German University in Cairo
Faculty of Media Engineering and Technology
Department of Computer Science and Engineering
Dr. Nourhan Ehab

Barcode

CSEN 703 Analysis and Design of Algorithms
Winter Semester 2021
Midterm Exam
November 15^{th}, 2021

Instructions: Read carefully before proceeding.

1. The allowed time for this exam is **2 hours** (120 minutes).
2. Non-programmable calculators are allowed.
3. No books or other aids are permitted for this test.
4. This exam booklet contains **14 pages**, including this one. Three extra sheets of scratch paper and a formula sheet are attached and have to be kept attached. Note that if one or more pages are missing, you will lose their points. Thus, you must check that your exam booklet is complete.
5. Please write your solutions in the space provided. If you need more space, please use the back of the sheet containing the problem or on the three extra sheets and make an arrow indicating that.
6. When you are told that time is up, please stop working on the test.

Good Luck! :)

Exercise	1	2	3	4	5	Total
Grade						
Max Grade	14	10	20	10	21	75

Question 1: Modified Quick Sort (14 points = 2 + 2 + 2 + 4 + 4)

Recall the following QuickSort algorithm discussed in class.

```
1  QuickSort(A, p, r)
2  if p < r then
3  |  q = Partition(A, p, r);
4  |  QuickSort(A, p, q − 1) ;
5  |  QuickSort(A, q + 1, r) ;
6  end
```

Suppose that the Partition function at line 2 was replaced by the following function Modified_Partition.

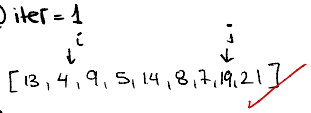
```
1  Modified_Partition(A, p, r)
2  x = A[p];
3  i = p;
4  j = r;
5  while TRUE do
6  |  while j > p and A[j] ≥ x do
7  |  |  j = j − 1;
8  |  end
9  |  while i < r and A[i] ≤ x do
10 |  |  i = i + 1;
11 |  end
12 |  if i < j then
13 |  |  Exchange A[i] with A[j];
14 |  else
15 |  |  Exhange A[p] with A[j];
16 |  |  return j;
17 |  end
18 end
```

(a) Demonstrate the operation of Modified_Partition when called with $A = [13, 19, 9, 5, 14, 8, 7, 4, 21]$, $p = 1$, and $r = 9$. Show the values of the array after each iteration of the while loop in lines 5-18 and the final return value.

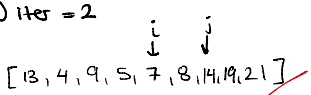
$x = A[1] = 13$

$i = 1 \quad j = 9$

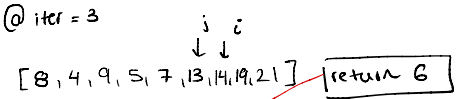
@ iter = 1



@ iter = 2



@ iter = 3



(b) Explain the functionality of Modified_Partition.

sets the pivot to be the first element and as long as the $A[j] \geq$ pivot j++ until j points at a value less than pivot & same for i But for $A[i] \leq x$ The first two loops terminate with $A[j] < x$ & $A[i] > x$ they're swapped to keep any value in $A[1..i] \leq$ pivot & $A[j..r] \geq$ pivot

malosh lazma I guess

finally when i passes j the pivot is exchanged with $A[j]$ so that ⇒ everything to its left is less than it & right greater than it

(c) Is the modified QuickSort algorithm correct when it uses Modified_Partition? Explain your reasoning.

mesh lazem correctness pivot ir

* The sorting is done in the Modified_Partition so proving the correctness of the modified QuickSort is proved by proving Modified_Partition

Yes. One each call the pivot is put in it's correct position maintaining that at the end the entire array will be sorted

* Loop invariant
  1) for the subarray $A[1..i-1]$ any value $A[K] \leq$ pivot where $1 \leq K \leq i-1$, pivot $= x$
  2) for the subarray $A[j+1..r]$ any value $A[K] \geq$ pivot for $j+1 \leq K \leq r$
  3) for $K =$ pivot $A[K] = x$

Intialization

initially $i = p = 1$

1) $A[1..i-1] = A[1..0]$ ⇒ empty array so cond 1 trivially true
2) $A[r+1..r]$ ⇒ empty array so trivially true
3) $x = A[p]$ satisfies third condition

Maintaince

* the first loop keeps decrementing j as long as $A[j] \geq x$ ensuring that $A[j+1 \ldots r]$ has only elements greater than the pivot so satisfies condition 2
* the second loop keeps incrementing i as long as the values seen so far are less than the pivot so ensuring cond 1 is true as elements in $A[1..i-1]$ all $\leq$ pivot

* When the two loops terminate $A[j] < x$ and $A[i] > x$

We have two cases: 1) $i < j$ then $A[j]$ & $A[i]$ swap ensuring conditions 1 & 2 hold & leaving the other element unchanged

2) $i \geq j$ then the pivot is swapped with $A[j]$ ensuring that cond1 is satisfied as we know that $A[j] < x$ so $A[i] < x$ which is in subarray $A[l \dots i-1]$ and the loop terminates with all conditions satisfied

Termination

$i > j$ with $A[i] > x$ & $A[j] =$ pivot

So the loop terminates with all conditions satisfied and the array partitioned into 3 sets: one less than or equal $x \Rightarrow A[l \dots i-1]$,

one greater than or equal $x \Rightarrow A[j+1 \dots r]$

& singleton set with element $= x$

So The modified Quicksort is correct

(d) What is the best and worst cases of the modified QuickSort?

Best case is that Array is not sorted such that the two subproblems would be balanced

Worst case is that the array is sorted so the problem would be divided into one subproblem with size n-1 each time

(e) Write the recurrences representing the best and worst case running times of the modified QuickSort (you don't need to solve them).

Best case

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 2T(n/2) + \Theta(n) & n > 1 \end{cases}$$

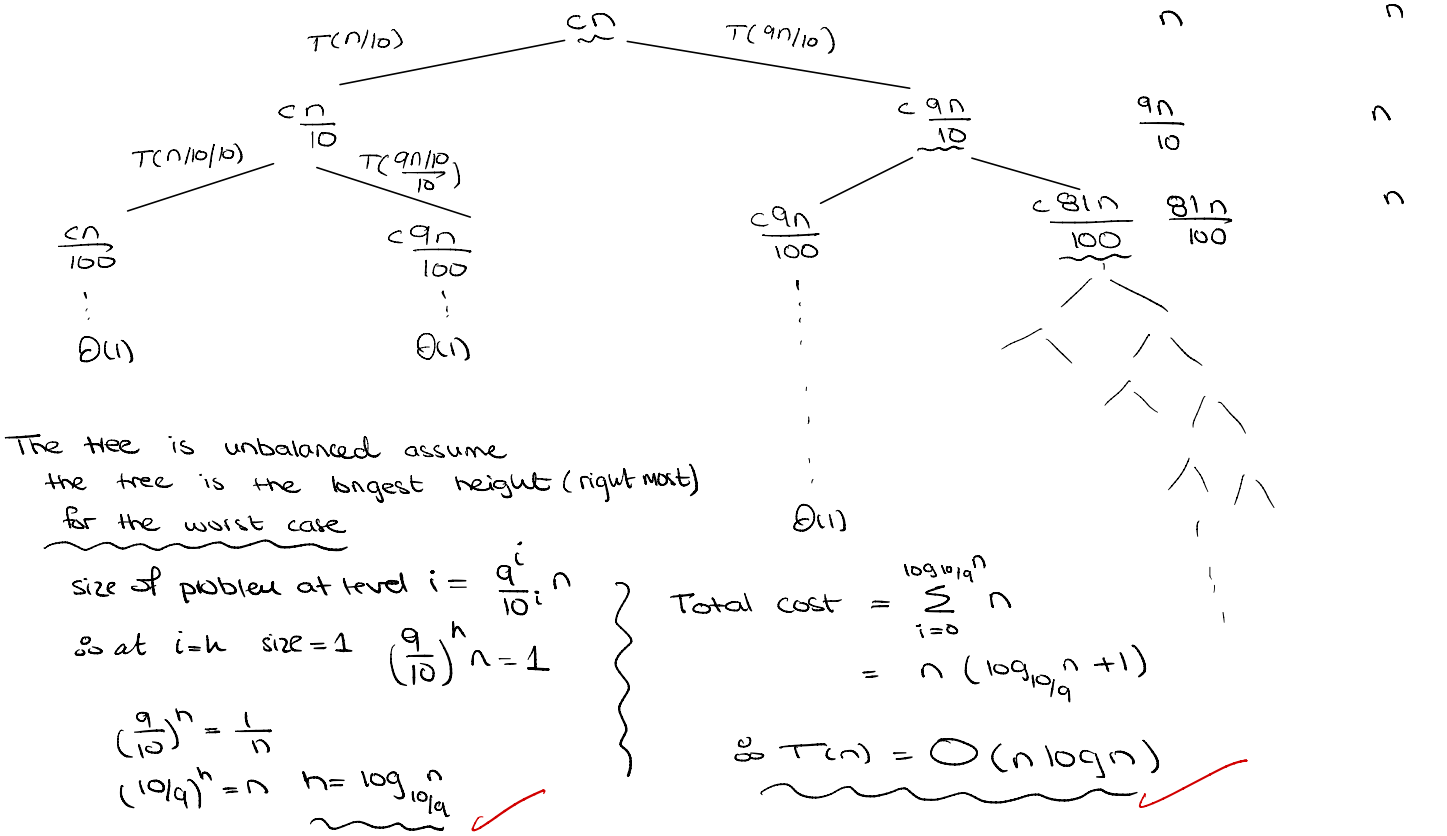
Worst case

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ T(n-1) + \Theta(n) & n > 1 \end{cases}$$

Question 2: Recursion Trees in Rev Lec (10 points = 8 + 2)

(a) Get an **upper bound** on the running time of the following recurrence by using the recursion tree method. Draw the recursion tree and show all of your workout.

$$T(n) = T(\frac{n}{10}) + T(\frac{9n}{10}) + \Theta(n)$$



The tree is unbalanced assume the tree is the longest height (right most) for the worst case

size of problem at level $i = \frac{9^i}{10^i} n$

so at $i = h$ size $= 1$ $\left(\frac{9}{10}\right)^h n = 1$

$\left(\frac{9}{10}\right)^h = \frac{1}{n}$

$(10/9)^h = n$ $h = \log_{10/9} n$

Total cost $= \sum_{i=0}^{\log_{10/9} n} n$

$= n(\log_{10/9} n + 1)$

$\therefore T(n) = O(n \log n)$

(b) Can the recurrence in part (a) be solved using the master method? Justify your answer.

No as it's not in the form

$T(n) = aT(n/b) + D(n) + C(n)$

$a \geq 1$ & $b > 1$

Question 3: Master Theorem (20 points = 4 + 4 + 4 + 4 + 4)

Can the following recurrences be solved using the master method? If yes, solve them. If not, explain why. Show all of your workout.

(a) $T(n) = 4T(\frac{n}{2}) + n^3$

$f(n) = n^3$ $\quad COL = n^{\log_2 4} = n^2$

$n^3 = \Omega(n^2)$ looks like case 3

Prove $n^3 = \Omega(n^{2+\varepsilon})$ $\quad \varepsilon > 0$

$n^3 \geq n^2 \cdot n^\varepsilon$

$n \geq n^\varepsilon$ $\quad \varepsilon = 1$ Holds ✓

Prove that $a f(\frac{n}{b}) \leq c f(n)$ $\quad 0 < c < 1$

$4(\frac{n}{2})^3 \leq c n^3$

$\frac{4}{8} \leq c \Rightarrow \frac{1}{2} \leq c$

let $c = 1/2$ Holds ✓

so $T(n) = \Theta(n^3)$

(b) $T(n) = 7T(\frac{n}{2}) + n^2$

$f(n) = n^2$ $\quad COL = n^{\log_2 7} = n^{2.8}$

$n^2 = O(n^{2.8})$ looks like case 1

Prove $n^2 = O(n^{2.8-\varepsilon})$ $\quad \varepsilon > 0$

$n^2 \leq n^{2.8-\varepsilon}$ $\quad \varepsilon = 0.1$ Holds ✓

so $T(n) = \Theta(n^{\log_2 7})$

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

$f(n) = 1 = c$ $\quad COL = n^{\log_2 1} = n^0 = c$

case 2

so $T(n) = \Theta(\log n)$ ✓

(d) $T(n) = 3T(\frac{n}{2}) + \frac{n^{log_2(3)}}{log_2(n)}$

$f(n) = \frac{n^{\log_2 3}}{\log_2 n}$ $\quad COL = n^{\log_2 3}$

$\frac{n^{\log_2 3}}{\log_2 n} = O(n^{\log_2 3})$ looks like case 1

Prove $\frac{n^{\log_2 3}}{\log_2 n} = O(n^{\log_2 3 - \varepsilon})$ for $\varepsilon > 0$

$\frac{n^{\log_2 3}}{\log_2 n} \leq \frac{n^{\log_2 3}}{n^\varepsilon}$

$\log_2 n \geq n^\varepsilon \Rightarrow$ impossible for $\varepsilon > 0$ as polynomials grow faster than logarithmics

so falls in gap between case 1 & 2 can't solve with masters theorem ✓

(e) $T(n) = 2T(\frac{n}{2}) - \frac{1}{n}$

$f(n) = \frac{1}{n}$ $\quad COL = n^{\log_2 2} = n$

$\frac{1}{n} = O(n)$ looks like case 1

Prove that $\frac{1}{n} = O(n^{1-\varepsilon})$ $\quad \varepsilon > 0$

$\frac{1}{n} \leq n^{1-\varepsilon}$

$n^{-1} \leq n^{1-\varepsilon}$ $\quad \varepsilon = 0.1$ Holds ✓

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

✗ can't be solved as $-\frac{1}{n}$ is not +ve

checked B Java

Question 4: Divide and Conquer Algorithms (10 points)

Write an $O(log_2(n))$ time divide and conquer algorithm in **Pseudo Code** to find the smallest number of an array A in which the input array A first strictly decreases then strictly increases. For example, if $A = [6, 4, 2, 4, 7, 9]$, it first strictly decreases from 6 to 4 to 2, then it strictly increases from 2 to 4 to 7 to 9. Your algorithm is supposed to return the unique smallest element 2. No credit will be given to any algorithm that runs in more than $O(log_2(n))$.

↳ also w base > 2

[5,3,2,1,4,5] mid

[1,4,5]

Base case → [1] ⇐ n = 1 ⇒ return

[5,1,2,3,4,5] mid

[5,1] ⇒ Base case

A[mid+1]

```
Smallest ( A, start, end)
    if (start = end)   // n=1
        return A[start];
    if (end - start = 1)   // n = 2
        if A[end] < A[start]
            return A[end]
        else
            return A[start]
    mid = (start+end)/2
    if ( A[mid] ≤ A[mid+1] and A[mid] ≤ A[mid-1])
        return A[mid]
    else
        if (A[mid+1] ≤ A[mid])
            Smallest (A, mid+1, end)
        else
            smallest (A, start, mid-1)
```

OR

```
def bina(arr, low, high):
    if low==high:
        return arr[low]
    mid= (low+high)//2
    if arr[mid]<arr[mid+1]:
        return bina(arr, low,mid)
    else:
        return bina(arr, mid+1,high)
```

[5,1,2,3,4,5] → [5,1,2] → [5,1] → [1] ⇒ 1

[5,3,2,1,4,5] → [1,4,5] → [1,4] → [1]

Question 5: Greedy Algorithms (21 points = 6 + 3 + 6 + 6)

(a) There are N squirrels and N pits placed on a horizontal number line. Each pit can accommodate only 1 squirrel. A squirrel can stay at its position, move one step right from x to $x + 1$, or move one step left from x to $x - 1$. Any of these moves consumes 1 minute. Given an array S of squirrels' positions, and an array H of pits' positions, the problem is to figure out an assignment for the squirrels to the pits so that the total time it takes them all to move into their assigned pits is minimized, and return this total time.

For example, if $S = [4, -4, 2]$ and $H = [4, 0, 5]$, the output should be 4. One possible assignment is for the squirrel at position 4 will be assigned at the pit at position 5 taking 1 minute, the squirrel at position -4 will be assigned to the pit at position 0 taking 4 minutes, and the squirrel at position 2 will be assigned to the pit at position 4 taking 2 minutes. Thus, the minimum overall needed time is 4 minutes (this is not the only possible assignment to get an overall of 4 minutes).

i. Write a greedy algorithm for the above problem to return the minimum required time. You can describe your algorithm in English.

ii. Argue why your algorithm is correct.

(b) There are a number of policemen and thieves placed on a horizontal number line. You are given an array of size n where each element in the array contains either a policeman or a thief, and an integer K where a policeman can not catch a thief who is more than K units away (in either directions). Knowing that each policeman can only catch one thief, you need to find the maximum number of thieves that can be caught. After studying CSEN 703, your friend suggests that you should use a greedy algorithm to solve the problem. They suggest to use one of the two following greedy properties.

1. For each policeman from the left catch the nearest possible thief.
2. For each policeman from the left catch the farthest possible thief.

i. Argue why using the above two properties might not result in finding the optimal solution.

ii. Suggest a correct greedy choice property for the same problem. Explain your reasoning.

Scratch Paper

Scratch Paper

Scratch Paper

Useful Formulas

Summations:

- Constant series: $\sum_{i=j}^{k} a = a(k - j + 1)$
- Arithmetic series: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- Finite Geometric series: $\sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r}$

Logarithms:

- $ln(n) = log_e(n)$
- $log^k(n) = (log(n))^k$
- $log_c(ab) = log_c(a) + log_c(b)$
- $log_c(\frac{a}{b}) = log_c(a) - log_c(b)$
- $log_b(a^n) = nlog_b(a)$
- $a^{log_b(c)} = c^{log_b(a)}$
- Logarithmic change of base: $log_b(a) = \frac{log_c(a)}{log_c(b)}$
- $\frac{d}{dn} ln(n) = \frac{1}{n}$.
- $\frac{d}{dn} log_b(n) = \frac{1}{n\ ln(b)}$.