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
Muhammed Yasir Fidan 161044056						

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) $(n!)^n$

Firstly, lets compare $\sqrt[4]{n}$ and $\ln^3(n)$

$$\lim_{n\to\infty} \frac{\sqrt[4]{n}}{\ln^3 n} \overset{\text{L'Hospital}}{=} \lim_{n\to\infty} \frac{\frac{1}{4}\cdot n^{-\frac{3}{4}}}{\frac{3\ln^2 n}{n}}$$

$$\overset{\text{L'Hospital}}{=} \lim_{n\to\infty} \frac{\frac{1}{4}\cdot\frac{-3}{4}\cdot n^{-\frac{7}{4}}}{3\left(\frac{\frac{2\ln(n)}{\cancel{n}}\cdot \cancel{n} - \ln^2(n)}{n^2}\right)} = \lim_{n\to\infty} \frac{-\frac{1}{16}\, n^{\frac{1}{4}}}{2\ln(n) - \ln^2(n)}$$

$$\overset{\text{L'Hospital}}{=} \lim_{n\to\infty} \frac{-\frac{1}{64}\cdot n^{-\frac{3}{4}}}{\frac{2}{n} - \frac{2\ln(n)}{n}} = \lim_{n\to\infty} \frac{-\frac{1}{64}\cdot n^{\frac{1}{4}}}{2 - 2\ln(n)}$$

$$\lim_{n\to\infty} \frac{-\frac{1}{64}\cdot n^{\frac{1}{4}}}{2-2\ln(n)} \overset{\text{L'Hospital}}{=\!=} \lim_{n\to\infty} \frac{-\frac{1}{256}\cdot n^{-\frac{3}{4}}}{-\frac{2}{n}} = \lim_{n\to\infty} \frac{1}{512}\cdot n^{\frac{1}{4}} = \infty$$

limit result is ∞, because of that $\underline{\sqrt[4]{n} > \ln^3(n)}$ (1)

Now lets compare 5^n and $\sqrt[4]{n}$

$$\lim_{n\to\infty} \frac{5^n}{n^{\frac{1}{4}}} \overset{\text{L'Hospital}}{=\!=} \lim_{n\to\infty} \frac{5^n \cdot \ln 5}{\frac{1}{4\cdot n^{3/4}}} = \lim_{n\to\infty} 5^n \cdot \ln 5 \cdot 4 \cdot n^{3/4} = \infty$$

This limit result is ∞, so it means $5^n > \sqrt[4]{n}$ (2)

and from (1) and (2) we can see $5^n > \sqrt[4]{n} > \ln^3(n)$ (3)

Now lets compare $(n^2)!$ and $(n!)^n$

$$\lim_{n\to\infty} \frac{(n^2)!}{(n!)^n} = \lim_{n\to\infty} \frac{n^2\cdot(n^2-1)\cdot(n^2-2)\cdots 2\cdot 1}{\underbrace{n!\cdot n!\cdot n!\cdots n!}_{n \text{ times}}}$$

for this, every $\frac{n\text{th term}}{n!}$ will be greater or equal 1

for example lets say our $n=3$ then;

$$\frac{9\ \ 8\ \ 7}{3!}\ \ \frac{6\ \ 5\ \ 4}{3!}\ \ \frac{3\ \ 2\ \ 1}{3!}$$

this equal 1

This greater than 1

This greater than 1.

This will be satisfied for all n, so when $n\to\infty$ the numerator will bigger than denomitor. So this limit goes to ∞.

Because limit result is ∞, it means $(n^2)! > (n!)^n$ (4)

Lastly, compare 5^n and $(n!)^n$

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

This limit results is 0 becouse $\frac{5}{n!}$ goes 0 for $n\to\infty$ $\left(\frac{c}{\infty}=0\right)$

and $\lim_{n\to\infty} 0^\infty = 0$

so, becouse of limit result 0, growth rate of $(n!)^n$ bigger tha 5^n so; $(n!)^n > 5^n$ (5)

After combine (3), (4) and (5) we found

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

from lowest to highest

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

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)**

```
function    FindAbsent( L [1:n][1:m])
        prev = 1
        for i = 1 to n do
            if (L[i][m] = prev) then
                return i-1
            end if
            prev = L[i][m]
        end for
end
```

In my algorithm, I store integers in my list as binary representations.
So, I use 2D array. My array form like this;

```
[
  [0],
  [0,1],
  [1,0],
  [1,1],
]
```

So, like this binary representation is used for the array elements.

In binary representation, for consecutive numbers, the least significant bit always change (0 to 1 or 1 to 0). So I can store previous integer's least significant bit and traverse list then check previous bit and its consecitive number's least significant bit. If they are both 0 or 1 that mens there is an absent integer.

Also this is a linear algorithm. I just traverse list, access binary representation's least significant bit and compare it with prev variable. This access and compare operation is constant time. So this algorithm has a linear time complexity.

Best Case

If absent integer is 0 best case will be occur, In this case

$B(n) = 1 \in \Theta(1)$

Worst case

Worst case occurs if we found absent integer in last for iteration

In this case $W(n) = n \in \Theta(n)$

Average case

We know there is always 1 absent integer in our array. So, the probability of absent integer being any index is $\frac{1}{n}$.

$$A(n) = \sum_{i=1}^{n} i.\frac{1}{n} = \frac{1}{n} + \frac{2}{n} + \frac{3}{n} + \cdots \frac{n}{n} = \frac{n.(n+1)}{n} \in \Theta(n)$$

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)**

```
function QuickSort(L[low:high])
    if (low<high)
        if (high-low <=5)
            InsertionSort(L[low:high])
        end if
        else
            piv = partitation(L[low:high])
            QuickSort(L[low: piv-1])
            QuickSort(L[piv+1: high])
        end else
    end if
end

function partitation(L[low:high])
    pivot = L[low]
    right = high
    left = low

    while left<right do
        repeat left=left+1 while left<right and L[left]<=pivot
        repeat right=right-1 while L[right] > pivot
        if left<right
            temp = L[left]
            L[left] = L[right]
            L[right] = temp
        end if
    end while
    L[low] = L[right]
    L[right] = pivot
    return right
end
```

```
function    InsertionSort( L[low:high])

        for i = low to high   do
            temp = L[i]
            j = i
            while j > 0 and temp < L[j-1] do
                    L[j] = L[j-1]
                    j = j-1

            end while
            L[j] = temp

        end for
   end
```

Insertion sort is a fast algorithm for small size and already sorted arrays.
Quick sort place pivot element to the right place then divede array
2 smaller array. When array size smaller than a number I set (6)
then my algorithm sort this by using insertion sort. Becuse of
that our sorting algorithm fasts because insertion sort more efficient
for sorting smaller and almost sorted array than quick sort.

$T = T_1 + T_2 + T_3$

$T_1 \rightarrow$ # of operation in partitation

$T_2 \rightarrow$ # of operation in recursive call

$T_3 \rightarrow$ # of operation in insertion sort

$A(n) = E[T] = E[T_1] + E[T_2] + E[T_3]$

We can ignore $E[T_3]$ because here insertion sort doesn't depend
on input n. It always call for array size smaller than 6. So its # of
operation not depend on input size because of that its $\Theta(1)$

So $A(n) \in \Theta(1)$ for $n < 6$

and for $n \geq 6$ it's average time complexity will be same as quick sort.

$$A(n) = \begin{cases} A(n) \in \Theta(n \log n) & \text{for } n \geq 6 \\ A(n) \in \Theta(1) & \text{for } n < 6 \end{cases}$$

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)**

a) $X_n = 7X_{n-1} - 10X_{n-2}$ $\qquad \alpha^2 = 7\alpha - 10$ $\qquad \alpha^2 - 7\alpha + 10 = 0$

$-2 \quad -5$

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

$\alpha_1 = 2 \quad \alpha_2 = 5$ roots real and distinct

So;

$x(n) = c_1 \alpha_1^n + c_2 \alpha_2^n$

$x(n) = c_1 2^n + c_2 5^n$

$-2/ \quad x(0) = c_1 + c_2 = 2$

$x(1) = 2c_1 + 5c_2 = 3$

$3c_2 = -1$

$c_2 = -\frac{1}{3}$ $\qquad c_1 = 2 + \frac{1}{3} = \frac{7}{3}$ $\qquad x(n) = \frac{7}{3} 2^n - \frac{1}{3} 5^n$

b) $X_n = 2X_{n-1} + X_{n-2} - 2X_{n-3}$

$\alpha^3 = 2\alpha^2 + \alpha - 2$

$\alpha^3 - 2\alpha^2 - \alpha + 2 = 0$

$\alpha^2(\alpha-2) - 1(\alpha-2) = 0$

$(\alpha^2-1).(\alpha-2) = 0$

$(\alpha-1).(\alpha+1)(\alpha-2) = 0$

$(\alpha-1).(\alpha+1)(\alpha-2)=0$

$\alpha_1=1 \quad \alpha_2=-1 \quad \alpha_3=2$

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

$x(0)=2 \qquad x(1)=1 \qquad x(2)=4$

$2=c_1+c_2+c_3$

$1=c_1+2c_2-c_3$

$4=c_1+4c_2+c_3$

Using gauss-elimination

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 2 \\ 1 & 2 & -1 & 1 \\ 1 & 4 & 1 & 4 \end{array}\right] \xrightarrow{-R_1+R_3\to R_3} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 2 \\ 1 & 2 & -1 & 1 \\ 0 & 3 & 0 & 2 \end{array}\right]$$

$$\xrightarrow{-R_1+R_2\to R_2} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 2 \\ 0 & 1 & -2 & -1 \\ 0 & 3 & 0 & 2 \end{array}\right] \xrightarrow{-3R_2+R_3\to R_3} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 2 \\ 0 & 1 & -2 & -1 \\ 0 & 0 & 6 & 5 \end{array}\right]$$

$6c_3=5 \qquad c_2-2c_3=-1 \qquad c_1+c_2+c_3=2$

$c_3=\frac{5}{6} \qquad c_2=-1+\frac{10}{6}=\frac{4}{6}=\frac{2}{3} \qquad c_1+\frac{2}{3}+\frac{5}{6}=2$

$c_1=\frac{1}{2}$

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

c) $X_n = X_{n-1} + 2^n$

$f(n) = 2^n$ $\quad X_n = X_n^h + X_n^p$

$X_n = X_{n-1}$ $\quad X_n - X_{n-1} = 0$

$x - 1 = 0 \longrightarrow$ homogene part

$X_n^h = \alpha 1^n$

$X_n = X_{n-1} + 2^n$

$A 2^n - A 2^{n-1} = 2^n$

$A - \frac{A}{2} = 1 \Rightarrow \frac{A}{2} = 1 \Rightarrow A = 2$

$X_n^p = A 2^n = 2^{n+1}$

$X_n = X_n^h + X_n^p = \alpha . 1^n + 2^{n+1}$

$X(0) = \alpha + 2 = 5 \Rightarrow \alpha = 3$

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

d)

$X(n) = a^n$ $\quad X(n) \stackrel{?}{=} c a^n + d b^n$

$X(n) = b^n$

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

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

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

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

$+$

$c a^n + d b^n = (c + d) . \underbrace{(\alpha X_{n-1} + \beta X_{n-2})}_{X_n}$

$\frac{c a^n + d b^n}{(c+d)} = X_n$

$X_n = \frac{c . X_n + d . X_n}{c + d} = \frac{\cancel{(c+d)} X(n)}{\cancel{c+d}} \Rightarrow X(n) = X(n)$ ✓

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)**