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
OZAN YEŞİLLER 171044033						

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) $(n!)^n$

① $5^n \quad \sqrt[4]{n} \quad \ln^3 n \quad (n^2)! \quad (n!)^n$

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

Proof:

ⓐ $(n^2)! > (n!)^n$

$\lim\limits_{n \to \infty} \frac{(n!)^n}{(n^2)!}$ using string formula $n! \cong \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$

$$\lim_{n \to \infty} \frac{\left(\sqrt{2\pi n} \left(\frac{n}{e}\right)^n\right)^n}{\sqrt{2\pi n} \left(\frac{n^2}{e}\right)^{n^2}} \xrightarrow[\text{L'hopital}]{\text{apply}} \lim_{n \to \infty} \frac{(\sqrt{2\pi})^{n-1} (\sqrt{n})^{n-2} \cdot \left(\frac{n}{2}\right)^{n^2}}{\left(\frac{n^2}{e}\right)^{n^2}}$$

$$\lim_{n \to \infty} (\sqrt{2\pi})^{n-1} \cdot (\sqrt{n})^{n-2} \left(\frac{\frac{n}{e}}{\frac{n^2}{e}}\right)^{n^2} \xrightarrow{\text{simplify}} \lim_{n \to \infty} \frac{(\sqrt{2\pi})^{n-2} \sqrt{2\pi} \cdot (\sqrt{n})^{n-2}}{n^{n-2} \cdot n^{(n^2-(n-2))}}$$

$$\lim_{n \to \infty} \sqrt{2\pi} \left(\frac{\sqrt{2\pi}}{\sqrt{n}}\right)^{n-2} \cdot \frac{1}{n^{(n^2-(n-2))}} \Rightarrow \frac{\text{number}}{\text{inf.}} \Rightarrow \frac{1}{\infty} = 0$$

The limit is zero so it proves $(n!)^n$ growth rate is slower than $(n^2)!$

ⓑ $(n!)^n > 5^n$

$5^n < (n!)^n$

$\sqrt[n]{5^n} < \sqrt[n]{(n!)^n}$

$= |5| < |n!|$

for $n \geq 3$ it is true

ⓒ $\sqrt[4]{n} < 5^n$

$$\lim_{n \to \infty} \frac{5^n}{\sqrt[4]{n}} \xrightarrow{\text{L'H}} \lim_{n \to \infty} \frac{\ln 5 \cdot n}{\frac{1}{4} n^{-\frac{3}{4}}}$$

$$\xrightarrow{\text{Simplify}} \lim_{n \to \infty} 4 \cdot n^{\frac{7}{4}} \cdot \ln 5$$

The limit is infinity so it proves that $\sqrt[4]{n} < 5^n$

ⓓ $\ln^3 n < \sqrt[4]{n}$

$$\lim_{n \to \infty} \frac{\sqrt[4]{n}}{\ln^3 n} \xrightarrow[\text{L'hopital}]{\text{apply}} \lim_{n \to \infty} \frac{\frac{1}{4} \cdot n^{-\frac{3}{4}}}{3 \ln^2 n \cdot n^{-1}} = \lim_{n \to \infty} \frac{n^{\frac{1}{4}}}{12 \ln^2}$$

$$\xrightarrow[\text{L'hopital}]{\text{apply}} \lim_{n \to \infty} \frac{\frac{1}{4} n^{-\frac{3}{4}}}{24 \ln n \cdot \frac{1}{n}} = \lim_{n \to \infty} \frac{n^{\frac{1}{4}}}{96 \ln n}$$

$$\xrightarrow[\text{L'hopital}]{\text{apply}} \lim_{n \to \infty} \frac{\frac{1}{4} n^{-\frac{3}{4}}}{96 \cdot \frac{1}{n}} = \lim_{n \to \infty} \frac{n^{\frac{1}{4}}}{96 \cdot 4}$$

The limit is infinity so it is proves that growth rate of $\sqrt[4]{n}$ is greater than growth rate of $\ln^3 n$

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)**

```
(2)
FUNCTION binary (arr, bitIndex, bitSize = 32)
    index = bitIndex - (bit Size - length of arr))
    if arr[index-1] == "0"
     then do return 0
    else if arr[index-1] == "1"
     then do return 1
   else
     throw exception ("invalid array")
FUNCTION findMissing (arr, bitPos, bitSize)
  if bit Pos < 0
    then do return 0
  oddArr → emptyList
  evenArr → emptyList
  index → 0
 bound → length of array
 while (index < bound)
    if binary (arr[index], bitPos, bitSize) == 0
     then do evenArr. append (arr[index])
  else
    then do oddArr. append (arr[index])
  index → index +1
 if length of oddArr >= length of evenArr
  then do return findMissing (evenArr, bitPos-1, bitSize) << 1 | 0
else
 then do return findMissing (oddArr, bitPos-1, bitSize) << 1 | 1
```

2. Complexity Analysis

This algorithm contains a recursive call. At any step of recursion the algorithm iterate on array almost half of length of array at each step. So the algorithm iterates are

$n + \frac{n}{2} + \frac{n}{4} \dots$

According to master theorem

$T(n) = a \cdot T(\frac{n}{b}) + f(n)$

$a \to 1$, according to this algorithm
$b \to 2$, according to this algorithm
$c \to 1$, according to this algorithm

while $a < b^c$
then $\theta(n)$

$\boxed{T(n) = T(\frac{n}{2}) + n}$

$\boxed{\text{time complexity} \Rightarrow \theta(n)}$

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)**

```
3 FUNCTION parseArr (arr, l, h)
        piv → arr[h]
        i → j → l
      for i = 0 to range(l, h)
        if arr[i] < piv
          then do
            swapping a[i] and a[j]
            j → j+1
      swapping a[j] and a[h]
      return j

   FUNCTION quickS (arr, l, h)
      if(l < h)
       then do
          piv = parseArr (arr, l, h)
          quickS (arr, l, piv+1)
          quickS (arr, piv+1, h)
      return arr

   FUNCTION insertionS (arr, l; h)
      for i = 0 to range (l+1, n+1)
        value = arr[i]
        j = i
        while j > low and arr[j-1] > val
          arr[j] = arr[j-1]
          j → j-1
        arr[j] = val
      return arr
```

FUNCTION hybrid (arr, l, h)

while l<h do

if h-l+1 > 15

Piv = parseArr (arr, l, h)

if piv-l > h-piv

then do

hybrid (arr, piv+1, h)

h → piv+1

else

then do

hybrid (arr, l, piv-1)

l → piv+1

else

then do

insertions (arr, l, h)

Complexity Analysis and Explanation

If the length of array is very large then quick sort is very efficient. But insertion sort more efficient than quick sort if the array length is very small. Number of comparisions and swapping are less compared to quick sort. I combined this algorithms if the number of array size is less than 15 using insertion sort otherwise using quick sort.

quick sort runs average $\theta(n\log n)$, when array's size is small, insertion more efficient than quick sort.

Therefore this algorithm time complexity is $\theta(n\log n)$

Q4. Solve the following recurrence relations

a) $x_n = 7x_{n-1} - 10x_{n-2}$, $x_0=2$, $x_1=3$ **(4 points)**

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

(4)

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

$x_n = y^n$

$$\frac{y^n}{y^{n-2}} = \frac{7y^{n-1}}{y^{n-2}} - \frac{10y^{n-2}}{y^{n-2}} \Rightarrow y^2 = 7y - 10$$

$\boxed{y_1 = 5 \mid y_2 = 2}$

$x_n = c_1 y_1^n + c_2 y_2^n$

$(x_n = c_1 5^n + c_2 2^n)$ and $x_0 = 2, x_1 = 3$

$x_0 = c_1 + c_2 = 2$ $\quad -2/3$

$x_1 = 5c_1 + 2c_2 = 3$

$3c_1 = -1$ $\boxed{c_1 = -1/3}$ $\boxed{c_2 = \frac{7}{3}}$

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

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

(1) $x_n = y^n$

(2)

$$\frac{y^n}{y^{n-3}} = \frac{2y^{n-1}}{y^{n-3}} + \frac{y^{n-2}}{y^{n-3}} - \frac{2y^{n-3}}{y^{n-3}} \Rightarrow \boxed{y^3 = 2y^2 + y - 2} \rightarrow y^3 - 2y^2 - y + 2 = 0$$

(3)

$y^3 - 2y^2 - y - 2 \mid y - 1$

$y^3 - y^2 \quad \mid y^2 - y - 2$

$-y^2 + y$

$-y^2 - y$

$-2y + 2$

$-2y + 2$

0

$(y-1)(y^2-y-2) = 0$

$y_0 = 1, \; y_1 = 2, \; y_2 = -1$

(4) $\Rightarrow x_n = c_1 \cdot 1 + c_2 \cdot 2^n + c_3 (-1)^n$

$x_0 = c_1 + c_2 + c_3 = 2$

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

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

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

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

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

$x_n^h = x_{n-1} \qquad \frac{y^n}{y^{n-1}} = \frac{y^{n-1}}{y^{n-1}} \qquad \boxed{y=1}$

$x_n^p = c_2 . 2^n$

$c_2 . 2^n = c_2 . 2^{n-1} + 2^n$

$c_2 . \cancel{2^n} = \cancel{2^n}(c_2 . 2^{-1} + 1)$

$\frac{c_2}{2} = 1 \Rightarrow \boxed{c_2 = 2}$

$x_n = c_1 + c_2 . 2^n$

$x_0 = c_1 + 2.1 = 5$

$\boxed{c_1 = 3}$

$\boxed{x_n = 3 + 2.2^n}$

(d)

$x_n = \alpha x_{n-1} + \beta x_{n-2}$

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

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

$c . y^n = c . \alpha y^{n-1} + c \beta y^{n-2}$

$d . z^n = d . \alpha z^{n-1} + d \beta z^{n-2}$

$c . y^n + d . z^n = c . \alpha y^{n-1} + c . \beta y^{n-2} + d \alpha z^{n-1} + d \beta z^{n-2}$

$= \alpha (c . y^{n-1} + d . z^{n-1}) + \beta (c\, y^{n-2} + d .\, z^{n-2})$

if y^n, z^n are both solution to recurrence relation of the form x_n then $c . y^n + d . z^n$ is also solution 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)**

```
arrey Size = 4
class Obj :
    init(self):
        self.parent = None
        self.totalCost = 0
        self.cost = 0
        self.worker = 0
        self.job = 0
        self.assigned = [False]* arrey Size

    def __lt__(self, other):
        return self.cost < other.cost

FUNCTION newObj (j1, j2, assigned, parent)
    ob = obj()
    ob.assigned = assigned
    ob.assigned[j2] = True
    ob.parent = parent
    ob.worker = j1
    ob.Job = j2

return ob
```

```
FUNCTION findMinCost(arr):

  queue = PriorityQueue
  assigned = [False]* arraySize
  root = newObj(-1, -1, assigned, obj())
  root.totalcost = root.cost = 0
  root.worker = -1
  queue.put(root)

  while(not queue.empty()):
     min = queue.get()
     i = min.worker + 1
     if(i == arraySize):
        return minCost

   for j in range(0, arraySize):
     if (not minn.assigned[j]):
        child = newObj(i, j, minn.assigned, minn)
        child.totalcost = minn.totalcost + arr[i][j]
        child.cost = child.totalCost + costCalc(arr, i, j,
                                          child.assigned)
        queue.put(child)
```

```
FUNCTION costCalc(arr, j1, j2, assigned):
    cost = 0
    available = [True]* arraysize

    i = j1+1
    while(i < arraysize):
        min = infinity
        min_ind = -1

        j = 0
        while(j < arraysize):
            if(not assigned[j] and available[j] and arr[i][j]<min):
                min-ind = j
                min = arr[i][j]

            j += 1

        cost += min
        available[min-ind] = false
        i += 1

    return cost
```

I wrote the Branch and Bound algorithm for this problem in python language.

This algorithm runs $\Theta(N^4)$

cause of

while (!q.empty()):

for J to N

function call

function

for i to N

for J to N

$O(N) \times O(N) \times O(N) \times O(N)$

$= \Theta(N^4)$