Kandidat: 1103
Dato: 16.05.2023

Question 1

Question 1.1

Information I have access to:

Index = 1704

baseAddress = 1022

elementByteSize = 2

Formula for operation:

A[index] = baseAddress + (index * elementByteSize)

Calculation

A[index] = 1022 + (1704 * 2)

A[index] = 1022 + 3408

The address is:

A[index] = 4430.

Question 1.2

1. Write a function called 'findLargestNumber' that accepts an integer array as its single parameter.

2. Initialize a variable to hold the value of the first index in the array.

3. I need to iterate through the array by setting the initial value of i to 1 and the condition for the loop to continue as long as i is less than the length of the arrayWithNumber.

4. Check and compare if the largestNumberFound is less than the value of arrayWithNumbers[i].

5. Then return if we find the largestNumber.

Question 2

Question 2.1

1. Initially, I need to utilize the predefined Stack class, which accepts integers as an example.
2. Create an "isFullStack" function that accepts two arguments: "stack" of type Stack<> and "maxSize". This function will determine whether the size of the stack is greater than or equal to the maxSize. If this condition is true, the function will return without performing any further actions.
3. Create a push function that adds elements to a stack and includes an isFullStack function. This function should accept a parameter called stack and "maxSize," which will allow you to

specify the maximum size of the stack. By doing so, you can have greater control over the size of your stack.

4. Create an isEmptyStack function to check for underflow, which accepts a Stack<> parameter. Initialize an integer variable called stackSize to 0, and return true if stack.size is equal to stackSize.
5. Create a function that performs two tasks: popping an element from a stack and checking if the stack is empty. To do this, place the implementation of the isEmptyStack function inside an if statement. If the isEmptyStack function returns True, print an "underflow" message and return -1.

Question 2.2

Explanation of code

My code follows an approach that doesn't depend on the size of the stack. Instead, it first checks if the stack is full before pushing a new element. Then, it checks if the new element is the new minimum value and, if so, pushes it to the stackForKeepingTrackOnMinValues. These operations take constant time O(1), meaning that the time it takes to execute them doesn't depend on the size of the stack.

Question 3

Question 3.1

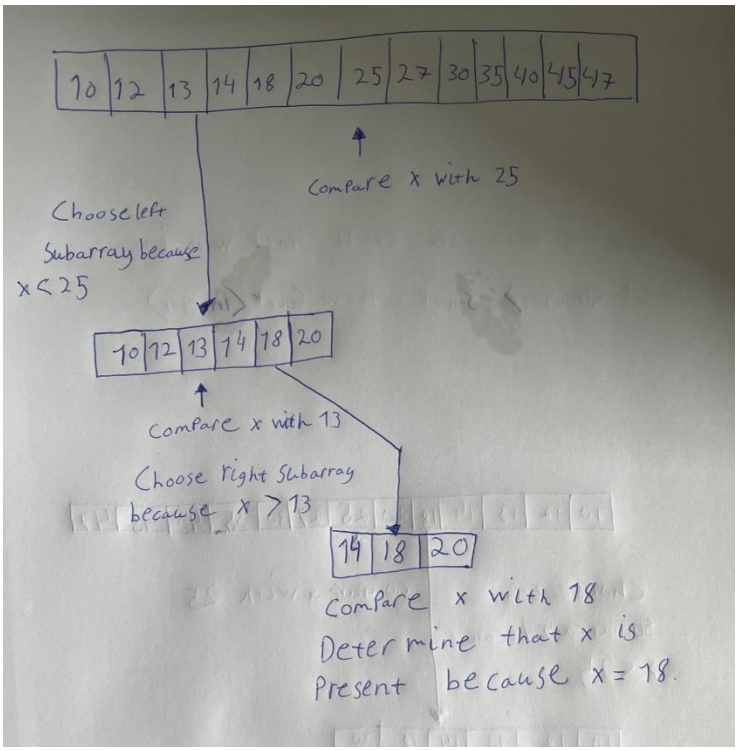
Algorithm for binary search

1. Before we Create a function int binarySearch that takes in four parameters int array and int key int "leftSide" and int "rightSide" we need to make sure that the array is in a non-decreasing order.
2. We can use a while loop with a condition that checks if the value of "leftSide" is less than or equal to the value of "rightSide".
3. First, determine the middle index of the array with the formula mid = (leftSide + rightSide) divided by 2. Then check if array[mid] is equal to key if so, then return mid because we found the key.
4. To determine the correct subarray in which to search for the key, first compare it to the value of the middle element in the array. If the key is less than the middle element, recursively call the binarySearch function with the "leftSide" and mid - 1 parameters; otherwise, call the function with mid + 1 and "rightSide" parameters to search in the "rightSide" subarray, since elements from mid + 1 onwards belong to the right side of the array.

Step by step human representation

binary search using x = 18 and array consisting of numbers like 10,
12,13,14,18,20,25,27,30,35,40,45,47

Kandidat: 1103
Dato: 16.05.2023



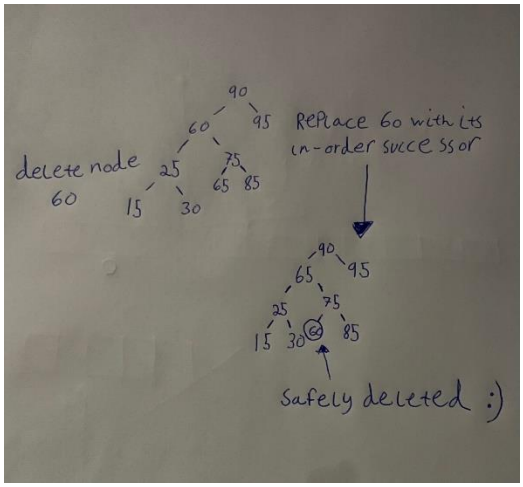
Question 3.2

Algorithm for binary trees

1. Create a function called "deleteNode" that takes in two parameters: "value" and "root". Inside this function, use the "search" function to search for the node containing the specified value within the tree rooted at "root". Assign the resulting node to a variable called "nodeToDelete". If the value is not found in the tree, "nodeToDelete" will be null.
2. Verify whether the function twochild(x, root) returns true when the node with the value x has two children, and false otherwise.
3. If the value (x) has two children, it is necessary to replace x with its in-order successor.
4. First, obtain the parent value (x) using the function getparent(x, root), assigning the result to (p). Next, retrieve the inorder successor, denoted as (i), using the function getIs(x,root), which should return the node that immediately follows (x) in the tree rooted at (root).In order to determine whether the value (x) is located on the left or right side of the tree, we must check if (x.data) is equal to (p.left.data). If this condition is true, then (x) is the left sub-child of (p). In such a case, we need to set the in-order successor to the left sub-child of (p), which means that (p.left) should be set equal to (i).
5. To complete the task at hand, we need to assign the children of the value (x) to the children of (i). This involves setting (i.left) to (x.left) and (i.right) to (x.root). Once this is done, we can return the result.
6. To proceed, we need to verify whether the value (x) corresponds to the right sub-child of (p). If (p.right.data) equals (x.data), we should assign (i) to (p.right). Next, we repeat the following steps: (i) assign (x.left) to (i.left), and (ii) assign (x.right) to (i.right). Once these steps are completed, we can return.

7. To delete the node 60, you should first check if it exists. If it does, compare its value with the root node. If 60 is less than the root, you should explore the left side of the tree.
8. To delete a node 60 from the binary tree, we first need to check if it has any children. In this case, node 60 has two children. To find its inorder successor, we start traversing the left subtree of node 60, as the inorder traversal starts with the inner left side of the tree. So, we visit the nodes in the following order: 15, 25, 30, 60, 65, 75, 85, 90, 95. We can see that the inorder successor of 60 is 65. To delete node 60, we replace it with 65, which will take its place in the tree, and then we delete the leaf node that previously held 65.

Step by step graphical solution



Question 4

Question 4.1

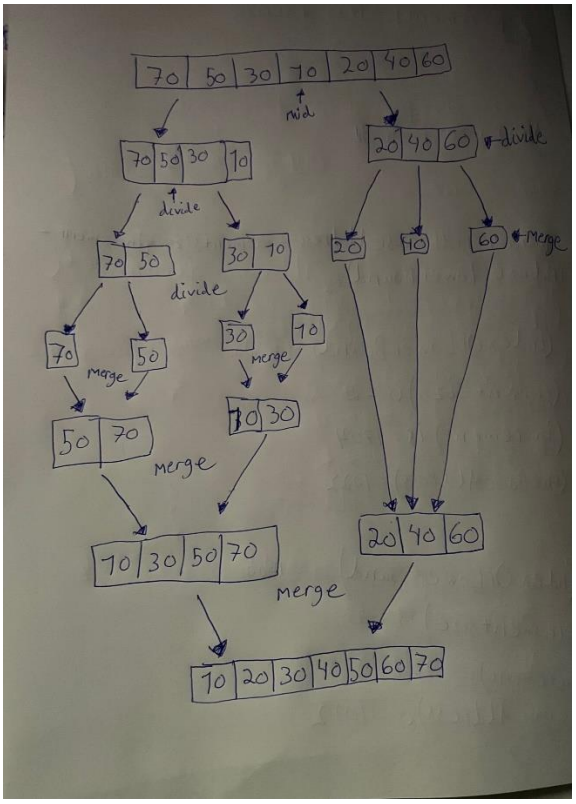
Algorithm for merge sort

1. Define a function named "mergeSort" that accepts a single argument, which is an array of integers. This function will implement the Merge Sort algorithm to sort the input array.
2. To ensure efficient storage and easy access to the array's length, we should assign it to an integer variable called "input length".
3. To prevent the division of subarrays when only one or zero elements remain, we must verify whether the input length variable is less than 2.
4. Create an integer variable called "midIndex" to hold the midpoint value by setting it equal to half of the input length.
5. Create two arrays of integers: one for the left half and one for the right half. To populate the left half array, assign its elements from index 0 up to (and including) the midpoint of the input array. To populate the right half array, assign its elements from the index after the midpoint of the input array up to the last index of the input array. This method works well for arrays with an odd number of elements.
6. To assign values to the left half of an array, create a loop with an integer variable i initialized to 0 and a loop condition of i < mid index. Inside the loop, increment i by 1, and assign the i-th element of the array to the left half.

Kandidat: 1103
Dato: 16.05.2023

7. To assign values to the right half of the array, you can create a loop that starts from the middle index (mid) and iterates until it reaches the end of the array (input length). Within this loop, you can increment the value of the loop counter (i) after each iteration. Finally, you can assign the values to the right half of the array by setting the corresponding array index (i-mid) to the desired value.
8. Next, recursively call mergesort to sort the left half of the array.
9. Next, recursively call mergesort to sort the right half of the array.
10. Create the final function for the complete merge, which accepts three parameters: the input array, the left half array, and the right half array.
11. Define two integers, 'left_size' and 'right_size', which correspond to the lengths of the left and right halves, respectively. Also, initialize three new integer variables 'i', 'j', and 'k' to zero.
12. Write a loop that checks whether the values of (i) and (j) are both less than the size of their respective arrays, left and right.
13. To implement the merge sort algorithm, compare the values in the left half of the input array at index i with the values in the right half at index j. If the value in the left half is less than or equal to the value in the right half, assign the value at index k in the input array to the value in the left half at index i, then increment i. Otherwise, if the value in the right half is less than or equal to the value in the left half, assign the value at index k to the value in the right half at index j, then increment j.
14. Create a loop that checks for any remaining elements on either the left or right side of the array. If there are remaining elements and the index 'i' is less than the size of the left side, then set the element at index 'k' of the input array to be equal to the element at index 'i' of the left half, and increment 'i' and 'k'. Do the same for the right side.

<u>Step by step graphical solution</u>



Kandidat: 1103
Dato: 16.05.2023

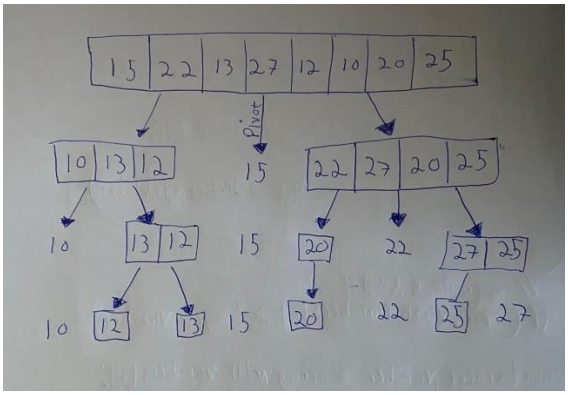
K	U	V	S (Result)
1	10 30 50 70	20 40 60	10
2	10 30 50 70	20 40 60	10 20
3	10 30 50 70	20 40 60	10 20 30
4	10 30 50 70	20 40 60	10 20 30 40
5	10 30 50 70	20 40 60	10 20 30 40 50
6	10 30 50 70	20 40 60	10 20 30 40 50 60
7	10 30 50 70	20 40 60	10 20 30 40 50 60 70

Question 4.2

Algorithm for quick sort

-

Graphical representation



Question 5

Question 5.1

justifying which algorithm is better

In my opinion, the recursive variant of the algorithm is excellent because of its ease of understanding and adherence to the same mathematical definition. On the other hand, the iterative variant is advantageous for its ability to handle large inputs quickly during execution. When dealing with large input data, it is advisable to opt for an iterative approach instead so I would go for that.

Question 5.2

Algorithm for DPF and BFS

-

Graphical representation

Kandidat: 1103
Dato: 16.05.2023

-

Question 6

Question 6.1

A complexity class consists of problems that have been partially solved, while others still require further solutions to be discovered. There are several complexity classes, including P, NP, co-NP, NP-hard, and NP-complete.

Question 6.2

The big O notation is used to analyze algorithms in terms of their runtime or space requirement as the input size grows. For instance, if an algorithm involves three nested loops, its time complexity would increase significantly, resulting in a notation of O(n3), which is different from a single loop operation with a notation of O(n).

Question 6.3

The complexity class P, short for polynomial time, deals with decision problems that have a yes or no answer. This class includes problems that are easy to find solutions for and are typically straightforward to solve. However, it also encompasses problems that have a slightly higher level of complexity, such as those with O(nlogn) time complexity when sorting or O(logn) for searching.

Question 6.4

The NP class refers to a set of decision problems that can be solved by a non-deterministic machine in polynomial time. The word "NP" stands for non-deterministic polynomial time. Suppose there's a school with 1000 students, each of whom wears a unique uniform. The school has 200 rooms available, but the administration has found it challenging to assign roommates when students have different uniform colors. This scenario represents an example of an NP-class problem.

Question 6.5

The NP-complete class is significant because problems in the NP class can be transformed or reduced to NP-complete problems in polynomial time. Therefore, if we can solve an NP-complete problem, we can also solve NP problems in polynomial time.

Sources

QuickSort

LO 3_SortingAlgorithms_PG4200.pptx (page 13)

Kandidat: 1103
Dato: 16.05.2023

P ,NP,NP-complete Class

LO 6_Computability and Complexity_PG4200.pptx (page 5) for P class.

LO 6_Computability and Complexity_PG4200.pptx (page 6) for NP class.

LO 6_Computability and Complexity_PG4200.pptx (page 9) for NP- complete class.

Big O Notation

LO 2_Lo 5_SearchingAlgorithms_Algorithm Efficiency_PG4200.pptx (page 3)

Complexity class

LO 6_Computability and Complexity_PG4200.pptx (page 3-4)

Fibonacci series iterative approach and recursive

https://www.geeksforgeeks.org/different-ways-to-print-fibonacci-series-in-java/

MergeSort algorithm and graphical representation

LO 3_SortingAlgorithms_PG4200.pptx (page 9 -10)

Binary search Tree deletion algorithm and human representation

LO 2_Lo 5_SearchingAlgorithms_Algorithm Efficiency_PG4200.pptx (page 24)

Binary search algorithm and human representation

LO 2_Lo 5_SearchingAlgorithms_Algorithm Efficiency_PG4200.pptx (page 12)