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
171044077 Mustafa Tokgöz						

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

Mustafa Tokgöz 1710464077

Answer of Question 1)

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

$(5^n, \sqrt[4]{n})$

$$\lim_{n\to\infty} \frac{5^n}{\sqrt[4]{n}} = \lim_{n\to\infty} \frac{5^n}{n^{1/4}} \left(\frac{\infty}{\infty}\right)$$

I use L'hopital rule to salve

$$\lim_{n\to\infty} \frac{\frac{d}{dn} 5^n}{\frac{d}{dn} n^{1/4}} = \lim_{n\to\infty} \frac{5^n \ln(5)}{\frac{1}{4} n^{-3/4}} = \lim_{n\to\infty} 5^n \cdot \ln(5) \cdot 4 \cdot n^{3/4} = \infty$$

Because the limit answer is ∞, then growth rates of them

$\boxed{5^n > \sqrt[4]{n}}$

$(\sqrt[4]{n}, \ln^3(n))$

$$\lim_{n\to\infty} \frac{\sqrt[4]{n}}{\ln^3(n)} = \lim_{n\to\infty} \frac{n^{1/4}}{\ln^3(n)} \left(\frac{\infty}{\infty}\right)$$

To salve this limit, I use L'hopital rule

$$\lim_{n\to\infty} \frac{\frac{d}{dn} n^{1/4}}{\frac{d}{dn} \ln^3(n)} = \lim_{n\to\infty} \frac{\frac{1}{4} n^{-3/4}}{3 \cdot \frac{1}{n} \cdot \ln^2(n)} = \lim_{n\to\infty} \frac{\frac{1}{4} n^{1/4}}{3 \ln^2(n)} \left(\frac{\infty}{\infty}\right)$$

So, I use L'hopital rule again then, $\frac{1}{12} \lim_{n\to\infty} \frac{\frac{d}{dn} n^{1/4}}{\frac{d}{dn} \ln^2(n)}$

$$= \frac{1}{12} \lim_{n\to\infty} \frac{\frac{1}{4} n^{-3/4}}{2 \cdot \frac{1}{n} \ln(n)} = \frac{1}{8} \cdot \frac{1}{12} \lim_{n\to\infty} \frac{n^{1/4}}{\ln(n)} \left(\frac{\infty}{\infty}\right), \text{ Again I}$$

use L'hopital rule $\frac{1}{96} \cdot \lim_{n\to\infty} \frac{\frac{d}{dn} n^{1/4}}{\frac{d}{dn} \ln(n)} = \frac{1}{96} \lim_{n\to\infty} \frac{\frac{1}{4} n^{-3/4}}{\frac{1}{n}}$

$$\to \frac{1}{96} \cdot \frac{1}{4} \lim_{n\to\infty} n^{1/4} = \infty$$

Because the answer of this limit is ∞, then growth rates of them are $\boxed{\sqrt[4]{n} > \ln^3(n)}$

$((n!)^n, 5^n)$

$$\text{limit}_{n\to\infty} \frac{(n!)^n}{5^n} = \lim_{n\to\infty} \left(\frac{n!}{5}\right)^n$$

$$x = e^{\ln(x)} \Rightarrow \lim_{n\to\infty} \left(\frac{n!}{5}\right)^n = \lim_{n\to\infty} e^{\ln\left(\frac{n!}{5}\right)^n} = \lim_{n\to\infty} e^{n \ln\left(\frac{n!}{5}\right)}$$

$$= e^{\lim_{n\to\infty} n \ln\left(\frac{n!}{5}\right)} = \lim_{n\to\infty} n \ln\left(\frac{n!}{5}\right) = \infty \Rightarrow$$

$$\lim_{n\to\infty} \frac{(n!)^n}{5^n} = \infty$$

Because the answer of this limit is ∞, then growth rates of them are $\boxed{(n!)^n > 5^n}$

$((n^2)!, (n!)^n)$

$$\text{limit}_{n\to\infty} = \frac{(n^2)!}{(n!)^n}$$

I use Stirling's formula because of factorial

Stirling's formula $\Rightarrow n! \cong \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

$$\text{limit}_{n\to\infty} = \frac{\sqrt{2\pi n^2}\left(\frac{n^2}{e}\right)^{n^2}}{\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)^n} = \lim_{n\to\infty} \frac{\sqrt{2\pi n^2}\left(\frac{n^2}{e}\right)^{n^2}}{\left(\sqrt{2\pi n}\right)^n \left(\frac{n}{e}\right)^{n^2}} \Rightarrow$$

$$\Rightarrow \lim_{n\to\infty} \frac{\sqrt{2\pi n^2}\left(\frac{n^2}{e}\cdot\frac{e}{n}\right)^{n^2}}{(\sqrt{2\pi n})^n} = \lim_{n\to\infty} \frac{\sqrt{2\pi n^2}\, n^{n^2}}{(\sqrt{2\pi n})^n} = \lim_{n\to\infty} \frac{\sqrt{2\pi}\cdot n\cdot n^{n^2}}{(\sqrt{2\pi})^n \cdot \sqrt{n}^n}$$

$$\Rightarrow \lim_{n\to\infty} \frac{n^{n^2}}{(2\pi)^{n-1}\, n^{\frac{1}{2}n-1}} = \lim_{n\to\infty} \frac{n^{n^2}}{(2\pi)^{n-1}\cdot n^{\frac{1}{2}n-1}} \cdot \left(\frac{n^{1/2n}}{n^{1/2n}}\right) \Rightarrow$$

$$\lim_{n\to\infty} \frac{n^{n^2-1/2n}}{(2\pi)^{n-1}\cdot n^{n-1}} = \lim_{n\to\infty} \frac{n^{n^2-1/2n}}{(2\pi n)^{n-1}} \cdot \left(\frac{(2\pi n)^{n^2-3/2n+1}}{(2\pi n)^{n^2-3/2n+1}}\right)$$

$$\lim_{n\to\infty} \frac{n^{n^2-1/2n}\cdot(2\pi n)^{n^2-3/2n+1}}{(2\pi n)^{n^2-1/2n}} = \lim_{n\to\infty} \left(\frac{n}{2\pi n}\right)^{n^2-1/2n}\cdot(2\pi n)^{n^2-3/2n+1}$$

$$\Rightarrow \lim_{n\to\infty} \left(\frac{1}{2\pi}\right)^{n^2-1/2n}\cdot(2\pi n)^{n^2-3/2n+1} = \infty$$

Because the limit answer is ∞, then growth rates of then $\boxed{(n^2)! > (n!)^n}$

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

The growth rates are

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

Answer of Question 2)

Pseucode of My Algorithm:

```
procedure find_missing_integer (array)
var i:=0
if array[0][(length array[0])-1] != "0"
        return 0
for i to (length array) -1
    var temp1 = array[i]
    var temp2 = array[i+1]
    if temp1[(length temp1) -1] ==
        temp2[(length temp2) -1]::
            return i+1
    end if
end for
```

Explanation:
In my algorithm, Firstly I check the first binary integer's last bit to ensure that is zero or not then I traverse the string array with for loop, then I assign ith element of array to temp1 and (i+1)th element to temp2. Then if the last bit of number in temp1 is equal to the last bit of number in temp2, then I make it return i+1 for the missing integer. Why I check the last bits of numbers is that numbers goes that way even, odd, even, odd, So 0000, 0001, 0010, 0011, If last bits are equal, then sequance is even,even or odd,odd. Therefore It can't be even, even and odd, odd, I check the missing element looking this by increasing i.

For example:

array = {"0000", "0001", "0011"}

first if statement is obey because 000[0] is equal to zero.

then for loop

→ 000[0] == 000[1] 0 == 1 i = 0

→ 000[1] == 001[1] 1 == 1 i = 1

return i+1

the resut is 2 that is correct integer.

Analysis of this algorithm:

O(1) ← var i = 0

O(1) ← { if array[0] [(length array[0]) - 1] != "0"
return 0

for i to (length array) - 1

O(1) ← var temp1 = array[i]

O(1) ← var temp2 = array[i+1]

O(1) { if temp1 [(length temp1) - 1] == temp2 [(length temp2) - 1]:
return i+1
end if

O(n-1) = O(n)

end for

The time complexity is O(1) + O(1) + O(n) = O(n) because of for loop. Best case Ω(1) because of if statement. So the complexity for this procedure is = **O(n)** linear time.

Answer of Question 3)

pseudocode of the algorithm=

```
procedure insertion_sort (array, start, end)
    for i = start+1 to end+1 :
        element = array[i]
        J = i-1
        while J>=0 and array[J]>element:
            array[J+1] = array[J]
            J = J-1
        end while
        array[J+1] = element
    end for
end procedure

procedure partition (array, start, end):
    pivot = array[end]
    i = start-1
    for J = start to end
        if array[J] < pivot:
            i = i+1
            temp = array[i]
            array[i] = array[J]
            array[J] = temp
        end if
    end for
    temp = array[i+1]
    array[i+1] = array[end]
    array[end] = temp
    return i+1
```

```
procedure quicksort_with_insertion_sort(array, start, end):
    if start < end:
        size = start - end + 1
        if size < 6
            insertion_sort(array, start, end)
        else:
            pivot = partition (array, start, end)
            quicksort_with_insertion_sort(array, start,
                                          pivot-1)
            quicksort_with_insertion_sort (array, pivot+1
                                           , end)
        end if
    end if
end procedure
```

Explanation and Analysis:

In my algorithm, I use the insertion sort when the size of array is less than 6 because when size of array is less than 6, then insertion sort is better than the quick sort. If we don't ignore the constant factors, time complexity of quick sort is $O(k(n \log n))$ and, time complexity of insertion sort is $O(m(n^2))$, $k, m \in \mathbb{R}$ So when $(m(n^2)) < (k(n \log n))$, insertion sort is faster than the quick sort. So arrays with small size is better for sorting by insertion sort. Insertion sort also use less memory. Also insertion sort is better best case than quick sort's because insertion sort's best case $\Omega(n)$ for sorted arrays. These are all

improving efficiency and complexity. The best case of this algorithm is $\Omega(n)$ thanks to insertion sort, when array is sorted and array size is less than 6. And the complexity is $O(n \log n)$ it does not change but more efficiently. The worst case is $\theta(n^2)$ but it also more efficiently thanks to insertion sort. Because in insertion sort, there are less overhead than quicksort. Why the complexity is $O(n \log n)$ even I use insertion sort is that I use insertion sort for small size array where $m(n^2) < k(n \log n)$, m and k constant factors. The algorithm [best case $\Omega(n)$], Avarage case [$O(n \log n)$], [worst case $\theta(n^2)$].

Answer of Question 4)

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

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

characteristic roots $= r^2 - 7r + 10 = 0 \quad (r-5)(r-2) = 0$

$r_1 = 5 \quad r_2 = 2$

$x_n = \alpha_1 r_1^n + \alpha_2 r_2^n \Rightarrow x_n = \alpha_1 5^n + \alpha_2 2^n$

$x_0 = 2 \Rightarrow \alpha_1 5^0 + \alpha_2 2^0 = 2 \Rightarrow \alpha_1 + \alpha_2 = 2 \quad \backslash -2$

$x_1 = 3 \Rightarrow \alpha_1 5^1 + \alpha_2 2^1 = 3 \Rightarrow +\ 5\alpha_1 + 2\alpha_2 = 3$

$-2\alpha_1 - 2\alpha_2 = -4 \quad \Rightarrow \quad 3\alpha_1 = -1 \quad \alpha_1 = -\frac{1}{3}$

$5\alpha_1 + 2\alpha_2 = 3 \quad \alpha_2 = \frac{7}{3}$

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

b) $x_n = 2x_{n-1} + x_{n-2} - 2x_{n-3}$ $\quad x_0 = 2 \quad x_1 = 1$

$x_2 = 4$

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

characteristic roots = 1 is a root of this

$r_1 = 1 \quad r - 1 = 0$

$$\begin{array}{r|l} r^3 - 2r^2 - r + 2 & r - 1 \\ -(r^3 - r^2) & \hline \\ \hline -r^2 - r + 2 & r^2 - r - 2 \\ -(-r^2 + r) & \\ \hline -2r + 2 & \\ -(-2r + 2) & \\ \hline 0 & \end{array}$$

$r^2 - r - 2 = 0 \quad (r-2)\cdot(r+1) = 0 \quad r_2 = 2 \quad r_3 = -1$

$x_n = \alpha_0 r_1^n + \alpha_1 r_2^n + \alpha_2 r_3^n$

$x_n = \alpha_0 1^n + \alpha_1 2^n + \alpha_2 (-1)^n$

$$\left.\begin{array}{l} \alpha_0 + \alpha_1 + \alpha_2 = 2 \\ \alpha_0 + 2\alpha_1 - \alpha_2 = 1 \\ \alpha_0 + 4\alpha_1 + \alpha_2 = 4 \end{array}\right\} \begin{array}{l} -\frac{1}{2}\alpha_0 + 3\alpha_1 = 3 \\ 2\alpha_0 + 6\alpha_1 = 5 \\ \hline 3\alpha_1 = 2 \end{array} \quad \alpha_1 = \frac{2}{3}$$

$\alpha_0 = \frac{1}{2}$

$\frac{1}{2} + \frac{2}{3} + \alpha_2 = 2 \rightarrow \alpha_2 = 2 - \frac{7}{6}$

$\alpha_2 = \frac{5}{6}$

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

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

$F(n) = 2^n$

$x_n = \{x_n^{(p)}\} + \{x_n^{(h)}\}$ (particular, homogeneous)

$x_n = x_{n-1}$ = homogeneous part

characteristic equation $r - 1 = 0$ $r = 1$

$\{x_n^{(h)}\} = \alpha 1^n = \alpha$

$F(n) = 2^n$ $\{x_n^{(p)}\} = c \cdot 2^n$

$c \cdot 2^n = c \cdot 2^{n-1} + 2^n$

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

$\frac{1}{2}c = 1$ $c = 2$ $\{x_n^{(p)}\} = 2 \cdot 2^n = 2^{n+1}$

$x_n = \alpha + 2^{n+1}$

$x_0 = \alpha + 2^1 = 5$ $\alpha = 3$

$x_n = 3 + 2^{n+1}$

d)

$$X_n = \alpha X_{n-1} + \beta X_{n-2}$$

if a^n is a solution then

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

and b^n is also solution then

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

I will choose an n to find α and β with this two equation.

$n=1$

$$a = \alpha + \beta \frac{1}{a} \quad | -1$$

$$b = \alpha + \beta \frac{1}{b}$$

$$b - a = \beta\left(\frac{1}{b} - \frac{1}{a}\right) \Rightarrow b - a = \beta\left(\frac{a-b}{ba}\right)$$

$$\Rightarrow -1 = \beta \frac{1}{ab} \qquad \boxed{\beta = -ab} \qquad \boxed{\alpha = a+b}$$

$$X_n = (a+b) X_{n-1} - ab X_{n-2}$$

$$c a^n + d b^n = X_n$$

So

$$ca^n + db^n = (a+b)(ca^{n-1} + db^{n-1}) - ab \cdot (ca^{n-2} + db^{n-2}) = ca^n + db^{n-1}a + ca^{n-1}b + db^n - cba^{n-1} - db^{n-1}a = ca^n + db^n = X_n$$

$$ca^n + db^n = ca^n + db^n$$

$ca^n + db^n$ is also solution of X_n

Answer of Question 5)

pseudocode of my algorithm

```
procedure find_min_index (array):
    var min_index=0
    if (array[0]==-1:
        min_index=1
    end if
    for i to length array
        if array[i] != -1 && array[i] <
                                array[min_index]
            min index =i
        end if
    end for
    return min_index
end procedure

procedure find_max_index (array):
    var max_index=0
    if array[0] ==-1:
        max_index=1
    end if
    for i to length array
        if array[i] != -1 && array[i] >
                                array[max_index]
            max_index =i
        end if
    end for
    return max_index
end procedure
```

```
procedure clone (a):
    var new = deepcopy of a
    return new
end procedure

procedure algorithm (array):
        var count_max=0
        var arr2= clone(array)
        var result
        var last_result

    for i to length array:
            count_max2=count_max
            count_max=0
            if i==1:
                last_result= result
            end if
            for j to length array:
                min = find_min_index (arr2[j])
                max = find_max_index (arr2[j])
                if max==min:
                    count_max +=1
                end if
                result[j]= arr2[j][min]
                if i!=(length array)-1:
                    for k to length array:
                        arr2[k][min] =-1
                    end for
                end if
            end for
```

```
        if count_max <= count_max2:
            last_result = result

        end if
        arr2 = clone(array)
        mini = find-min-index (array[i])
        arr2[i][mini] = -1
        array[i][mini] = -1

    end for

    return last_result

end procedure
```

Best Case = $\Omega(n^3)$

Average Case = $\Theta(n^3)$

Worst Case = $O(n^3)$

Explanation and Analysis of my algorithm:

In my algorithm, I traverse array with for loop then keeping the maximum of the result and the inner loop, I find minimum index of that assignment and maximum index of the that assignment, then I check if the min is equal to max or not. If it is then I increase the count of max Because I need to know the number of the maximum in the result. to choose minimum of maximum. After that I assign minimum number to result. Then I assign that column -1 not to choose that column. Then it goes inner for loop.

Then Outter loop I control the result has less maximum tha last result or not. Then if it is, I assigns. Then I turn the array original but assigns -1 to ith row minimum element And at the end of function, last result has the solution of the algorithm. After that I make it return last result. And the complexity of this algorithm is $O(n^3)$ because $O(1) * O(n) * O(n)$ ($O(n)$ for loop, $O(n)$ for loop)

$(O(1) + O(1) + O(1)) + O(n) = O(n^3)$

find_min_index, find_max_index, for loop, clone

And the others $O(1)$ time that does not effects running time. The best case is $\Omega(n^3)$. The worst case is $O(n^3)$ and average case is $O(n^3)$.

Therefore the algorithm complexity $O(n^3)$