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
Türker Tercan 171044032						

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) $(n!)^n$

Q1. List the following functions according to their order of growth from to lowest to highest. Prove the accuracy of your ordering.

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

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

$$\lim_{n\to\infty} \frac{\frac{1}{4}n^{\frac{1}{4}-1}}{24 \ln n} = \frac{1}{96 \ln n \cdot \sqrt[4]{n^3}} = 0$$ So $\underbrace{\sqrt[4]{n} < \ln^3(n)}$

$\lim_{n\to\infty} \frac{\ln^3(n)}{5^n} = 0$ because as 5 is real number as 'n' tends to '∞' $(n\to\infty)$

We can say denominator has larger number
so, denominator is greater than numerator

denominator > numerator

let say : $n^2 > 2x$ as we are increasing 'n^2' value. It will go on decreasing than $\frac{1}{2}$

$\frac{1}{2} > \frac{x}{n^2}$ i.e. $n < n_1 < n_2 < n_3 < n_4$

$$\frac{1}{2} > \frac{x}{n_1^2} > \frac{x}{n_2^2} > \frac{x}{n_3^2}$$

we can write

$$\frac{\ln^3 n}{5^n} = \frac{\ln^3 1}{5} \cdot \frac{\ln^3 2}{5} \cdot \frac{\ln^3 3}{5} \cdots \frac{\ln^3 n}{5} < \frac{1}{2}$$

as $n \longrightarrow \infty$

$\lim_{n\to\infty} \frac{\ln^3 n}{5^n} = 0$ $\underbrace{\text{So } \ln^3(n) < 5^n}$

$$\lim_{n \to \infty} \frac{(5^n)}{(n^2)!} = 0$$

we can write

$$\frac{5^n}{(n^2)!} = \frac{5}{1} \cdot \frac{5}{4!} \cdot \frac{5}{9!} \cdot \frac{5}{16!} \cdots \frac{5}{(n^2)!} < \frac{1}{2}$$

as $n \to \infty$

$$\lim_{n \to \infty} \frac{5^n}{(n^2)!} = 0 \quad \underline{\text{So} \quad 5^n < (n^2)!}$$

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

$$\frac{(n^2)!}{(n!)^n} = \frac{1}{1} \cdot \frac{4!}{(2!)^2} \cdot \frac{9!}{(3!)^3} \cdot \frac{16!}{(4!)^4} \cdots \frac{(n^2)!}{(n!)^n} < \frac{1}{2}$$

similarly deminator is greater than numerator

limit will be zero

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

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

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)**

```
procedure convert_to_bits (binary [0:n])
    bits = new bool[32]
    i = 0
    for j in BITSIZE-n to BITSIZE do
        if binary [i] == true
            bits [j] = true
        end if
        else
            bits [j] = false
        end else
        i += 1
    end for
    return bits
end
```

Analyze :

Function converts given binary (boolean) array to a fixed sized boolean array In our case BITSIZE equals to 32

For loop runs n times

$f(n) \in \Theta(n)$

LSB → Least Significant Bit

```
procedure find_absent(array, LSB)
    if LSB < 0
        return 0
    odd Numbers = []
    even Numbers = []
    for-each i in array
        if i[LSB]
            oddNumbers.append(i)
        end if
        else
            even Numbers.append(i)
        end else
    end for
    if odd_numbers.length >= even_numbers.length
        retVal = find_absent (evenNumbers, LSB -1)
        retVal = retVal << 1
        return retVal
    end if
    else
        retVal = find_absent (oddNumbers, LSB -1)
        retVal = retVal << 1
        retVal = retVal + 1
        return retVal
    end else
end
```

Analyze : We can determine the absent number by looking at the least significant bits.

Ex

Let's say 2 is absent from the array

0	0000
1	0001
2	$001\cancel{0}$
3	0011
4	0100

LSB

0000
0001
0011
0100

odd number size and even number size are equal so discard odd numbers

Shift return value

$1 = 1 << 1$

2 //

LSB

0000
0100

They all are even discard even numbers

Shift return value and add 1

$0 = 0 << 1$
$0 = 0 + 1$

1

Empty 0

When LSB reach 0 return 0

Result is 2

So we can find the absent value by looking at the their LSBs and once we traversed all bits we can call it recursively and print the result

```
procedure main()
    array = [ convert_to_bits([False, False, False, False]),
              convert_to_bits([False, False, False, True]),
              convert_to_bits([False, False, True, True]),
              convert_to_bits([False, True, False, False]) ]
    absent = find_absent (array, BIT_SIZE-1)
    print(absent)
end
```

Time Complexity : Initialize of the array → $O(k \cdot n)$ number of array initializations $\in O(n)$

Recursive function → O(BIT_SIZE) in our case $O(32)$ number of recursive calls are always 32 $\in O(1)$

$f(n) \in O(n)$

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)**

```
procedure InsertionSort(L[1 : n])
    for i = 2 to n do
        current = L[i]
        position = i – 1
        while position >= 1 and current < L[position] do
            L[position + 1] = L[position]
            position = position – 1
        end while
        L[position + 1] = current
    end for
end
```

```
procedure QuickSort(L[low : high])
    if high > low
        position = call Rearrange(L[low : high], position)
        call QuickSort(L[low : position - 1])
        call QuickSort(L[position + 1 : high])
    end if
end
```

```
procedure Rearrange(L[low : high], position)
    right = low
    left = high + 1
    x = L[low]
    while right < left do
        repeat right += 1 until L[right] >= x
        repeat left -= 1 until L[left] < x
        if right < left
            interchange(L[left], L[right])
        end if
    end while
    position = left
    L[low] = L[position]
    L[position] = x
end
```

```
procedure HybridQuickSort(L[low : high], threshold)
        if low >= high
                return
        end if
        if low – high < threshold
                InsertionSort(array, low, high)
        end if
        else
                position = call Rearrange(L[low : high], position)
                call HybridQuickSort(L[low : position -1]
                call HybridQuickSort(L[position + 1 : high]
        end else
end
```

Analyze:

QuickSort algorithm is very efficient for large inputs. But for small inputs InsertionSort is more efficient comparing to the QuickSort because in small arrays the number of comparisons and swaps are less.

When the function is started QuickSort will be applied to the array. After some time, portion of the array size will be smaller than given threshold(k), so we can use InsertionSort instead of QuickSort.

InsertionSort will take O(k . n) which is linear because k is a constant value. I used k as integer 10.

Also, QuickSort takes O(nlogn) time

All function takes O(nlogn) time

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

$\alpha^2 = 7\alpha - 10$

$\alpha^2 - 7\alpha + 10 = 0$ $\quad$ $(\alpha-5)(\alpha-2)=0$

-5, -2 $\quad$ $\alpha_+ = 5 \quad \alpha_- = 2$

$x_n = c_1\alpha_+^n + c_2\alpha_-^n = c_1 5^n + c_2 \cdot 2^n$

$x_0 = c_1 + c_2 = 2 \longrightarrow -2 \quad -2c_1 \cancel{-2c_2} = -4$

$x_1 = 5c_1 + 2c_2 = 3 \quad + \quad 5c_1 \cancel{+2c_2} = 3$

$3c_1 = -1 \quad c_1 = -1/3 \quad c_2 = 2 - c_1$

$= 2 + \frac{1}{3} = \frac{7}{3}$

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

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

$\alpha^3 = 2\alpha^2 + \alpha - 2$ $\quad \alpha_1 = 2 \ \alpha_2 = -1 \ \alpha_3 = 1$

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

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

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

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

$x_0 = c_1 + c_2 + c_3 = 2$

$x_1 = c_1 \cdot 2 + -c_2 + c_3 = 1$

$x_2 = 4c_1 + c_2 + c_3 = 4$

$x_2 - x_0 = 4c_1 + c_2 + c_3 = 4$

$\quad -c_1 - c_2 - c_3 = -2$

$3c_1 = 2$

$c_1 = \frac{2}{3}$

$x_1 + x_0 = c_1 + c_2 + c_3 = 2$

$\quad 2c_1 - c_2 + c_3 = 1$

$3c_1 + 2c_3 = 3$

$2 + 2c_3 = 3$

$c_3 = 1/2$

$\frac{2}{3} + c_2 + \frac{1}{2} = 2$

$c_2 = \frac{12 - 3 - 4}{6} = \frac{5}{6}$

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

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

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

$x_n - x_{n-1} = f(n)$

$x_n = x_0 + \sum_{i=1}^{n} f(i)$

$x_n = 5 + \sum_{i=1}^{n} 2^i$

$a_1 = 2 \quad r = 2$

$\sum_{i=1}^{n} 2^i = \frac{a_1 (1 - r^n)}{(1 - r)} = 2 \cdot \frac{1 - 2^n}{1 - 2} = -2(1 - 2^n)$

$x_n = 5 - 2 + 4^n$

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

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)**

Q5.

Lets Create an example

	Job 1	Job 2	Job 3	Job 4	Job 5
Person 1	550	210	1250	1235	999
Person 2	186	842	982	396	453
Person 3	666	245	852	367	1750
Person 4	110	962	456	385	750
Person 5	975	265	310	1125	870

1- Give each Job-person to random non negative value

2- Our goal is minimize the maximum cost among the assignments (not the total cost)

	Job 1	Job 2	Job 3	Job 4	Job 5
Person 1	550	210	1250	1235	999
Person 2	186	842	982	396	453
Person 3	666	245	852	367	1750
Person 4	110	962	456	385	750
Person 5	975	265	310	1125	870

3- We need to minimize these values. In order to do that lets select maximum assignment

Job 5
999
453
P3 666 245 852 367 1750
750
870

	J1	J2	J3	J4	J5
P1	550	210	1250	1235	999
P2	186	842	982	396	453
~~P3~~	~~666~~	~~245~~	~~852~~	~~367~~	~~1750~~
P4	110	962	456	385	750
P5	975	245	310	1125	870

How to minimize these value
- Find the minimum value within its row or column
- In our case (245)
- After that assign Job 2 to Person 3 so 1750 can not be used

	J1	J2	J3	J4	J5
P1	550	210	~~1250~~	1235	999
P2	186	842	982	396	453
~~P3~~	~~666~~	~~245~~	~~852~~	~~367~~	~~1750~~
P4	110	962	456	385	750
~~P5~~	~~975~~	~~245~~	~~310~~	~~1125~~	~~870~~

4- After that find maximum value (1250)

	J1	J2	J3	J4	J5
P1	550	~~210~~	1250	1235	999

J3:
1250
982
~~852~~
456
310

- Find min value = 310
- Assing J3 → P5

	J1	J4	J5
P1	550	1235	999
P2	186	396	453
P4	110	385	750

- Find max = 1235

		J4	
P1	550	1235	999
		396	
		385	

	J1	J4	J5
P1	550	1235	999
P2	186	396	453
~~P4~~	~~110~~	385	~~750~~

- Find min = 385
  Assign J4 → P4

	J1	J5
P1	550	999
P2	186	453

- Find max 999 to minimize it

		J5
P1	550	999
		453

- Find min = 453
- Assign P2 → J5

	J1	J5
P1	550	999
~~P2~~	~~186~~	453

- And assign P1 → J1
  So the total

5th	J1	J2	J3	J4	J5
P1	550	210	1250	1235	999
P2	186	842	982	396	453 4th
P3	666	1st 245	852	367	1750
P4	110	962	456	3rd 385	750
P5	975	265	310 2nd	1125	870

```
procedure minimize_maximum_assignments ( Assignment[0:n][0:m] )
// n is person count, m is job count and
// always must be m >= n
  rows = [true] * n
  columns = [true] * m
  total = 0
  links = []
  while rows != [false] * n  and  columns != [false] * m  do
      i = 0
      j = 0
      max=0                                  // In order to find max
      for i=0 to n do                        // First, we need to find
          for j=0 to m do                    // usable first assignment
              if rows[i] and columns[j]
                  max = assignment[i][j]
                  break
              end if
          end for
      end for
```

```
        x = i
        y = j
        for i=0 to n do
            for j=0 to m do
                if rows[i] and colums[j] and max < assignment [i][j]
                    max = assignment [i][j]
                                                    // We found maximum cost
                    x = i                           // among the assignments
                    y = j
                end if
            end for
        end for
        min = max  min_x = x, min_y = y
        for j = 0 to m do
            if rows [x] and columns [j] and min > assignment [x][j]
                min = assignment [x][j]
                min_x = x
                min_y = j
            end if
        end for

        for i=0 to n do
            if rows[i] and columns [y] and min > assignment [i][y]
                min = assignment [i][y]
                min_x = i
                min_y = j
            end if
        end for
        total += assignment [min_x][min_y]
        links.append ([min_x, min_y, assignment [min_x][min_y])
        rows [min_x] = false
        columns [min_x] = false
    end while
end
```

Analyze :

while outer loop → dob size times $O(m)$

1st loop // to find start location → $B(n) = O(1)$ $W(n) = O(n.m)$
$A(n) = O(n.m)$

2nd loop // to find maximum assignment's location → $B(n) = O(1)$ $W(n) = O(n.m)$
$A(n) = O(n.m)$

3rd and 4th // to find maximum assignment's neighbor minimum assignment location → $B(n) = O(1)$ $W(n) = O(n+m)$
$A(n) = O(n+m)$

So all function takes $O(m.m.n)$ time and $\in O(n^3)$

Our function does not rely on given array it always traverses the array. So best and average case is $O(n^3)$

If all the elements are the same worst case occurs. but our expected running time will be again $O(n^3)$