Gebze Technical University
Department of Computer Engineering
CSE 321 Introduction to Algorithm Design
Fall 2020
Midterm Exam (Take-Home)
November 25th 2020-November 29th 2020

Student ID and Name	Q1 (20)	Q2 (20)	Q3 (20)	Q4 (20)	Q5 (20)	Total
Yusuf Can Kan 161044007						

Read the instructions below carefully

- You need to submit your exam paper to Moodle by November 29th, 2020 at 23:55 pm as a single PDF file.

- You can submit your paper in any form you like. You may opt to use separate papers for your solutions. If this is the case, then you need to merge the exam paper I submitted and your solutions to a single PDF file such that the exam paper I have given appears first. Your Python codes should be in a separate file. Submit everything as a single zip file.

Q1. List the following functions according to their order of growth from the lowest to the highest. Prove the accuracy of your ordering. **(20 points)**

Note: Your analysis must be rigorous and precise. Merely stating the ordering without providing any mathematical analysis will not be graded!

a) 5^n

b) $\sqrt[4]{n}$

c) $\ln^3(n)$

d) $(n^2)!$

e) $(n!)^n$

Q2. Consider an array consisting of integers from 0 to n; however, one integer is absent. Binary representation is used for the array elements; that is, one operation is insufficient to access a particular integer and merely a particular bit of a particular array element can be accessed at any given time and this access can be done in constant time. Propose a linear time algorithm that finds the absent element of the array in this setting. Rigorously show your pseudocode and analysis together with explanations. Do not use actual code in your pseudocode but present your actual code as a separate Python program. **(20 points)**

Q3. Propose a sorting algorithm based on quicksort but this time improve its efficiency by using insertion sort where appropriate. Express your algorithm using pseudocode and analyze its expected running time. In addition, implement your algorithm using Python. **(20 points)**

Q4. Solve the following recurrence relations

a) $x_n = 7x_{n-1}-10x_{n-2}$, $x_0=2$, $x_1=3$ **(4 points)**

b) $x_n = 2x_{n-1}+x_{n-2}-2x_{n-3,}$ $x_0=2$, $x_1=1$, $x_2=4$ **(4 points)**

c) $x_n = x_{n-1}+2^n$, $x_0=5$ **(4 points)**

d) Suppose that a^n and b^n are both solutions to a recurrence relation of the form $x_n=\alpha x_{n-1}+\beta x_{n-2}$. Prove that for any constants c and d, ca^n+db^n is also a solution to the same recurrence relation. **(8 points)**

Q5. A group of people and a group of jobs is given as input. Any person can be assigned any job and a certain cost value is associated with this assignment, for instance depending on the duration of time that the pertinent person finishes the pertinent job. This cost hinges upon the person-job assignment. Propose a polynomial-time algorithm that assigns exactly one person to each job such that the maximum cost among the assignments (not the total cost!) is minimized. Describe your algorithm using pseudocode and implement it using Python. Analyze the best case, worst case, and average-case performance of the running time of your algorithm. **(20 points)**

1) $\ln^3 n < \sqrt[4]{n} < 5^n < (n!)^n < (n^2)!$

Yusuf Can Kaş
16104640074

$\Rightarrow \lim \frac{\sqrt[4]{n} - \ln^3(n)}{\frac{\ln^3(n)}{\sqrt[4]{n}}} = \lim \frac{3 \ln \ln n}{\frac{1}{4} \cdot \ln n} = 0$ ($\ln n$ grows faster than $\ln \ln(n)$)

so: $\sqrt[4]{n} > \ln^3 n$

$\Rightarrow \lim \frac{\sqrt[4]{n} - 5^n}{\frac{\sqrt[4]{n}}{5^n}} = \lim \frac{\frac{1}{4} \log n}{n \log 5}$ l'hopitals rule $= \lim \frac{\frac{1}{4n}}{1} = \lim \frac{1}{4n} = 0$ so $5^n > \sqrt[4]{n}$

$\Rightarrow \lim \frac{5^n - (n!)^n}{\frac{5^n}{(n!)^n}} = \lim \frac{n \cdot \log 5}{n \cdot \log(n!)} = \lim \frac{\log 5}{\log(n!)} = 0$ so $(n!)^n > 5^n$

$\Rightarrow (n!)^n - (n^2)!$

$n! \sim \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n$

$= \lim_{n \to \infty} \frac{(n!)^n}{(n^2)!} =$ from stirlings formula $= \frac{\left(\sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n\right)^n}{\sqrt{2\pi n^2} \cdot \left(\frac{n^2}{e}\right)^{n^2}} = \lim_{n \to \infty} \frac{\left(\sqrt{2\pi n}\right)^n \cdot \left(\left(\frac{n}{e}\right)^n\right)^n}{\sqrt{2\pi} \cdot n \left(\frac{n^2}{e}\right)^{n^2}} =$

$= \frac{\left(\sqrt{2\pi}\right)^n \cdot \sqrt{n}^n \cdot \left(\frac{n}{e}\right)^{n^2}}{\sqrt{2\pi} \cdot n \cdot \left(\frac{n^2}{e}\right)^{n^2}} = \lim \frac{\left(\sqrt{2\pi}\right)^{n-1} \cdot \left(\sqrt{n}\right)^{n-2}}{(n)^{n^2}} = \lim \frac{\left(\sqrt{2\pi}\right)^{n-2} \cdot \left(\sqrt{2\pi}\right) \cdot \left(\sqrt{n}\right)^{n-2}}{n^{n-2} \cdot (n)^{n^2-(n-2)}} =$

$(n)^{n-2} \cdot (n)^{n^2-(n-2)}$

$= \lim \frac{\left(\sqrt{2\pi}\right)^{n-2} \cdot \sqrt{2\pi}}{\left(\sqrt{n}\right)^{n-2} \cdot n^{n^2-(n-2)}} = \lim \left(\frac{\sqrt{2\pi}}{\sqrt{n}}\right)^{n-2} \cdot \sqrt{2\pi} \cdot \frac{1}{n^{n^2-(n-2)}} = \frac{1}{\infty} = 0$

$\left(\frac{\sqrt{2\pi}}{\sqrt{n}}\right) \to 1/\infty$; $\sqrt{2\pi} \to -$; $\frac{1}{n^{n^2-(n-2)}} \to \frac{1}{\infty}$ so

$(n!)^n < (n^2)!$

2) procedure which_number (array, result)

```
if array is empty
    return result
endif
for i in array:
  if LSB is '0':
     add i to array1 (except LSB)
  else:
     add i to array2. (except LSB)
  endif
endfor

if  size(array1) > size(array2):
    add 1 to result
    which_number (array2, result)
else:
    add 0 to result
    which_number (array1, result)
endif
end
```

Yusuf Can Kaya
16104007

LSB => least significant Bit.

each time iterates n times and divides array to 2.

Basically every time function looks the last bit and calculates counts. If count of (LSB=0) is bigger than count of (LSB=1), it means the bit is 1, otherwise 0. After this operation procedure adds bit to result and calls itself again with divided arrays. If the bit was 0, it call itself with elements that ends with 0, otherwise it calls itself with elements that ends with 1 and does the same count comparison operation to 1 left bit.

Each time function iterates all array for looking LSB, it consumes n complexity. $\leftarrow$ size.

Each recursive call divides array to two. So;

$T(n) = T(n/2) + n$

- $T(n/2)$: recursive call
- n: iterating array for looking LSB.

from master theorem;

$a = 1$, $b = 2$, $d = 1$

if $a < b^d \Rightarrow \Theta(n^d)$

Complexity is $\Theta(n)$ //

3) procedure quick_insertion_sort (arr, low, high)

Yusuf Can Kaş
161421009

```
while low<high do :
    if high.low +1 < threshold :
        insertionSort (arr, low, high)
        return
    else:
        pivot= rearrage (arr, low, high)
        if  high-pivot > pivot -low :
            quick_insertion-sort (arr , low, pivot-1) # recursive call.
            low= pivot +1                              # same as normal quicksort.
        else:
            quick_insertion_sort (arr, pivot +1, high)
            high =pivot -1.
        endif
    endif
end endwhile
```

This program works as same as quicksort. It divides array same as quicksort, does the rearrage operation same as quicksort. The improvement comes in, if array size smaller than particular size. In this time instead of quick sort, insertion sort works and does the sorting more efficiently than quicksort. Since the quick sort divides array smaller size, using insertion sort makes procedure more faster.

All the time complexities are same as quick sort, because it does the same procedure (divide, rearrage). If the array is small it uses insertion sort so;

General time complexity => $O(n \lg n)$

One of the other improvement is, if the small array is in sorted (or nearly sorted) situation instead of quicksort $O(n^2)$ method, we use best case of insertion sort $(O(n))$. In worst case of small array is, if small array is inserted (reverse) => $O(n^2)$

Yusuf Can Kaya
16104007

4) a) $x_n = 7x_{n-1} - 10x_{n-2}$, $x_0 = 2$, $x_1 = 3$

$r^2 - 7r + 10 = 0$ => characteristic equation

$r \quad -5$

$r \quad -2$ | characteristic roots => +5, +2

$a_n = \alpha_1 \cdot (+5)^n + \alpha_2 \cdot (+2)^n$

$x_0 \Rightarrow \quad \alpha_1 + \alpha_2 = 2$ |-2

$+5\alpha_1 + 2\alpha_2 = 3$

$+3\alpha_1 = -1 \qquad \alpha_2 = 2 + 1/3$

$\alpha_1 = -1/3 \qquad \alpha_2 = 7/3$

so;

$x_n = -1/3 \cdot (5)^n + \frac{7}{3} \cdot (2)^n$

b) $x_n = 2x_{n-1} + x_{n-2} - 2x_{n-3}$, $x_0 = 2$, $x_1 = 1$, $x_2 = 4$

$r^3 - 2r^2 - r + 2 = 0$ => characteristic equation

first root = 1

$r^3 - 2r^2 - r + 2 \;|\; r - 1$

$-r^3 + r^2$

$-r^2 - r + 2$

$-r^2 + r$

$-2r + 2$

$-2r + 2$

0

$\Rightarrow \quad (r-1) \cdot (r^2 - r - 2)$ => characteristic roots => 1, 2, -1

$r^2 - r - 2$

$r \quad -2$

$r \quad 1$

1

2, -1

$\alpha_1 1^n + \alpha_2 2^n + \alpha_3 (-1)^n$

$x_0 \Rightarrow \quad \alpha_1 + \alpha_2 + \alpha_3 = 2$

$x_1 \Rightarrow \quad \alpha_1 + 2\alpha_2 - \alpha_3 = 1$

$x_2 \Rightarrow \quad \alpha_1 + 4\alpha_2 + \alpha_3 = 4$

$2\alpha_1 + 3\alpha_2 = 3$

$2\alpha_1 + 6\alpha_2 = 5$

$\alpha_2 = 2/3 \qquad \alpha_1 = 1/2 \qquad \alpha_3 = 5/6$

Therefore;

$x_n = 1/2 \, 1^n + \frac{2}{3} 2^n + \frac{5}{6} (-1)^n$

Yusuf Can Kaş
161044007

4) c) $x_n = x_{n-1} + 2^n \quad x_0 = 5$

$F(n) = 2^n$

$P_n = c.2^n$

↓ put it in equation.

$c.2^n = c.2^{n-1} + 2^n$

↳ divide both sides to 2^{n-1}

$\frac{c.2^n}{2^{n-1}} = \frac{c.2^{n-1}}{2^{n-1}} + \frac{2^n}{2^{n-1}} \Rightarrow 2c = c + 2$

$c = 2$

Homogen part

$x - 1 = 0$

$x = 1 \rightarrow$ first root.

$a(n) = \alpha_1 (1)^n + 2.2^n$

↙ homogen roots ↘ hetegen part

$x_0 = 5 \Rightarrow \alpha_1 + 2 = 5$

$\alpha_1 = 3$

So; $a(n) = 3(1)^n + 2^{n+1}$

$a(n) = 3 + 2^{n+1}$

d) let's put a_n and b_n to the equation

$a^n = \alpha . a^{n-1} + \beta a^{n-2}$

$b^n = \alpha . b^{n-1} + \beta . b^{n-2}$

let's put $c.a^n$ and $d.b^n$ to the equation

$c.a^n = \alpha . c . a^{n-1} + \beta . c . a^{n-2}$

$d.b^n = \alpha . d . b^{n-1} + \beta . d . b^{n-2}$

+

$ca^n + db^n = \alpha (ca^{n-1} + db^{n-2}) + \beta (c.a^{n-2} + d\,b^{n-2})$

↳ This equation satisfies the recurrence relation form. So for any constant c and d, $ca^n + db^n$ is solution for this relation.

Yusuf Can Kes
1604404007

5) procedure assignment(arr):

```
for i from 0 to sizeof(arr):
    i, j = find_max(arr) # finds the max. element of array. Assigns indexes to i and j.
    i, j = find_min(arr, i, j) # finds the minimum element
                               # in the given column and row.
    add (i, j) to result
    Set -1 to all ith row and all jth column.
endfor
return result
end.
```

Explanation of algorithm: Function first finds the maximum element in all array. After that it looks the row and the column of max element and finds the min element. Adds min element to the result (this element represent (index of south person, index of column jobs cost) After that, since we assign the job and person for that row and column, it sets row of min and column of min to -1. Repeats the same process until all the jobs are assigned.

for example;

persons \ jobs	J	5	2
	8	1	6
	(9) max elen.	7	5

=>

(3) min elen.	5	2
8	1	6
(9)	7	5

=> result 3

-1	-1	-1
-1	1	6
-1	(7) max elen	5

=>

-1	-1	-1
-1	(1)	6
-1	(7) min elen	5

=> result 3, 1

-1	-1	-1
-1	-1	-1
-1	-1	(5)

=>

result: Person 0 assigned to job 0. Cost = 3
person 1 assigned to job 1. Cost = 1
person 2 assigned to job 2. Cost = 5

Complexity analysis:

Yusuf Can Kes
16104007

- The for loop iterates n times. $O(n)$
- The find-max function iterates every element one by one. It looks n column and n row and, every time it runs, it iterates all the array. So; $O(n^2)$
- The find_min function iterates only given row and given column. Every time it runs it iterates 1 row and 1 column. So $O(n+n) \Rightarrow O(2n) \Rightarrow O(n)$
- Appending result array has constant time complexity.
- Setting -1 to given colum and row every time iterates 1 column and row. So; $O(n+n) \Rightarrow O(n)$

So, $O(n) \Rightarrow$ for loop

$O(n^2)$
$O(n)$
$O(1)$
$O(n)$
$\Big\rangle O(n^2) \Big\rangle \underline{O(n^3)}$

In best - worst and average case all the complexities are the same. $O(n^3)$.

The reason of that is, different cases of the array does not decreases the time complexity of my algorithm.

find-max is always has $O(n^2)$ complexity.

find_min is always has $O(n)$ complexity.

Setting -1 to row and column always has $O(n)$ complexity.

All the complexities are $\underline{O(n^3)}$