Number of Assignment : 01

Course Tittle : Data Structure

Course Code : ICT-2101

Submission Date : 23-02-2021

Submitted To

Dr.Mohammad Abu Yousuf

Professor

IIT-JU

Submitted By

Nahidul Islam

Roll-2028

2nd year 1st Semester

IIT-JU

Question 1 : Find the sum of k highest values in the given array

Solve :

We should use **Mergesort** algorithm for sorting the array in descending order.

Then if we calculate the sum of the first k numbers from the array , we will get our answer.

Thus we can solve this problem in **O(n log n)** time.

Data Structure : Array

Algorithom :

1. Enter the array size.
2. Enter the element of array.
3. Sort array.
4. Enter the value of k
5. Sum from last element while k!=0.decrase k one by one.
6. Print sum.

Marge – Sort :

If r > l

1. Find the middle point to divide the array into two halves:

   middle m = l+ (r-l)/2

2. Call mergeSort for first half:

   Call mergeSort(arr, l, m)

3. Call mergeSort for second half:

Call mergeSort(arr, m+1, r)

4.Merge the two halves sorted in step 2 and 3:
Call merge(arr, l, m, r)

Complexity: O(nlogn)

Question 02:

Design a data structure for storing a set of integers. It should support the following operations:

- new(): create a new, empty set
- insert(x): add an integer x to the set
- member(x): test if a given integer x is in the set
- increaseBy(x): add x to all the integers in the set

Solution:

We can use an AVL tree for this problem. Because we know that, AVL tree takes O(log n) time for both inserting and searching a member from the set.

If we solve this from by using AVL tree , we will get O(1) time for new function, O (log n) for insert and member function and O(n) time for increaseBy function.

Pseudocode:

Here we should use some function for operating those functions. First of all we will create a structure named Node which will contain left node address, right node address and height for each parent node.

For calculating Height of the AVL tree we will use Height function. This function will return an integer data type.

Height (*N)

1. if N == NULL

   return 0

2. return N->height and exit.

For doing right rotation, we will use RightRotate Function. This function will return a Node data type pointer variable.

RightRotate (*y)

1. assign two Node data type pointer variable *x and *T2
2. *x = y->left
3. *T2 = x->right
4. x->right = y
5. y->left = T2
6. y->height = max(height(y->left), height(y->right)) + 1
7. x->height = max(height(x->left), height(x->right)) + 1
8. return x and exit.

For doing left rotation, we will use LeftRotate Function. This function will return a Node data type pointer variable.

LeftRotate (*x)

1. assign two Node data type pointer variable *y and *T2
2. *y = x->right
3. *T2 = y->left
4. y->left = x

```
5. x->right = T2
6. x->height = max(height(x->left), height(x->right))
   +1
7. y->height = max(height(y->left), height(y->right))
   +1
8. return y and exit.
```

For getting balancing factor, we will use GetBalance function which will return an integer data.

GetBalance (*N)

```
1. if N == NULL
       return 0
2. return height(N->left) - height(N->right) and exit.
```

Those above function will be used by **New(), Insert(), Member()** and **IncreaseBy()** funcitons.

Here I am giving the pseudocode for those four Functions:

New (key)

```
1. Assign a Node data type pointer variable node.
2. *node = new Node()
3. node->key = key
4. node->left = NULL
5. node->right = NULL
6. node->height = 1
7. return node and exit.
```

This New ( ) function will take O(1) time which is satisfying the above constraints.

Insert (node, key)

```
1. if node == NULL
      return newNode(key) and exit
2. if key < node->key
      node->left = insert(node->left, key)
3. else if key > node->key
      node->right = insert(node->right, key)
4. else
      return node
   [end of if else statements]
5. node->height = 1 + max(height(node->left), height(node
   >right))
6. set balance = getBalance(node)
7. if balance > 1 and key < node->left->key
        return rightRotate(node)
   [end of if structure]
8. if balance < -1 and key > node->right->key
        return leftRotate(node)
   [end of if structure]
9. if balance > 1 and key > node->left->key
        node->left = leftRotate(node->left)
        return rightRotate(node)
   [end of if structure]
10.     if balance < -1 and key < node->right->key
```

```
            node->right = rightRotate(node->right)
            return leftRotate(node)
   [end of if structure]
11.    return node and exit.
```

This Insert ( ) function will take O(log n) time which is satisfying the above constraints.

Member (head, x)

```
1. if head == NULL
       print out "This element is not found!"
       exit
   [end of if]
2. else
3.     while head != 0
4.         if x > head->key
5.             head = head->right
6.         else if x < head->key
7.             head = head->left
8.         else
9.             print out "This element is found!"
10.            return
11.        [end of else statement]
12.    [end of while loop]
13.    print out "This element is not found"
14.[end of else statement]
```

This Member ( ) function will take O(log n) time which is satisfying the above constraints.

IncreaseBy (head, x)

```
1. if head != 0
        increaseBy(head->left,
        x)   head->key=   head->
        key          +          x
        increaseBy(head-
        >right,x)
2. exit
```

This IncreaseBy ( ) function will take O(n) time which is satisfying the above constraints.

```
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

Question 2 : Design a data structure for sorting a set of integers

Algorithom