Text

Description automatically generated

SEMESTER 1, 2021/2022

CSCI 3302 DSA II Section 03

DATA STRUCTURES AND ALGORITHMS II

**Assignment 1**

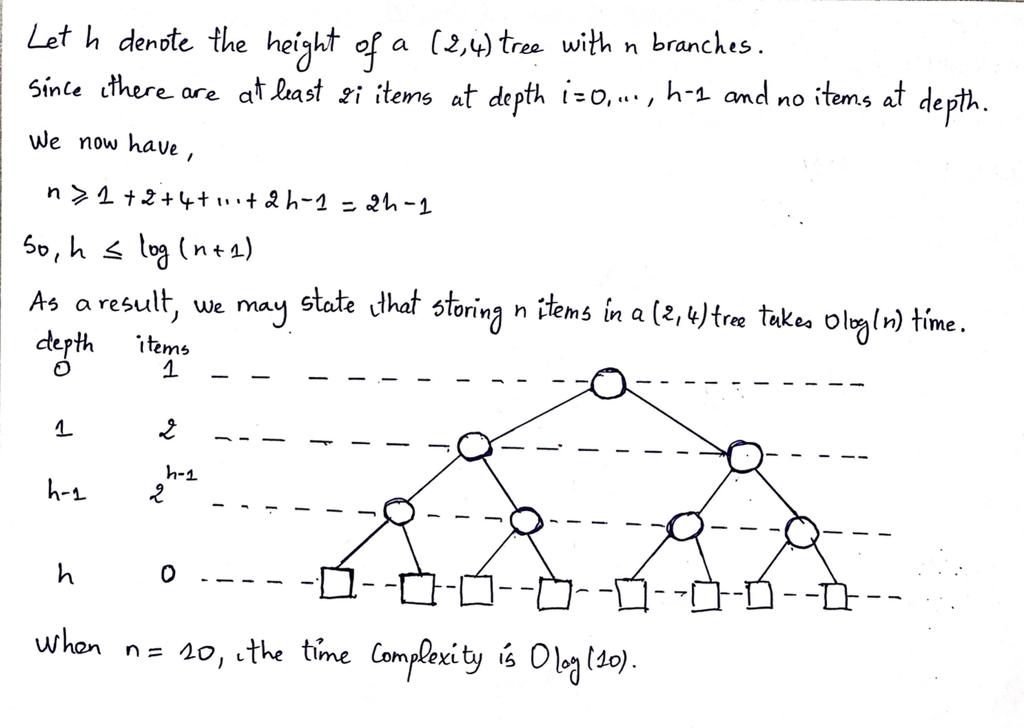
**PREPARED BY:**

|  |  |
| --- | --- |
| **NAME** | **MATRIC NO.** |
| MOHAMED MOUBARAK MOHAMED MISBAHOU MKOUBOI | 1820705 |

LECTURER:

Dr. ASMARANI BINTI AHMAD PUZI

**PART A**

1. Proof that (2,4) tree storing n keys has height Ο (log n). What is the time complexity in this case if n=10? **(4 marks)**
2. What is the property of red black trees? **(5 marks)**

Red-black trees have the following characteristics:

* Each node of the tree is either red or black.
* The tree's root node is always black.
* Each path from the root to a leaf node must contain the same amount of black nodes.
* There can be no two adjacent red nodes, which means that a red node cannot be the parent or child of another red node.

1. What are the criteria you need to consider when choosing the suitable sorting algorithm for your programming? **(5 marks)**

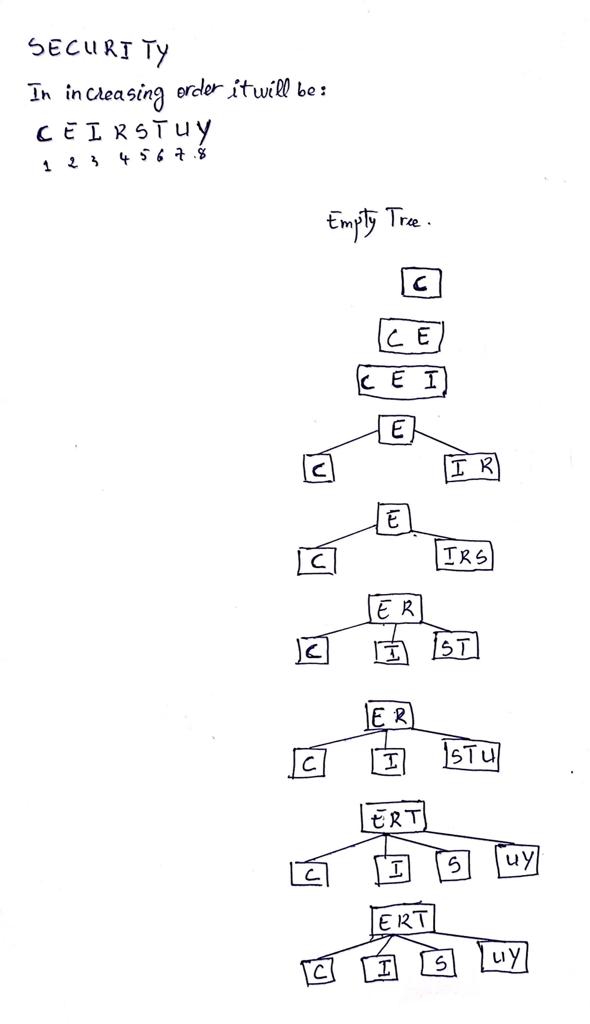
When selecting an appropriate sorting algorithm for programming, two criteria are considered: time complexity and space complexity.

1. Differentiate between merge sort and bubble sort? Please list down minimum of 2 differentiation. **(4 marks)**

|  |  |
| --- | --- |
| **Merge Sort** | **Bubble Sort** |
| * It's a divide-and-conquer algorithm. | * A brute force algorithm is used. |
| * O(n logn) is the time complexity. | * O(n2) is the time complexity. |
| * There is no in-place algorithm. | * There is in-place algorithm. |
| * Extra space is required for sorting elements in a temporary array. | * There is no need for extra space. |

**PART B**

* 1. Construct a 2-4 tree for the list S, E, C, U, R, I, T, Y. Use the alphabetical order of the letters and insert them successively starting with the empty tree. **(8 marks)**



* 1. Assuming that the probabilities of searching for each of the keys (i.e., the letters) are the same, find the largest number of key comparisons for successful searches in this tree. **(2 marks)**

Explore the Left-sub-tree if the key is smaller than the current element.

Otherwise, if the key is greater than the current element, look for the right element in the same node and set it as the current element.

Otherwise, look into the Left-sub-tree if the key is smaller than the current element.

Otherwise, investigate right-subtree if the key is bigger than the current element.

Otherwise, the current element will be returned.

Due to the fact that the 2-4 tree level is 2. And if we want to find Y, we'll have to go through two stages. As a result, the tree's highest number of key comparisons for successful searches is 2.

* 1. In the language of your choice (preferably using PHYTON), Implement MERGE SORT algorithm to sort the numbers of 10, 59, 8, 24, 41, 34, 99 in ascending order. **(8 marks)**

Code:

# Python program for MergeSort implementation

def mergeSort(arr):

if len(arr) > 1:

# Finding the mid of the array

mid = len(arr)//2

# Dividing the array elements

L = arr[:mid]

# divided into two parts

R = arr[mid:]

# Sorting the first half

mergeSort(L)

# Sorting the second half

mergeSort(R)

i = j = k = 0

# Copy data to temp arrays L[] and R[]

while i < len(L) and j < len(R):

if L[i] < R[j]:

arr[k] = L[i]

i += 1

else:

arr[k] = R[j]

j += 1

k += 1

# Checking if any element was left

while i < len(L):

arr[k] = L[i]

i += 1

k += 1

while j < len(R):

arr[k] = R[j]

j += 1

k += 1

# Printing code for the list

def printList(arr):

for i in range(len(arr)):

print(arr[i], end=" ")

print()

# Driver Code

if \_\_name\_\_ == '\_\_main\_\_':

arr = [10, 59, 8, 24, 41, 34, 99]

print("Given array is: ", end="\n")

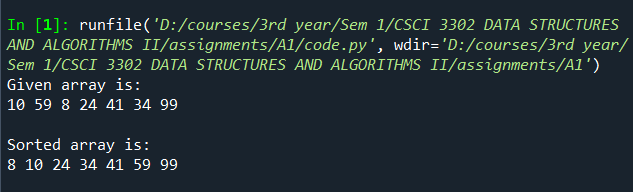
printList(arr)

mergeSort(arr)

print("\nMerge Sorted array is: ", end="\n")

printList(arr)

Output:



* 1. In the language of your choice (preferably using PHYTON), How you can improve the time efficiency when you can trade off an extra memory? **(8 marks)**

Code:

def countSort(myArray):

largestElement = int(max(myArray))

smallestElement = int(min(myArray))

elementRange = largestElement - smallestElement + 1

countArray = [0] \* elementRange

displayArray = [0] \* len(myArray)

# count occurange of each element

for i in range(0, len(myArray)):

countArray[myArray[i]-smallestElement] += 1

# Adding two elements

for i in range(1, len(countArray)):

countArray[i] += countArray[i-1]

# Assign each element to correct index

for i in range(len(myArray)-1, -1, -1):

displayArray[countArray[myArray[i] - smallestElement] - 1] = myArray[i]

countArray[myArray[i] - smallestElement] -= 1

return displayArray

countSortArray = [10, 59, 8, 24, 41, 34, 99]

print("Given array is: ", end="\n")

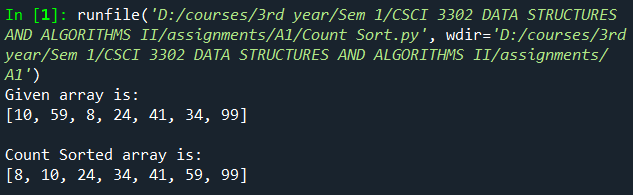
print(countSortArray)

countSortArray = countSort(countSortArray)

print("\nCount Sorted array is: ", end="\n")

print(countSortArray)

Output:

****

* 1. Investigates the time complexity for both algorithms. Which one is consider as better algorithm in terms of time complexity? **(4 marks)**

The time complexity of MERGE SORT is O(nLog(n)) while for COUNTING SORT is O(n). So we can conclude base on the time complexity of both algorithms that MERGE SORT is the fastest one and it is consider as better algorithm in terms of time complexity.