COMP3121 Assignment 1

# **Q1**

**1.1**

Consider an algorithm which runs in time and checks whether a user’s popularity is at least .

Consider the array , which is an integer array organised in descending order of length . To determine whether the user’s popularity is at least , the given popularity index can be used.

# **Q2**

**2.1**

To select a banker to remove while maximising profitability, DAC investments will have to remove the banker with the lowest contribution to the company’s overall profitability.

To determine what is the lowest contribution to overall profitability, DAC can look at each employees’ contribution to overall profitability by computing the overall profitability if each banker were to be removed.

DAC should analyse these values in a way which makes it easy to determine the largest value of overall profitability.

With this information, DAC can remove the banker who resulted in the highest overall profitability once removed because having the highest profitability after being removed shows that their removal least affects the overall profitability, thus maximising profitability.

By doing this, DAC Investments can determine which employee to remove which will maximise profitability.

--

The calculation of the profitability of DAC is given by the total sum of multiplied by the smallest value of .

This is otherwise represented as:

Given the integer arrays and , we can select the banker which has the lowest contribution to the company’s overall profitability as the term in both and . Removing Banker will result in a profitability calculation:

Remove the banker which calculation results in highest overall profitability

As their removal affects overall profitability the least

can iterate through both arrays simultaneously and compute the overall profitability with the given equation if each banker is removed.

**2.2**

Consider the algorithm, , which will select members to be taken from employees that maximises the profitability of DAC Investments. This algorithm must also run in time. **2.3**

Consider the algorithm, , which will select members to be taken from employees that maximises the profitability of DAC Investments. This algorithm must also run in time.

The overall profitability of DAC investments is given by

The time complexity of this equation is .

To maximise profitability by selecting team members, consider selecting team members from employees with the highest performance ratings and the highest investment amounts. To do this, we have to consider prioritising the high-performance employees but at the same time ensure that the total investment amount is high enough to make a significant contribute to overall profitability.

**3.1**

Consider an algorithm which will compute the post-order traversal of an binary search tree (BST) given an array of its pre-order traversal, . This algorithm must also run in time.

This algorithm will begin by initialising an empty stack (*stack*) and an empty array for the post-order traversal (*postArray*). A stack would be suitable because of its last-in-first-out properties which will allow the algorithm to effectively keep track of the order and path of the pre-order traversal.

First, push the first element of (the root node of BST) onto the stack. This is done as the first element of a pre-order traversal is always the root node and is always the last element of a post-order traversal.

Then, for the remaining elements in , iterate through the array and for each element and have a comparative operation which checks these conditions:

* If the element is **lesser** than the value on the top of the *stack*, push it onto *stack* and **go to the next iteration.**
* If the element is **greater** than value on the top of the *stack*, pop that *stack* value and append it to *postArray*. Additionally, push the element onto the *stack*
  + This is done because an element greater than the top of the stack must mean that it is right child of a value in the *stack* that has yet to be added to *postArray*.

Overall, this process involves popping elements from the stack until the parent of the current element is found and append each popped element into *postArray*.

Once all the remaining elements of have been iterated through, pop all the remaining elements from *stack* and append them to *postArray* in reverse order.

This should result in *postArray* which holds the post-order traversal of the BST. This algorithm will take time as each node is pushed onto *stack* exactly once and popped at most once.

**3.2**