**Data Structures and Algorithms**

**Assignment 1(PART 1)**

**Jan 1 Semester**

**Name: Jeslyn Ho Ka Yan:**

**ID: 8535383**

**Date of Submission: 29 Jan 2024**

**Table of Content**

1 Question 1 3

2 Question 2 4

3 Question 3 5

4 Question 5 10

4.1 Question 5 (i) 10

4.2 Question 5 (ii) 10

# Question 1

|  |  |
| --- | --- |
| **Constant** | 6+𝑛0 |
| **Log-logarithmic** | lg(2lglg𝑛)= log(log n), 3𝑙𝑜𝑔3lg𝑛+𝑛2/3 = log n = 𝑛2/3 |
| **Logarithm** | log(𝑛!), |
| **Polylogarithmic** | (𝑙𝑜𝑔3𝑛)3 |
| **Radicals** | - |
| **Linear** | - |
| **Linearithmic** | lg(𝑛)𝑛 |
| **Polynomial- Quadratic** | (1+2+3+⋯+𝑛), 16lg𝑛=(24)lgn=22lgn(n^2)=(2log n^n)2  =(n 2)2=n4 |
| **Polynomial- Cube** | 23lg𝑛 |
| **Exponentiation** | 16lg𝑛 , 2lg𝑛×(𝑙𝑜𝑔2𝑛)0, 23n |
| **Factorial** | (𝑛2+3)! |

# Question 2

Static void doIt (int n) {

A white rectangular object with black text

Description automatically generated with medium confidence int i

int j ← (2 × n)

loop while ( j > 0 ) {

i ← n

loop while ( i >= 1 ) {

i ← i / 2

}

j ← j – 1

}

}

A black and white text

Description automatically generated with medium confidenceStatic int myMethod (int n) {

sum 🡨 0

for i 🡨 1 to n {

sum = sum + doIt(i)

}

return 1

}

Dolt(It) = 1+2n+2n+Log2(n)+2n

= 1+6n+ Log2(n)

T(n) = n\*(1+6n+ Log2(n))+1+ n+ 1

= n\*(1+6n+ Log2(n))+2+ n

Ans: O n(n log(n)) = **O (n2 log(n))**

# Question 3

## Pseudocode

function findMax(unsortedList, max=NEGATIVE\_INFINITY):

if unsortedList is empty:

return max

// Randomly select an index from the unsortedList

index = random.nextInt(size of unsortedList)

// Retrieve the element at the randomly selected index

element = unsortedList[index]

// Update the max value if the current element is greater

max = maximum(max, element)

// Create a new list without the selected element

updatedList = removeFromList(index, unsortedList)

return findMax(updatedList, max)

function removeFromList(indexToBeRemove, unsortedList):

newList = []

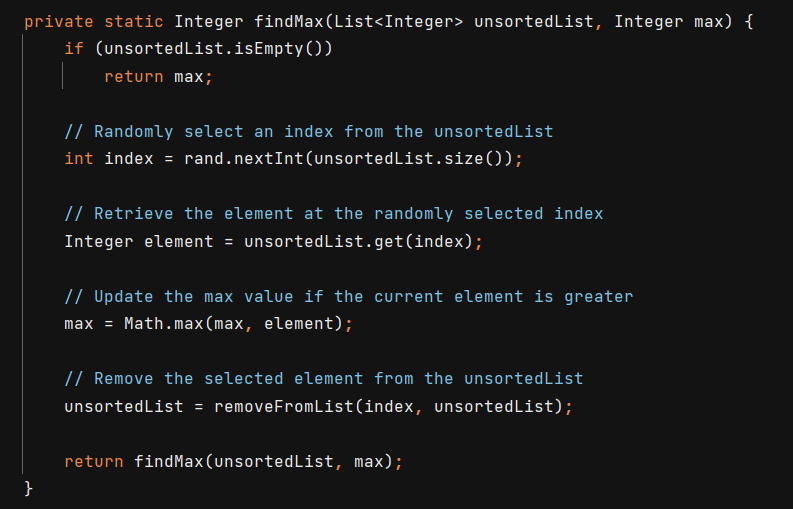
for i = 0 to size of unsortedList - 1: // Iterate through the original list

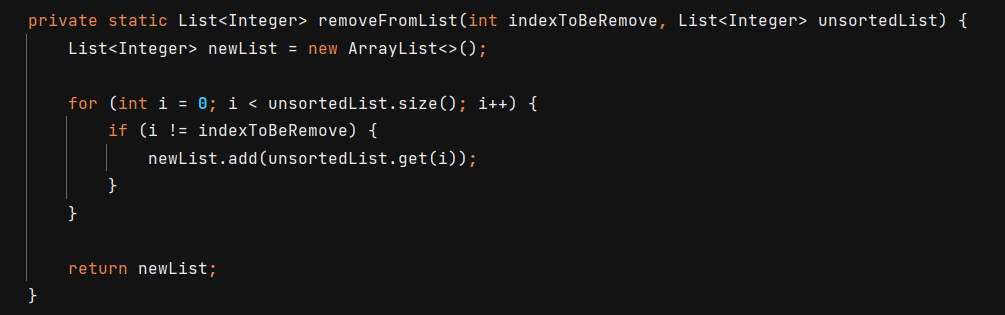
if i is != to indexToBeRemove:

 newList.add(unsortedList[i])

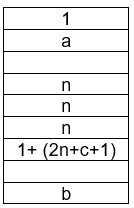
return newList

**Extra (java)**





## Caculate the Running times in terms of Θ notation

function findMax(unsortedList, max=NEGATIVE\_INFINITY):

if unsortedList is empty:

return max

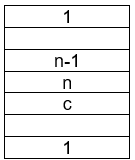
index = random.nextInt(size of unsortedList)

element = unsortedList[index]

max = max(max, element)

unsortedList = removeFromList(index, unsortedList)

return findMax(unsortedList, max)

function removeFromList(indexToBeRemove, unsortedList):

newList = []

for i = 0 to size of unsortedList - 1:

if i is != to indexToBeRemove:

newList.add(unsortedList[i])

return newList

removeFromList(indexToBeRemove, unsortedList):

T(n) = 1+(n-1)+n+c+1

= 2n+c+1

findMax(unsortedList, max):

T(n) =1+a+n+n+n+1+(2n+c+1)+b

= 5n+3+a+b+c

Where:  
 a= 0 or 1 or 0.5 (best case is 1, worst case is 0, average case is 0.5)

b= 0 or 1 or 0.5 (**best case is 0**, **worst case is 1**, average case is 0.5)

c= 0 or 1 or 0.5 (best case is 1 worst case is 0, average case is 0.5)

**Worst Case Scenario 0(n)**

T(n) =5n+3+(0)+(1)+(0) = 5n+ 4

**Average Case Scenario 0(n)**

T(n) =5n+3+(0.5)+(0.5)+(0.5) = 5n+4.5

**Best Case Scenario 0(n)**

T(n) =5n+3+(1)+(0)+(1) = 5n+ 5

# Question 4

1. **𝑇(𝑛)=4𝑇()+𝑛2+𝑛, 𝑎𝑛𝑑 𝑇(1)=1**

𝑇(𝑛)=aT() + f(𝑛)

a =4, b =2, f(n)=n2+n

Lets compare f(n) with nlogba= nlog24=n2

f(n)= n2+n

Since f(n) and nlog24 are in the same order, we will use case 2 of Master Theorem

Therefore, 4𝑇()+𝑛2+𝑛 is 𝑂(n2 log n).

1. **𝑇(𝑛)=2𝑇()+𝑛 𝑙𝑔3𝑛,𝑎𝑛𝑑 𝑇(1)=1**

a=2, b=2, c=1, p=3

= = 1 , use Case 2 from the Master Theorem

Since, p=3, use.

T(n) = 𝑂(n logba × logbp+1 n)

= 𝑂(n log22 × log23+1 n)

= 𝑂(n log24n)

1. **𝑇(𝑛)=3𝑇()+𝑛 𝑙𝑔 𝑛,𝑎𝑛𝑑 𝑇(1)=1**

a=3, b=2, c=1, p=1

use,

T(n) = 𝑂(nlogba)

= 𝑂(nlog23)

1. **𝑇(𝑛)=4𝑇()+𝑛 𝑙𝑔 𝑛,𝑎𝑛𝑑 𝑇(1)=1**

𝑇(𝑛)=4𝑇()+𝑛 𝑙𝑔 𝑛

𝑇(𝑛)=4𝑇 [4T () + ()lg()] + 𝑛 𝑙𝑔 𝑛

𝑇(𝑛)=42𝑇 () + n(lg n−2) + 𝑛 𝑙𝑔 𝑛

𝑇(𝑛)=42𝑇 () + 2n lg n − 2n

𝑇(𝑛)=42𝑇 [4T () + ()lg()] + 2n lg n − 2n

𝑇(𝑛)=43𝑇 () + n(lg n−4) + 2n lg n − 2n

𝑇(𝑛)=43𝑇 () + 3n lg n − 6n

𝑇(𝑛)=43𝑇 [4T ()) + ()lg()] + 3n lg n − 6n

𝑇(𝑛)=44𝑇 () + n(lg n−6) + 3n lg n − 6n

𝑇(𝑛)=44𝑇 () + 4n lg n − 12n

Genralise the Expansion 🡪 4kT () + klg n − (k (k−1))n

The recursive call will stop when ()=1, we have n = 4k, hence k=log4n

Substitute k into the genralise Expansion equation,

4log 4n T () + (log4n) lg n − ( log4n (log4n −1))n

= nT () + n × lg n − ( log4n (log4n −1))n

= nT (1) + n × lg n − n(log4n −1)n

The running time complexity is 𝑂=(n log n)

1. **𝑇(𝑛)=𝑇(𝑛−1)+𝑛2,𝑎𝑛𝑑 𝑇(0)=1**

𝑇(𝑛)= 𝑇(𝑛−1)+𝑛2

𝑇(𝑛)= 𝑇 [((𝑛−1) −1)+ (𝑛−1)2] +𝑛2

𝑇(𝑛)= 𝑇(𝑛−2)+ (𝑛−1)2+𝑛2

𝑇(𝑛)= 𝑇 [((𝑛−2) −1)+ (𝑛−2)2] + (𝑛−1)2+𝑛2

𝑇(𝑛)= 𝑇(𝑛−3)+ (𝑛−2)2 +(𝑛−1)2+𝑛2

𝑇(𝑛)= …..

𝑇(𝑛)=12+22+……+(𝑛−1)2+𝑛2 🡨 Sum of Squares=

𝑇(𝑛)= 𝑇(0) +

𝑇(𝑛)= 1+

𝑇(𝑛)= 1+

The running time complexity is 𝑂=(n3)

s

# Question 5

## Question 5 (i)

function enchantedForestGame(n):

adjacencyMatrix = initializeMatrix(n)

winners = set()

meetingPairs = set()

function meet(i, j):

adjacencyMatrix[i][j] = 1

adjacencyMatrix[j][i] = 1

meetingPairs.add((i, j))

function findWinner():

for each row in adjacencyMatrix:

if all elements in the row = 1 and the corresponding player is != in winners:

winners.add(player)

if winner.length== n

return player

function initializeMatrix(length):

matrix = empty n x n matrix filled with zeros

return matrix

In this algortihm I had decided to use Adjacency Matrix data structure to for this multiplayer game. The meet function is the adjacency matrix and meetingPairs will be updated when the two players meet. In the adjacency matrix, it sets the link between players I and J to 1. It also adds the pair (i, j) to the meetingPairs set.

The finding Winner function, checks if a player has met every other player by iterating over every row in the adjacency matrix. The player will then be added to the winner set if a row with all elements equal to one is not already in the winner set.Then the function will then check if the winner set is acutally equal to n, meaning that that player have met every player.

The Matrix intialization function, creates an empty set of ‘n × n’ matixs that is full of zero. It serves to configure the adjacency matrix's initial state.

## Question 5 (ii)

function Game(int n):

|  |
| --- |
| 𝑛2 |
| 1 |
| 1 |
|  |
|  |
| n |
| n |
| n |
|  |
|  |
|  |
| n |
| N\*n |
| a |
| N\*n |
| b |
|  |
|  |
|  |
| 𝑛2 |
| c |

adjacencyMatrix = emptyMatrix (n)

winners = set()

meetingPairs = set()

function meet(i, j):

adjacencyMatrix[i][j] = 1

adjacencyMatrix[j][i] = 1

meetingPairs.add((i, j))

function findWinner():

for each row in adjacencyMatrix:

if all elements in the row = 1 and the corresponding player is != in winners:

winners.add(player)

if winner.length== n

return player

function emptyMatrix(length):

matrix = empty n x n matrix fulled with zeros

return matrix

Where:  
 a, c= 0 or 1 (best case is 1, worst case is 0)

c= 0 or 1(**best case is 0**, **worst case is 1)**

**Worst Case Scenario**

T(n) = 4 𝑛2 + 4n+2+a+b+c

=4 𝑛2 + 4n+4

Ans: The overall run time complexity is 0( 𝑛2)