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 Akgül 171044007						

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) $(n!)^n$

Solution Q1: a) 5^n b) $4\sqrt{n}$ c) $\ln^3(n)$ d) $(n^2)!$ e) $(n!)^n$

Comparing $4\sqrt{n}$ and $\ln^3(n)$

$$\lim_{n\to\infty} \frac{4\sqrt{n}}{\ln^3 n} = \frac{\infty}{\infty} \rightarrow \text{L'Hospital's Rule}$$

$$\lim_{n\to\infty} \frac{\frac{1}{4} n^{-3/4}}{\frac{3\ln^2(n)}{n}} = \lim_{n\to\infty} \frac{\frac{1}{4n^{3/4}}}{\frac{3\ln^2(n)}{n}} = \lim_{n\to\infty} \frac{n}{12\ln^2(n)\cdot n^{3/4}} = \frac{\infty}{\infty}$$

Again L'Hospital

$$\frac{1}{12} \lim_{n\to\infty} \frac{\frac{1}{4\cdot n^{3/4}}}{\frac{2\ln(n)}{n}} = \frac{1}{12} \lim_{n\to\infty} \frac{n}{8n^{3/4}\ln(n)} = \lim_{n\to\infty} \frac{4\sqrt{n}}{96\ln n} = \frac{\infty}{\infty}$$

Again L'Hospital

$$\frac{1}{96} \lim_{n\to\infty} \frac{\frac{1}{4\cdot n^{3/4}}}{\frac{1}{n}} = \frac{1}{96} \cdot \lim_{n\to\infty} \frac{n}{4n^{3/4}} = \lim_{n\to\infty} \frac{n^{1/4}}{384} = \infty$$

Since the result of limit is ∞, $\boxed{4\sqrt{n} > \ln^3(n)}$

(Also polynomials functions grows faster than logaritmic functions)

Comparing 5^n and $4\sqrt{n}$

$$\lim_{n\to\infty} \frac{5^n}{4\sqrt{n}} = \frac{\infty}{\infty} \rightarrow \text{L'Hospital's Rule}$$

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

Since the result of limit is ∞, $\boxed{5^n > 4\sqrt{n}}$

Comparing $(n!)^n$ and 5^n ; I will use the Stirling's formula to make the comparision

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

$$\lim_{n\to\infty} \frac{(n!)^n}{5^n} = \lim_{n\to\infty} \frac{(n!)^n}{5^n} = \lim_{n\to\infty} \left(\frac{n!}{5}\right)^n$$ then use stirling's formula

$$\lim_{n\to\infty} \left(\frac{\sqrt{2\pi n}\cdot\left(\frac{n}{e}\right)^n}{5}\right)^n$$ then take log of this limit

$*\boxed{\text{note} = \ x = e^{\ln x}}$

$$e^{\lim_{n\to\infty} \ln \left(\frac{\sqrt{2\pi n}\cdot\left(\frac{n}{e}\right)^n}{5}\right)^n} = e^{\lim_{n\to\infty} \underbrace{n}_{\infty} \underbrace{\ln\left(\frac{\sqrt{2\pi n}\cdot\left(\frac{n}{e}\right)^n}{5}\right)}_{\infty}}$$

$$e^{\infty\cdot\infty} = \infty$$

Since the result of limit ∞, $\boxed{(n!)^n > 5^n}$

Comparing $(n^2)!$ and $(n!)^n$; I will use the stirling's formula to make the comparision.

Stirling's formula $= n! \cong \sqrt{2\pi n}\cdot\left(\frac{n}{e}\right)^n$ for large n.

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

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

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

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

Since the result of limit ∞, $\boxed{(n^2)! > (n!)^n}$

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

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)**

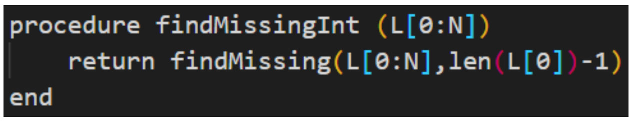
My algorithm steps ;

1- Start with the least significant bit; Create 2 arrays, one for zero bits and one for one bits.
2- If least significant bit is zero then push number zero bits array, else push one bits array.
3- So,if size(1's) < size(0's), then the missing number has 1 it's laeast significant bit, else it has 0.
4- Repeat steps 1 to 3, iterating from least sign bit (2LSB, 3LSB ... most sign bit.) and using an array of greater length (zero bits or one bits array) .

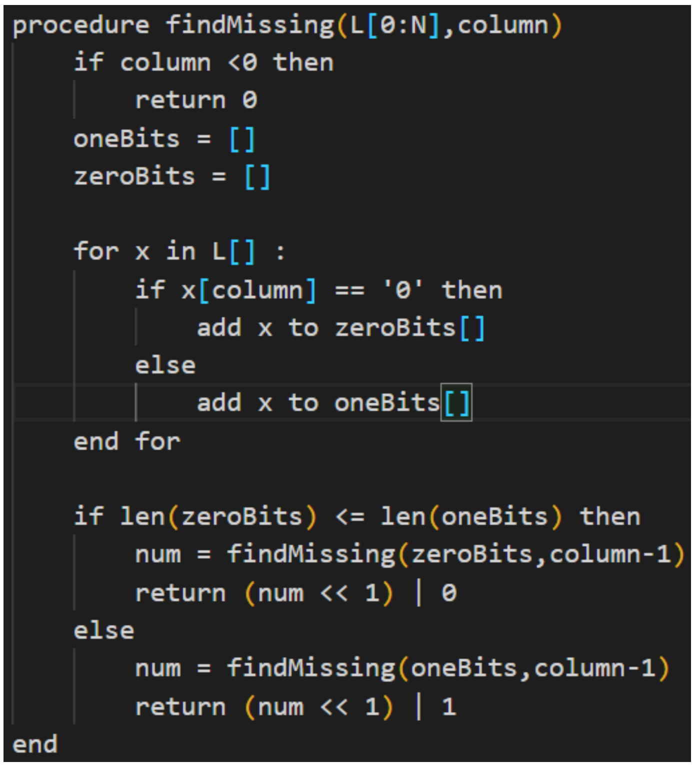
Also i know; if n is odd so the number of 0 bits and 1 bits should be the same in Last significant bit.They won't be, so the lower number is obviosly the missing one.
If n is even, so the number of 0 bits should be 1 greater than the number of 1 bits.If they are equal, it's the 0 bit missing one.Again if n is even , if the number of 0 bits is 2 greater than the number of 1 bits, the 1 bit is the missing one. By removing all the numbers that have been eliminated, applying the exact same problem again with recursively.

By the end of this process, we will have computed all bits in missing number. In each successive iteration, we look at n, then n / 2, then n / 4, and so on, bits. This results in a runtime of **O(N).**

Pseudo Codes ;



```
procedure findMissingInt (L[0:N])
    return findMissing(L[0:N],len(L[0])-1)
end
```

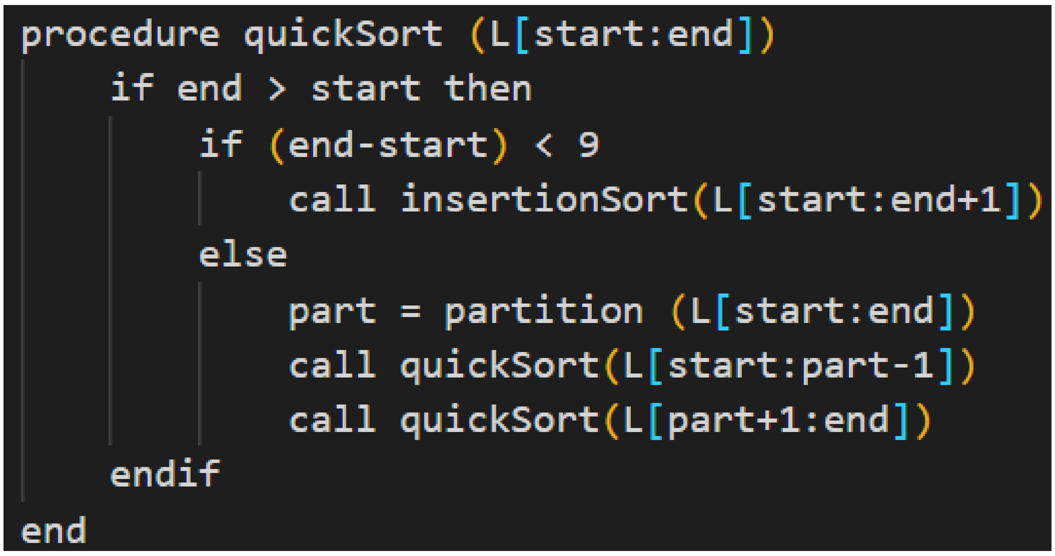


```
procedure findMissing(L[0:N],column)
    if column <0 then
        return 0
    oneBits = []
    zeroBits = []

    for x in L[] :
        if x[column] == '0' then
            add x to zeroBits[]
        else
            add x to oneBits[]
    end for

    if len(zeroBits) <= len(oneBits) then
        num = findMissing(zeroBits,column-1)
        return (num << 1) | 0
    else
        num = findMissing(oneBits,column-1)
        return (num << 1) | 1
end
```

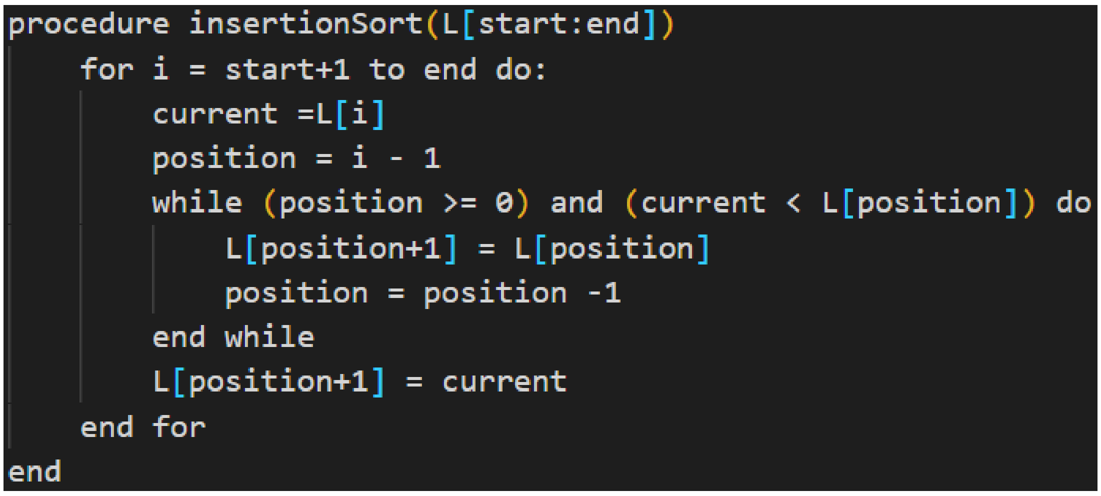
Q3. Propose a sorting algorithm based on quicksort but this time improve its efficiency by using insertion sort where appropriate. Express your algorithm using pseudocode and analyze its expected running time. In addition, implement your algorithm using Python. **(20 points)**

Quick sort is a much more efficient algorithm for sorting large data than insertion sort.But, insertion sort is faster for small data because Quick Sort has extra overhead from the recursive function calls. Insertion sort is also more stable than Quick sort and requires less memory. Based on this information, I wrote a new sort. I divide the data into smaller data using the quick sort algorithm and sort these data with insertion sort. I named this algorithm is hybirdSort.
I chose the threshold point as 9 for array. Because the threshold point for when the running time is significantly better in insertion sort than that of quicksort is around 9.

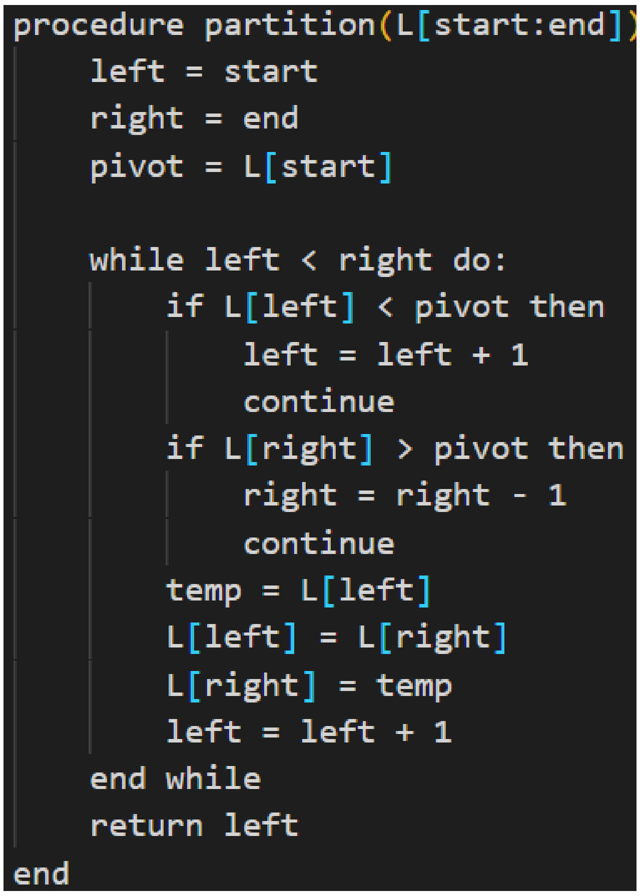
Pseudo Codes ;



```
procedure quickSort (L[start:end])
    if end > start then
        if (end-start) < 9
            call insertionSort(L[start:end+1])
        else
            part = partition (L[start:end])
            call quickSort(L[start:part-1])
            call quickSort(L[part+1:end])
    endif
end
```



```
procedure insertionSort(L[start:end])
    for i = start+1 to end do:
        current =L[i]
        position = i - 1
        while (position >= 0) and (current < L[position]) do
            L[position+1] = L[position]
            position = position -1
        end while
        L[position+1] = current
    end for
end
```



```
procedure partition(L[start:end])
    left = start
    right = end
    pivot = L[start]

    while left < right do:
        if L[left] < pivot then
            left = left + 1
            continue
        if L[right] > pivot then
            right = right - 1
            continue
        temp = L[left]
        L[left] = L[right]
        L[right] = temp
        left = left + 1
    end while
    return left
end
```

Analysis ;

In the best case; If the incoming data size is less than 9, which is the threshold point, and is in order.Then only insertion sort execute and time complexity of this situation is;

Best case = **O(n).**

In the avarage case; When the number of elements is less than some threshold k(9),then we stop. This process takes Θ(nlogn) time in total for the quick sort part.Later when the whole array has been processed, each element will be at most k positions away from its final sorted position.Now if we perform insertion sort on it, it will take O(kn) time to finish the sort, which is linear as k is a constant.Then time complexity of this situation is;

Avarage case = Θ(nlogn+n) = **O(nlogn)**

In the worst case; the worst case would occur when the array is already sorted in decreasing or increasing order. When the number of elements is less than some threshold k(9),then we stop. This process takes $O(n^2)$ time in total for the quick sort part.Then perform insertion sort on it, it will take O(kn) time (proved in the average case description). Then time cimplexity of this situation is ; Worst case = $O(n^2+n)$ = **$O(n^2)$**

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)**

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

$x_n - 7x_{n-1} + 10x_{n-2} = 0$

Lets say; $x_n = \alpha^2$ $\quad x_{n-1} = \alpha$ $\quad x_{n-2} = \alpha^0 = 1$ then;

$\alpha^2 - 7\alpha + 10 = 0$ $\rightarrow$ Characteristic equation

$(\alpha - 5)(\alpha - 2) = 0$

$\alpha_1 = 5$ and $\alpha_2 = 2$ $\quad$ 5 and 2 are our characteristic roots.

Recurrence relation form $\rightarrow$ $\underline{x_n = c_1(\alpha_1)^n + c_2(\alpha_2)^n}$

so $x_n = c_1 \cdot 5^n + c_2 \cdot 2^n$ then;

for $x_0 = 2$ $\quad 2 = c_1 . 5^0 + c_2 . 2^0 = c_1 + c_2$

for $x_1 = 3$ $\quad 3 = c_1 . 5^1 + c_2 . 2^1 = 5c_1 + 2c_2$

After solving this 2 equations, i find;

$c_1 = \frac{-1}{3}$ $\quad c_2 = \frac{7}{3}$ then result;

$$\boxed{x_n = \frac{7}{3} 2^n - \frac{1}{3} 5^n}$$ Recurrence relation

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

$x_n - 2x_{n-1} - x_{n-2} + 2x_{n-3} = 0$

Lets say; $x_n = \alpha^3$ $x_{n-1} = \alpha^2$ $x_{n-2} = \alpha$ $x_{n-3} = \alpha^0 = 1$

$\alpha^3 - 2\alpha^2 - \alpha + 2 = 0 \rightarrow$ characteristic equation

$\alpha^2(\alpha-2) - (\alpha-2) = 0$

$(\alpha-2)(\alpha^2-1)$

$\alpha_1 = 2$, $\alpha_2 = 1$, $\alpha_3 = -1$ are characteristic roots

Recurrence relation form;

$x_n = c_1(\alpha_1)^n + c_2(\alpha_2)^n + c_3(\alpha_3)^n$

So $x_n = c_1 . 2^n + c_2 . 1^n + c_3 . (-1)^n$

for $x_0 = 2$ $2 = c_1 . 2^0 + c_2 . 1^0 + c_3 . (-1)^0 = c_1 + c_2 + c_3$

for $x_1 = 1$ $1 = c_1 . 2^1 + c_2 . 1^1 + c_3 (-1)^1 = 2c_1 + c_2 - c_3$

for $x_2 = 4$ $4 = c_1 . 2^2 + c_2 . 1^2 + c_3 . (-1)^2 = 4c_1 + c_2 + c_3$

After solving this 3 equations, i find;

$c_1 = \frac{2}{3}$ $c_2 = \frac{1}{2}$ $c_3 = \frac{5}{6}$ then result;

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

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

Since the equations is not homogeneous, we will examine it in 2 parts.

$x_n^g = x_n^h + x_n^p$

$x_n^h \longrightarrow x_n = x_{n-1}$

$x_n - x_{n-1} = 0$

Let's say $x_n = \alpha$ $x_{n-1} = \alpha^0 = 1$ then

$\alpha - 1 = 0 \longrightarrow \alpha = 1$

$\boxed{x_n^h = a.1^n}$

$F(n) = 2^n$ then $x_n^p = c.2^n$

$c.2^n = c.2^{n-1} + 2^n \rightarrow 2^n(c) = 2^n(c2^{-1}+1) \Rightarrow c = \frac{c}{2} + 1$

$\boxed{c = 2}$

$\boxed{x_n^p = 2.2^n = 2^{n+1}}$ then;

$x_n^g = x_n^h + x_n^p$

$x_n^g = a.1^n + 2^{n+1}$

for $x_0 = 5$ $5 = a.1^0 + 2^{0+1} = a+2$

$3 = a$ then result

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

d) If a^n and b^n are both solutions to the given recurrence relation form then we can say;

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

Then multiply constants c and d

$$\begin{array}{ll} & c \cdot a^n = c \cdot \alpha \cdot a^{n-1} + c \cdot \beta a^{n-2} \\ + & d \cdot b^n = d \cdot \alpha \cdot b^{n-1} + d \cdot \beta b^{n-2} \\ \hline & c \cdot a^n + d \cdot b^n = \alpha(ca^{n-1} + d b^{n-1}) + \beta(ca^{n-2} + db^{n-2}) \end{array}$$

This is the proof, if a^n and b^n both solutions to recurrence relations of given form above, then $ca^n + db^n$ is also solutions for any constants c, d.

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)**