Name:Vishal Chauhan

PRN:2020BTECS00090

Ass no:1

Course: Design and analysis of algorithm Lab

Batch:T6 Class:TYCSE,WCE

Q1) You are given two sorted array, A and B, where A has a large enough buffer at the end to hold B. Write a method to merge B into A in sorted order.

```
ALgorithm
1.take input array A,B of size(n+m) and m
2.traverse array A and B simaltaneously
        i.if(A[i]<=b[j]) then increase i to i+1
        ii.else swap A and B element then Sort B so that at B[j=0],it should have
smallest nummber among B array
3.merge B to A
4.print A
```

```
#include <iostream>
#include <vector>
#include<algorithm>
using namespace std;
void swap(int &a,int &b)
{
    int temp=a;
    a=b;
    b=temp;
}

void display_arr(vector<int> arr)
{
    for(int i=0;i<arr.size();i++)
    cout<<arr[i]<<" ";
}
int main()
{
    int n, m;
    cout<<"Enter the size of A and B:"<<endl;
```

```
    cin >> n >> m;

    vector<int> A(n + m, 0),B(m);
    for (int i = 0; i < n; i++)
    {
        cin >> A[i];
    }
    for (int i = 0; i < m; i++)
    {
        cin>>B[i];
    }

// -->as given is that array is alrealy sorted-->so no use of sorting again
    // sort(A.begin(),A.end());
    // sort(B.begin(),B.end());
    // display_arr(A);cout<<endl;

    int i=0;
    int j=0;
    while(i<n && j<m)
    {
        if(A[i]<=B[j])
        {
            i++;
        }
        else
        {
            swap(A[i],B[j]);
            sort(B.begin(),B.end());
        }
    }
    for(int i=n;i<n+m;i++)
    A[i]=B[i-n];

    cout<<"A merged with B array:";
    display_arr(A);
}
```

```
OUTPUT:
Enter the size of A and B:
5 3
10 30 50 70 90
20 40 60
```

```
A merged with B array:10 20 30 40 50 60 70 90
```

Complexity analysis→

```
time complexity--> O(n)*nlog(m) traversing and for sorting at worst case--
>overall O(n)*nlog(m)

space complexity-->O(1) as used constant  space only
```

Q2) Write a method to sort an array of string so that all the anagrams are next to each other.

```
#include <bits/stdc++.h>
using namespace std;

ALGORITHM:-
1.Take the array of strings as an input to the array.
2 Iterate over each string in the array and sort that string and push it into the
unordered map
  as a key and insert the elements which are anagram of it into the value array
of the map.
3.After inserting the anagrams sort the array of anagrams. 4 Display the result.

int main()
{
    int n;
    cout<<"Enter the number of words in the array of string:";
    cin >> n;
    string arr[n];
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    unordered_map<string, vector<string>> umap;
    for (int i = 0; i < n; i++)
    {
        string word = arr[i];
        char letters[word.size() + 1];
        strcpy(letters, word.c_str());
        sort(letters, letters + word.size() + 1);
        string newword = " ";
        for (int j = 0; j < word.size() + 1; j++)
```

```
        {
            newword += letters[j];
        }
        if (umap.find(newword) != umap.end())
        {
            umap[newword].push_back(word);
            sort(umap[newword].begin(), umap[newword].end());
        }
        else
        {
            vector<string> words;
            words.push_back(word);
            umap[newword] = words;
        }
    }

    cout<<"ans:";
    for (auto it = umap.begin(); it != umap.end(); it++)
    {
        auto values = umap[it->first];
        if (values.size() >= 1)
        {
            cout << "[";
            for (int i = 0; i < values.size() - 1; i++)
            {
                cout << values[i] << ",";
            }
            cout << values[values.size() - 1];
            cout << "]";
        }
    }

    return 0;
}
```

```
OUTPUT:
Enter the number of words in the array of string:7
cat tac atc act god odg gdo
ans:[gdo,god,odg][act,atc,cat,tac]
```

```
Complexity Analyse
The time complexity of the above algorithm is O(n*mlogm)
```

```
where n is the number of words in the array and m is the maximum number anagrams
of a word in the array.
```

Q3) Given a sorted array of *n* integers that has been rotated an unknown number of times, write code to find an element in the array. You may assume that the array was originally sorted in increasing order.

EXAMPLE

Input: find 5 in {15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14}

Output: 8 (the index of 5 in the array)

```
Algorithm
1.initilise i=0 ,j=size-1 and find mid=(i+j)/2;
2.traversing until i<j
    i.if search element founded then return it's index
    ii.else if  try to search in sorted array part by checking arr[i]<arr[mid]
and update i/j index
        I. if arr[i]<=Search_ele && Search_ele<=arr[mid-1] then search left side
        II.else search right side
3.if not found the return -1
```

```
// Q3) Given a sorted array of n integers that has been rotated an unknown number
of times, write code to find an element in the array.
//  You may assume that the array was originally sorted in increasing order.
#include<iostream>
#include<vector>
using namespace std;
// Driver code
int searching(int *arr,int size,int Search_ele)
{
    int i=0;
    int j=size-1;
    while(i<j)
    {
        int  mid=i+(j-i)/2;
        if(arr[mid]==Search_ele)
        return mid;
        else if(arr[mid]>arr[i])
        {
            if(arr[i]<=Search_ele && Search_ele<=arr[mid-1] )
            j=mid-1;
            else
```

```
            i=mid+1;
        }
        else
        {
            if(Search_ele>= arr[mid+1] && Search_ele<=arr[j])
            i=mid+1;
            else
            j=mid-1;
        }
    }
    return -1;
}
int main()
{
    int arr[] = {15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14};
    int n = sizeof(arr) / sizeof(arr[0]);
    int serachedIndex=searching(arr,n,20);
    cout<<"search index founded:"<<serachedIndex<<endl;
    return 0;
}
```

```
OUTPUT:
PS C:\Users\Dell\OneDrive\Desktop\DAA\Ass-1> cd
"c:\Users\Dell\OneDrive\Desktop\DAA\Ass-1\" ; if ($?) { g++ q3.cpp -o q3 } ; if
($?) { .\q3 }

search index founded:3
```

```
Complexity Analyse:
1>time complexity is log(n) for binary search
2>space complexity is O(1) as only used constant space
```

Q4.) Imagine you have a 20GB file with one string per line. Explain how you would sort the file.

ANS:

For sorting the 20 GB file with one string per line we can use the external merge sort algorithm that is used to sort the large data that does not fit the RAM. Her we break the large data into smaller chunks the

One example of external sorting is the external merge sort algorithm, which is a K-way merge algorithm. It sorts chunks that each fit in RAM, then merges the sorted chunks together

The algorithm first sorts *M* items at a time and puts the sorted lists back into external memory. Itthen recursively does a K-way merge on those sorted lists. To do this merge, *B* elements from each sorted list are loaded into internal memory, and the minimum is repeatedly outputted.

For example, for sorting 20GB of data using only 1 GB of RAM:

1. Read 1 GB of the data in main memory and sort by some conventional method, like quicksort.

2. Write the sorted data to disk.

3. Repeat steps 1 and 2 until all of the data is in sorted 1GB chunks (there are 20GB / 1GB = 20 chunks), which now need to be merged into

one single output file.

4. Read the first 50MB(= 100MB / (20 chunks + 1)) of each sorted chunk into input buffers in main memory and allocate the remaining 50 MB

for an output buffer. (In practice, it might provide better performance to make the output buffer larger and the input buffers slightly smaller.)

5. Perform a 20-way merge and store the result in the output buffer. Whenever the output buffer fills,

write it to the final sorted file and empty it. Whenever any of the 20 input buffers empties, fill it with the next 50 MB of its associated 1GB sorted chunk until no more data from the chunk is available. This is the key step that makes external merge sort work externally—because the merge algorithm only makes one pass sequentially through each of the chunks, each chunk does not have to be loaded completely; rather, sequential parts of the chunk can be loaded as needed.

Q5) Given a sorted array of string which is interspersed with empty string, write a method to find the location of a given string.

EXAMPLE

Input: find “ball” in {“at”, “”, “”, “ball”, “”, “”, “car”, “”, “”, “dad”, “”,””}

```
ALGORITHM:-
The code is doing the following:
1.  It is creating a hashmap with key as the string and value as vector of
indices where that string occurs.
2.  It is taking input from user for the string to be searched.
3.  It is searching the string in the hashmap.
4.  If the string is found, it prints the first index where it occurs.
5.  If the string is not found, it prints -1.
The key is the string and the value is a vector of indices where that string
occurs. We can then use the find() function to check
if the string exists in the map or not. If it does, we can print the first index
of the vector.
```

```
#include <bits/stdc++.h>
using namespace std;


int main()
{
    int N=8;

    string arr[N] = {"at", "", "", "ball", "", "", "car", ""};
    unordered_map<string, vector<int>> umap;
    for (int i = 0; i < N; i++)
    {
        if (umap.find(arr[i]) != umap.end())
        {
            umap[arr[i]].push_back(i);
        }
        else
        {
            vector<int> v;
            v.push_back(i);
            umap[arr[i]] = v;
        }
    }
    string find;
    cout<<"enter string whose index want to know:";
    cin >> find;

    cout<<"index:";
```

```
    cout << umap[find][0] << endl;

    return 0;
}
```

```
OUTPUT:
enter string whose index want to know:car
index:6
```

```
Complexity analyse
1>The time complexity of the above algorithm is O(N) where N is the number of
string words in the array.
2>The space complexity of the above algorithm is O(N) as we are using hashmap to
store the values
```

Q6) Given an M*N matrix in which each row and each column is sorted in ascending order, write a method to find an element.

```
Algorithm
1.  In the below program we will take the input 2d array .
2.  Take input  the element which we need to find out and store it in key
variable.
3.  Start from the first row and last element check if the element is greater
than or equal to the key.
4.If the last element is greater than the key then search the element the row and
print its index
5. Else check the last element of the next row
```

```
// search in sorted matrix which is row and column wise sorted
#include <iostream>
#include <vector>
using namespace std;
vector<int> searching(vector<vector<int>> mat, int i, int j,int search)
{
    // cout<<"searching.\n";
    vector<int> ans(2,-1);
    while (i < mat.size() && j >= 0)
    {
        // cout<<"searching..\n";

        if(mat[i][j]==search)
        {
```

```
            ans[0]=i;
            ans[1]=j;
            return ans;
        }
        else if(mat[i][j]<search)
        i++;
        else if(mat[i][j]>search)
        j--;
    }
    return ans;
}
int main()
{
    vector<vector<int>> mat{
        {1, 2, 3,12,19},
        {4, 5, 6,15,29},
        {7, 8, 9,18,56}};
    // int i = mat.size() - 1;
    int i=0;
    int j = mat[0].size() - 1;
    vector<int> ans = searching(mat, i, j,18);

    cout<<"at pos index:[ "<<ans[0]<<","<<ans[1]<<" ]"<<endl;
}
```

```
OUTPUT:
at pos index:[ 2,3 ]

TIME AND SPACE COMPLEXITY:-

The time complexity of the above algorithm is O(N+M) where N is the number of
rows and M is the number of coulmns in the array.
The space complexity of the above algorithm is O(1)as there is no extra space is
used.
```

Q7) A circus is designing a tower routine consisting of people standing atop one another's shoulders. For practical and aesthetic reasons, each person must be both shorter and lighter than the person below him or her. Given the heights and weight of each circus, write a method to compute the largest possible number of people in such tower.

EXAMPLE:

Input(ht,wt): (65, 100) (70, 150) (56, 90) (75,190) (60, 95) (68, 110).

Output: The longest tower is length 6 and includes from top to bottom:

(56, 90) (60, 95) (65, 100) (68, 110) (70, 150) (75, 190)

```
// ALGORITHM:-
// 1.   First we take the pairs of height and weight in array of pairs
// 2.   We sort the above pairs using insort and then compare the adjacent
heights and weight if any of the pair violates the rules remove that pair.
// 3.   Display the result.

#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cout<<"enter n:";
    cin >> n;
    vector<pair<int, int>> vp(n + 1);
    for (int i = 0; i <= n; i++)
    {
        if (i == 0)
        {
            vp[i] = {0, 0};
            continue;
        }
        cin >> vp[i].first;
        cin >> vp[i].second;
    }

    sort(vp.begin(), vp.end());
    int count = 0;
    for (int i = 1; i <= n; i++)
    {
        if (vp[i].first < vp[i - 1].first)
        {
            continue;
        }
        if (vp[i].second < vp[i - 1].second)
        {
            continue;
        }
        count++;
        // cout << vp[i].first << " " << vp[i].second << endl;
    }
    cout << "The max length of the tower is " << count << endl;
```

```
    return 0;
}

OUTPUT:
enter n:5
40 65
60 75
50 45
90 110
68 99
The max length of the tower is 4
```

```
TIME AND SPACE COMPLEXITY:-
The time complexity for the above algorithm is O(NlogN) where N is the number of
pairs of height and weight.
The space complexity of the above algorithm is O(1) as there is no extra space
used.
```

Q8) Imagine you are reading in stream of integers. Periodically, you wish to be able to look up the rank of number *x* (the number of values less than or equal to *x*). Implement the data structures and algorithms to support these operations. That is, Implement the method *track (int x)*, which is called when each number is generated, and the method *getRankOfNumber (int x)*, which return the number of values less than or equal to *x* (not including x itself).

EXAMPLE

Stream (in order of appearance) : 5, 1, 4, 4, 5, 9, 7, 13, 3

getRankOfNumber(1) = 0

getRankOfNumber(3) = 1

getRankOfNumber(4) =3

```
// ALGORITHM:-
// 1.   Take the all elements as the input in the array.
// 2.   Sort all the elements in the array.
// 3.   Take input the element x whose Rank we need to calculate.
// 4.   Count the number of elements which are less than or equal to x.

#include <bits/stdc++.h>
using namespace std;
int getRankOfNumber(vector<int> &v, int x)
```

```
{
    int count = 0;
    int i = 0;
    while (true)
    {
        if (v[i] <= x && v[i + 1] <= x)
        {
            count++;
        }
        else
        {
            break;
        }
        i++;
    }
    return count;
}

int main()
{
    int N;
    cout<<"enter N:";
    cin >> N;
    vector<int> v(N);
    for (int i = 0; i < N; i++)
    {
        cin >> v[i];
    }
    sort(v.begin(), v.end());

    cout<<"enter which rank u wanted:";
    int getRank;
    cin >> getRank;

    // cout<<"The rank of "<<getRank<<" is "<<getRankOfNumber(v,getRank)<<endl;
    cout << getRankOfNumber(v, getRank) << endl;

    return 0;
}
```

```
OUTPUT:
enter N:6
23 12 45 22 66 52
enter which rank u wanted:45
```

```
3
```

```
TIME AND SPACE COMPLEXITY:-
The time complexity of the above algorithm is O(NlogN)
the space complexity is O(1) as there is no extra space used.
```

****