specify base cases. (7 points).
c) Using the recurrence formula in part b, write an iterative pseudo-code to find the solution. (5 points).
d) What is the complexity of your solution? Explain your answer. (3 points)

Sub-problem
a) let OPT[K,x] be the optimal algorithm that shall be able to find the suitable locations along the highway such that 'k' is the number of stations and 'x' be the distance of the highway.

b) Reccurence Relation: $OPT[K,X] = MAX(OPT[K-1, X-d[x]], P_K + OPT[K-1, X-d[x]-M])$

let d[x] be the distance we deduct by moving to the neat potential location.



C1ECCD77-D682-4CFD-8CDC-C0BB0A97E701
csci-570-sp-23-midterm-1
#36 Page 7 of 14

Base cases

(i) $OPT[K,X] = 0$ if $M > X$
(ii) $OPT[K,X] = 0$ if $K = 0$

c.) Pseudo code.

```
int max-profit (int P[], int d[], n, M ) {
    ~~let~~ int x = d[n]  // complete distance of the highway
        int OPT[n+1, n+1];
    for (int i=0; i≤n; i++){
        for (int j=0; j≤x; j++) {
            ~~if~~ ~~OPT~~ if (i==0 || j==0) {
                OPT[i][j] = 0
            else if (M > x) { OPT[i][j] = 0 }.
            else {
                OPT[i][j] = Math.max (P[i]+ OPT[i
                                       OPT[i+1]
            }
        }
    }
    return OPT[n][n]; // gives the maximum
                         can make.
```

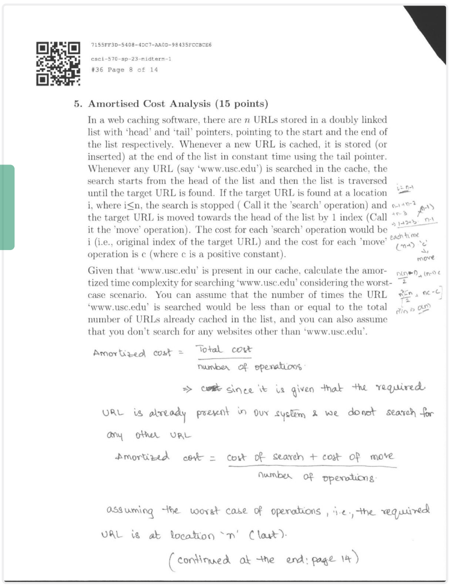
d.) Time complexity:

Here in the above pseudo code
~~∴ the~~ ~~there~~ one loop running
other running from 1 to x whe
∴ The time complexity of the
O(nx) but as there
T.c = O(n

Psudeo Poly-nomial solu-tion -10

Time Com-plexity is (n*X) it's pseudo-polynomial not poly-nomial -4

Q5 15



7155FF3D-5408-4DC7-AA0D-98435FCCBCE6
csci-570-sp-23-midterm-1
#36 Page 8 of 14

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'.

$i = n-1$

$n-1+n-2+n-3$ $(n-1)$

$\Rightarrow 1+2+3 \dots n-1$

each time $(n-1)$ 'c' move

$\frac{n(n-1)}{2} + (n-1)c$

$\frac{n^2-n}{2} + nc - c$

$n^2/n \Rightarrow O(n)$

Amortized cost = $\frac{\text{Total cost}}{\text{number of operations}}$

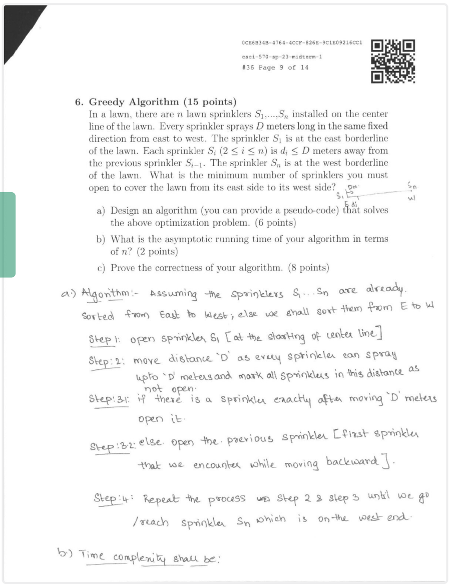
$\Rightarrow$ ~~cost~~ since it is given that the required URL is already present in our system & we donot search for any other URL

Amortized cost = $\frac{\text{cost of search + cost of move}}{\text{number of operations}}$

assuming the worst case of operations, i.e., the required URL is at location 'n' (last).

(continued at the end: page 14)

Q6 10



0CE6B34B-4764-4CCF-826E-9C1E09216CC1
csci-570-sp-23-midterm-1
#36 Page 9 of 14

6. Greedy Algorithm (15 points)
In a lawn, there are n lawn sprinklers $S_1,\ldots,S_n$ 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. (8 points)

a.) Algorithm:- Assuming the sprinklers $S_1 \ldots S_n$ are already. sorted from East to West; else we shall sort them from E to W

Step 1: open sprinkler S_1 [at the starting of center line]

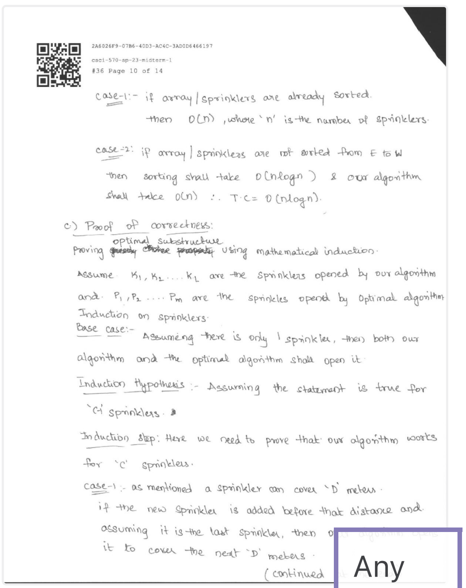
Step: 2: move distance `D' as every sprinkler can spray upto `D' meters and mark all sprinklers in this distance as not open.

Step: 3.1: if there is a sprinkler exactly after moving `D' meters open it.

Step: 3.2: else open the previous sprinkler [first sprinkler that we encounter while moving backward].

Step: 4: Repeat the process ~~[illegible]~~ Step 2 & step 3 until we go /reach sprinkler S_n which is on the west end.

b.) Time complexity shall be:



2A6026F9-07B6-40D3-AC4C-3AD0D6466197
csci-570-sp-23-midterm-1
#36 Page 10 of 14

case-1:- if array / sprinklers are already sorted.
then $O(n)$, where 'n' is the number of sprinklers.

case-2: if array / sprinklers are not sorted from E to W
then sorting shall take $O(n\log n)$ & our algorithm shall take $O(n)$ $\therefore$ T.C = $O(n\log n)$.

c) Proof of correctness:

Proving ~~greedy choice property~~ optimal substructure using mathematical induction.

Assume $K_1, K_2 \dots K_L$ are the sprinklers opened by our algorithm and $P_1, P_2 \dots P_m$ are the sprinkles opened by Optimal algorithm
Induction on sprinklers.

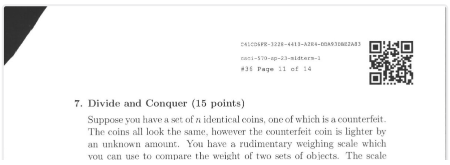
Base case:- Assuming there is only 1 sprinkler, then both our algorithm and the optimal algorithm shall open it.

Induction Hypothesis :- Assuming the statement is true for 'C-1' sprinklers.

Induction step: Here we need to prove that our algorithm works for 'C' sprinklers.

case-1 :- as mentioned a sprinkler can cover 'D' meters.
if the new sprinkler is added before that distance and assuming it is the last sprinkler, then o
it to cover the next 'D' meters.
(continued

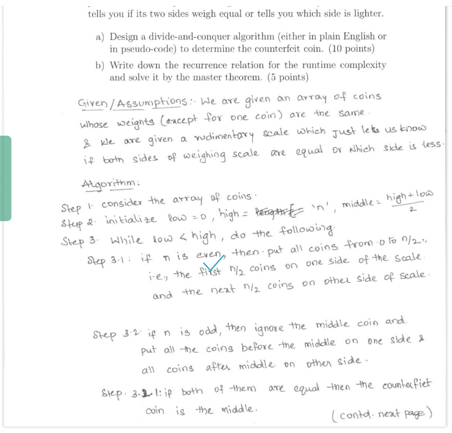
Any Induction Step +2



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

Q7 15



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)

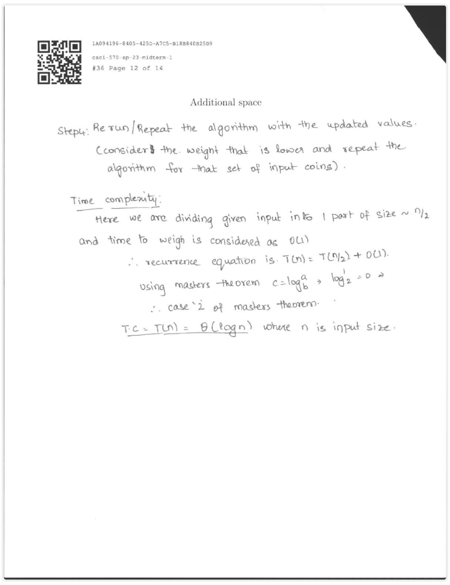
Given / Assumptions :- We are given an array of coins whose weights (except for one coin) are the same. & We are given a rudimentary scale which just lets us know if both sides of weighing scale are equal or which side is less.

Algorithm:

```
Step 1. consider the array of coins.
Step 2. initialize low = 0, high = ~~length~~ 'n', middle = $\frac{high+low}{2}$
Step 3. While low < high, do the following.
    Step 3.1 : if n is even, then put all coins from 0 to n/2.
               i.e., the first n/2 coins on one side of the scale.
               and the next n/2 coins on other side of scale.

    Step 3.2 if n is odd, then ignore the middle coin and
               put all the coins before the middle on one side &
               all coins after middle on other side.
        Step. 3.2.1: if both of them are equal then the counterfiet
               coin is the middle.
```

(contd. next page)

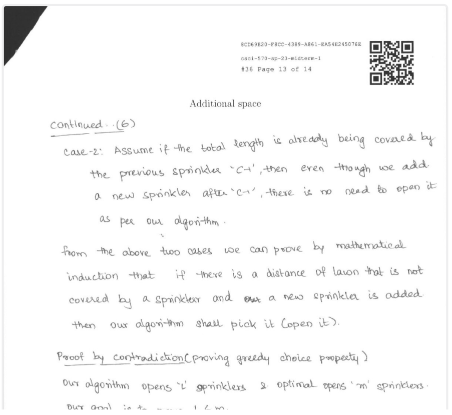


1A094196-8405-425D-A7C5-B18B840B25D9
csci-570-sp-23-midterm-1
#36 Page 12 of 14

Additional space

Step4: Re run/Repeat the algorithm with the updated values.
(considers the weight that is lower and repeat the algorithm for that set of input coins).

Time complexity:
Here we are dividing given input into 1 part of size ~ $n/2$ and time to weigh is considered as $O(1)$
$\therefore$ recurrence equation is $T(n) = T(n/2) + O(1)$.
using masters theorem $c = \log_b^a \Rightarrow \log_2^1 = 0 \Rightarrow$
$\therefore$ case 2 of masters theorem.
T.C = $T(n) = \theta(\log n)$ where n is input size.



8CD69E20-F8CC-4389-A861-EA54E245076E

csci-570-sp-23-midterm-1

#36 Page 13 of 14

Additional space

Continued : (6)

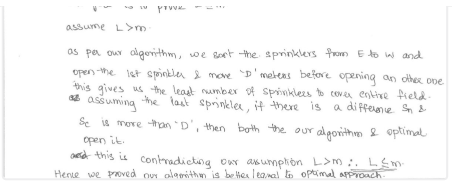
case-2: Assume if the total length is already being covered by the previous sprinkler 'C-1', then even though we add a new sprinkler after 'C-1', there is no need to open it as per our algorithm.

from the above two cases we can prove by mathematical induction that if there is a distance of lawn that is not covered by a sprinkler and ~~out~~ a new sprinkler is added then our algorithm shall pick it (open it).

Proof by contradiction (proving greedy choice property)

Our algorithm opens 'l' sprinklers & optimal opens 'm' sprinklers.

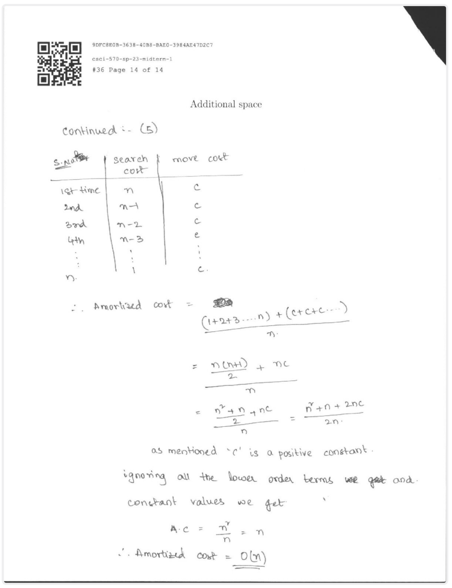
our goal is to prove $l \le m$.



is to prove

assume $L > m$.

as per our algorithm, we sort the sprinklers from E to W and open the 1st sprinkler & move 'D' meters before opening an other one. this gives us the least number of sprinklers to cover entire field.
~~as~~ assuming the last sprinkler, if there is a difference. S_n & S_c is more than 'D', then both the our algorithm & optimal open it.
~~and~~ this is contradicting our assumption $L > m$ $\therefore$ $L \le m$.
Hence we proved our algorithm is better/equal to optimal approach.



9DFC8E0B-3638-40B8-BAE0-3984AE47D2C7

csci-570-sp-23-midterm-1

#36 Page 14 of 14

Additional space

Continued :- (5)

S.No.	search cost	move cost
1st time	n	c
2nd	$n-1$	c
3rd	$n-2$	c
4th	$n-3$	c
⋮	⋮	⋮
n.	1	c.

$\therefore$ Amortized cost $= \dfrac{(1+2+3\ldots n) + (c+c+c\ldots)}{n}$

$$= \frac{\frac{n(n+1)}{2} + nc}{n}$$

$$= \frac{\frac{n^2+n}{2} + nc}{n} = \frac{n^2+n+2nc}{2n}$$

as mentioned 'c' is a positive constant.

ignoring all the lower order terms ~~we get~~ and constant values we get

$$A.c = \frac{n^2}{n} = n$$

$\therefore$ Amortized cost $= \underline{O(n)}$