z5416824 Jaden Khuu

COMP3121 Assignment 1

Q1

1.1

Consider an algorithm which runs in $O(1)$ time and checks whether a user's popularity is at least p.

For this algorithm, it is given the array V which is an integer array organised in descending order of length n. An integer p is also provided with the condition that $p \leq n$.

The popularity of a user is the largest value of p such that the user has at least p posts which each have at least p thousand views.

This algorithm will search the array value $V[p-1]$ (Note: $p-1$ is chosen as array indexes begin counting from 0) and check if the value of $V[p-1]$ is greater than or equal to p.

If $\boldsymbol{V[p-1]}$ is greater than or equal to $\boldsymbol{p}$, the popularity of the user **is at least $\boldsymbol{p}$,** thus the algorithm should return true**.**

This holds true as the comparison shows that the $(p-1)th$ post will have at least p thousand views. Additionally, the previous posts (array values with index less than $p-1$) will also have at least p thousand views as the array is sorted in descending order of views.

However, **if $\boldsymbol{V[p-1]}$ is less than $\boldsymbol{p}$**, the comparison shows that the $(p-1)th$ post has less than p thousand views, thus the user's popularity **is not at least $\boldsymbol{p}$,** thus the algorithm should return false.

This algorithm runs in $O(1)$ time as the time complexity of checking an index in an array is $O(1)$ and performing the comparison operation is $O(1)$. The total time complexity is of this algorithm is $O(1)$.

1.2

Consider an algorithm which runs in $O(\log n)$ time and computes p for a given user.

For this algorithm, it is given array V which is an integer array organised in descending order of length n. This array represents a user's posts and their views per post.

This algorithm will perform a binary search on the array V index values $[0, n-1]$.

With each iteration of the binary search, the algorithm will use a pointer to indicate the lower and upper bounds of the search interval. In this case, the starting values of these are, $l = 0$ and $u = n-1$. The midpoint index of the array, x, can be found by summing and lower and upper bounds and dividing by two.

For each iteration of the binary search, the algorithm will run the algorithm from question **1.1** with an input of $V[x]$ to check if the popularity of the user is at least p.

With each iteration, the values of the lower and upper bound will be modified, providing a new midpoint every iteration depending on the result of the **1.1** algorithm (**1.1** return true if it is **at least $\boldsymbol{p}$** and false if it is not)

To begin, initialise a variable, *max = 0*, which will be used to compare each value of valid p after the binary search concludes and update whenever a higher value is found. Then, the algorithm will run as follows:

1. Find the midpoint, x, of the current search interval by computing $\frac{l+u}{2}$.
2. Run the **1.1** algorithm with the value $V[x]$ as p.
   a. If the **1.1** algorithm returns true, the algorithm will show that the value of $V[x]$ and the elements of the array before $V[x]$ is a valid value of p. Thus, the variable, *max*, is updated to *max* = $V[x]$.
      i. However, this will not ensure that it is the largest value of p which this user can have. Thus, to continue the binary search, the search interval must be updated to the right most half of the array which will be done by updating the lower-bound to $x + 1$.
      ii. The algorithm will move to the next iteration, where the search interval is now from $l = x + 1$ to u.
   b. However, if the **1.1** algorithm returns false, the algorithm will show that the value of $V[x]$ is not a valid value to be p.
      i. Due to this, the algorithm should continue the binary search in the left most half of the array as the array is in descending order and value of p should exist further up the array. Update the upper-bound, $u = x - 1$.
      ii. The algorithm will move to the next iteration, where the search interval is now from l to $u = x - 1$.
3. Repeat from step 1 with the next search interval until the search intervals cannot be reduced any smaller.

When the binary search is unable to iterate further, the variable *max* will result in the highest possible value for p for the given user array.

The time complexities of the operations of the **1.1** algorithm and the search interval updates both have a time complexity of $O(1)$ which are less dominant than the time complexity of a binary search, $O(\log n)$ where n is the number of distinct indexes in the array. Thus, the algorithm will have an overall time complexity $O(\log n)$.