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
171044024 Nevzat Seferoğlu						

Read the instructions below carefully

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

- 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!

①

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

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

$$= \lim_{n\to\infty} \frac{6\ln(n) \cdot n^{-1}}{\frac{1}{16} \cdot n^{\frac{-3}{4}}} = \lim_{n\to\infty} \frac{6\ln(n)}{\frac{1}{16} \cdot n^{\frac{1}{4}}}$$

$$= \lim_{n\to\infty} \frac{6 \cdot n^{-1}}{\frac{1}{64} \cdot n^{\frac{-3}{4}}} = \frac{6}{\frac{1}{64} \cdot n^{\frac{1}{4}}} //$$

$$\ln^3(n) \in o\left(\sqrt[4]{n}\right)$$

little o

$$\lim_{n\to\infty} \frac{\sqrt[4]{n}}{5^n} = \lim_{n\to\infty} \frac{\frac{1}{4} \cdot n^{\frac{-3}{4}}}{5^n \cdot \ln 5} = \lim_{n\to\infty} \frac{\frac{1}{4}}{5^n \cdot \ln 5 \cdot n^{\frac{3}{4}}} = 0 // \quad \text{(L'Hospital)}$$

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

little o

$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \rightarrow$ Stirling's Approximation.

$$\lim_{n\to\infty} \frac{5^n}{\left(\sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)\right)^n} = \lim_{n\to\infty} \left(\frac{5}{\underbrace{\sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)}}\right)^n = 0 //$$

↳ approximate through zero

$$5^n \in o\left((n!)^n\right)$$

little o

$$n! \sim \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \rightarrow \text{Stirling's Approximation}$$

$$\lim_{n \to \infty} \frac{(n!)^n}{(n^2)!}$$

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

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

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

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

$$(n!)^n \in o\left(n^2\right)!$$

little o

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)**

2

make exponential operation constant time

```
binary_to_int(binary_num, i=0):
    length = length of binary_num
    if i == (length - 1)
        return integer(binary_num[i])
    return (integer(binary_num[i]) << (length-i-1)) +
                    binary_to_int(binary_num, i+1))
```

constant 8 bits of 1 and 0 $\Theta(1)$

```
q3(arr):
    arr_length = length of arr
    expected_sum = arr_length * (arr_length+1) / 2
    cur_sum = 0

    for i in arr:
        cur_sum += binary_to_int(i)

    return expected_sum - cur_sum
```

$\Theta(1)$

$\Theta(n)$

* Firstly, for binary_to_int function works in constant time, I suppose that every single given string has same amount of digit which is 8 bits of '1' and '0'. Therefore binary_to_int works in constant time.
* There is a for loop which calculates the current sum of integers and works in linear time.
* Total complexity will be $\Theta(n)$ at the 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)**

3

```
rearrange (arr, low, high):
    pivot = arr[high]
    index = low
    i = low

    while i < high:
        if arr[i] <= pivot:
            swap(arr, i, index)
            index +=1
        i +=1
    swap(arr, index, high)
    return index
```

* works for each pivot selection.
complexity = Θ(n) since n is the size of given array

```
swap (arr, i1, i2)
    temp = arr[i1]
    arr[i1] = arr[i2]
    arr[i2] = temp
```

swap to elements of an array.

Constant time.

```
insertion Sort (arr, low, high):
    i = low +1
    while i<=high:
        value = arr[i]
        j = i
        while (j > low and arr[j-1] > value):
            arr[j] = arr[j-1]
            j -=1
        arr[j] = value
        i +=1
```

Apply insertion-sort only for the given (low, high) space.

```
quick Sort (arr, low, high):
    while low < high:
        if (high - low) < 9 :
            insertion Sort (arr, low, high)
            break
        else:
            pivot = rearrange(arr, low, high)
            if (pivot - low) < (high - pivot):
                quick Sort (arr, low, pivot -1)
                low = pivot +1
            else:
                quick Sort (arr, pivot +1, high)
                high = pivot -1
```

Threshold value = 9 After that point insertion sort will be applied.

* There will be almost no differences in worst-case of the algorithm. After the threshold value, insertion sort make algorithm faster. Also there will be almost no differences at best case.

worst-case → $O(n^2)$ → maybe better than that

best-case → $O(n\log n)$ → " " " "

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)**

4

a) $x_n = r^2$ $\quad r^2 = 7r - 10$

$x_{n-1} = r$ $\quad r^2 - 7r + 10 = 0$ $\quad r_1 = 2$, $r_2 = 5$

$x_{n-2} = 1$

$x_n = a.5^n + b.2^n$

$x_0 = a + b = 2$ $\quad -2a - 2b = -4$

$x_1 = 5a + 2b = 3$ $\quad 5a + 2b = 3$

$3a = -1$

$a = -1/3$ $\quad b = \frac{7}{3}$

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

b)

$x_n = r^3$

$x_{n-1} = r^2$

$x_{n-2} = r$

$x_{n-3} = 1$

$r^3 = 2r^2 + r - 2$

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

$r^2(r-2) - (r-2) = (r-2)(r^2-1)$

$r_1 = 2$

$r_2 = 1$

$r_3 = -1$

$x_n = a\,2^n + b(-1)^n + c.1^n$

$x_0 = 4a + b + c = 4$

$x_1 = 2a - b + c = 1$

$x_2 = a + b + c = 2$

$$-4\;-2\begin{bmatrix} 1 & 1 & 1 & 2 \\ 2 & -1 & 1 & 1 \\ 4 & 1 & 1 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 1 & 2 \\ 0 & -3 & -1 & -3 \\ 0 & -3 & -3 & -4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 & 2 \\ 0 & -3 & -1 & -3 \\ 0 & 0 & -2 & -1 \end{bmatrix}$$

$-2c = -1$

$c = \frac{1}{2}$ //

$-3b - c = -3$

$-3b - \frac{1}{2} = -3$

$-3b = \frac{-5}{2}$

$b = \frac{5}{6}$ //

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

$a + b + c = 2$

$a + \frac{5}{6} + \frac{1}{2} = 2$

$a = 2 - \frac{8}{6}$ $\quad a = \frac{4}{6} = \frac{2}{3}$ //

c)

$x_n = r$ $\quad x_n = k \cdot 1^n$

$x_{n-1} = r - 1$ $\quad$ homogeneous

$x_n = A \cdot 2^n$

$x_{n-1} = A \cdot 2^{n-1}$

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

$A = \frac{A}{2} + 1$

$x_n = 2 \cdot 2^n$ } non-homogeneous $\boxed{A = 2}$

$x_{n-1} = 2 \cdot 2^{n-1}$

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

$\underset{5}{\underline{x_0}} = k + 2^1$

$5 = k + 2 \longrightarrow \underline{x_n = 3 + 2^{n+1}}$

$k = 3$

d)

$a^n = k \cdot x_{n-1} + p \cdot x_{n-2}$

$b^n = k \cdot x_{n-1} + p \cdot x_{n-2}$

$c \cdot a^n = c \cdot k \cdot a^{n-1} + c \cdot p \cdot a^{n-2}$ } $c \cdot a^n + d \cdot b^n = c \cdot k \cdot a^{n-1} + c p \cdot a^{n-2} + d \cdot k \cdot b^{n-1} + d \cdot k \cdot b^{n-2}$

$d \cdot b^n = d \cdot k \cdot b^{n-1} + d \cdot p \cdot b^{n-2}$ $\quad = k \cdot (c \cdot a^{n-1} + d b^{n-1}) + p\,(c \cdot a^{n-2} + d\, b^{n-2})$

Therefore:

it is obvious that $c \cdot a^n + d \cdot b^n$ is a solution for $\underline{c \text{ and } d}$ if $\underline{a^n \text{ and } b^n}$ are solution for the $\underline{x_n = k \cdot x_{n-1} + p\, x_{n-2}}$

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)**