Walchand College of Engineering, Sangli
Computer Science & Engineering
Third Year

Course: Design and analysis of algorithm Lab

Lab course coordinator:
Mrs A M Chimanna- Batch: - T1, T2,T3,T4

Week 1 Assignment

PRN : 21510111
Batch : T1

Part: 1

Sorting Algorithm

Q) 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.

```
#include<bits/stdc++.h>
using namespace std;
//Time Complexity == O(n)
//Space Complexity == O(1)
int main(){
    int A[10]={1,4,6,7,9};
    int B[5]={2,3,5,8,11};
    int n=5;
    int m=5;
    int temp=INT_MAX;

    int lasInd = 9;
    int i=n-1;
    int j=m-1;
    while(j>=0){
        if(i>=0 && A[i]>=B[j]){
            A[lasInd]=A[i];
            i--;
        }else{
            A[lasInd]=B[j];
            j--;
        }
        lasInd--;
    }
    for(int k=0;k<10;k++){
        cout<<A[k]<<" ";
    }
    return 0;
}
```

Q) 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;

int main(){
    vector<string>
args={"abc","dogg","bcs","ggod","donkey","keydon","bac","oggd"};
    vector<pair<string,int>> ha;
    for(int i=0;i<args.size();i++){
        ha.push_back({args[i],i});
        sort(ha[i].first.begin(),ha[i].first.end());
    }
    sort(ha.begin(),ha.end());
    for(int i=0;i<ha.size();i++){
        cout<<args[ha[i].second]<<endl;
    }
    return 0;
}
```

Q) 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)

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

int main(){
    vector<int> args = {15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14};
    int target = 16;
    int pivot;
    for(int i=0; i<args.size()-1; i++){
        if(args[i]>args[i+1]) pivot = i;
    }
    //cout<<pivot<<endl;
    int l1,r1;

    if(target>args[0] && target<=args[pivot]){
        l1=0;
        r1=pivot;
    }else{
        l1=pivot+1;
        r1=args.size()-1;
    }

    while(l1<=r1){
        int mid = l1+(r1-l1)/2;
        if(target==args[mid]){
            cout<<mid;
            return 0;
        }else if(target<args[mid]){
            r1=mid-1;
        }else{
            l1=mid+1;
        }
    }
    return 0;
}
```

Q) Imagine you have a 20GB file with one string per line. Explain how you would sort the file.
Solution: Divide the file into N chunks which are x megabytes each, where x is the amount of memory we have available. Each chunk is sorted separately and then saved back to the file system. Once all the chunks are sorted, we then merge the chunks according to the following algorithm:

1. Divide your memory into (N+1) parts. First N parts are used to read data from N chunks, the last one is used as a buffer.

2. Load data to fill the first N data parts from N chunks respectively, perform an N-way merge sort to the buffer.

3. While any data part is not empty, perform sort to the buffer.

4. If any data part is empty, load new content from the corresponding chunk.

5. If the buffer is full, write buffer to the disk as output file, clear buffer.

6. Repeat step 4-5 until all N chunks and buffer are empty.

At the end, we have output that is fully sorted on the disk. This algorithm is known as external sort.

One example of external sorting is the external merge sort algorithm, which sorts chunks that each fit in RAM, then merges the sorted chunks together.[1][2] For example, for sorting 900 megabytes of data using only 100 megabytes of RAM:

1. Read 100 MB 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 100 MB chunks (there are 900MB / 100MB = 9 chunks), which now need to be merged into one single output file.

4. Read the first 10 MB (= 100MB / (9 chunks + 1)) of each sorted chunk into input buffers in main memory and allocate the remaining 10 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 9-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 9 input buffers empties, fill it with the next 10 MB of its associated 100 MB 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.

Q) 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”, “”,””}
Output: 4

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

int main(){
    vector<string> test =
{"at","","","ball","","","car","","","dad","",""};
    string toFind = "ball";
    int l=0;
    int h=test.size()-1;
    while(l<=h){
        while(test[l]=="")l++;
        while(test[h]=="")h--;
        int mid = (l+h)/2;
        while(test[mid]=="" && l>mid)mid--;
        if(test[l]==toFind){
            cout<<l<<endl;
            break;
        }
        if(test[h]==toFind){
            cout<<h<<endl;
            break;
        }
        if(test[mid]==toFind){
            cout<<mid<<endl;
            break;
        }
        if(test[mid].compare(toFind)>0){
            l=mid+1;
            continue;
        }else{
            h=mid-1;
            continue;
        }

    }
    return 0;

}
```

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

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

vector<int> findAns(vector<vector<int>> mat, int key){
    int hr = mat.size()-1;
    int lr = 0;
    int hc = mat[0].size()-1;
    int lc = 0;
    int row =-1;
    while(hr>lr){
        int mid = (hr+lr)/2;
        if(key>=mat[mid][0] && key<=mat[mid][hc]){
            row = mid;
            break;
        }
        if(key> mat[mid][hc]){
            lr=mid+1;
        }else if(key < mat[mid][hc]){
            hr=mid;
        }
    }
    cout<<row<<endl;
    while(lc<hc){
        int mid = (lc+hc)/2;
        if(key==mat[row][mid]){
            return {row,mid};
        }
        if(key>mat[row][mid]){
            lc=mid+1;
        }else{
            hc=mid;
        }
    }
    return {-1,-1};
}

int main()
{

    vector<vector<int> > arr = { { 1, 2, 3, 4 },
                                 { 5, 6, 7, 8 },
                                 { 9, 10, 11, 12 } };
    vector<int> ans = findAns(arr, 7 );

    cout << "Element found at index: [";
    for (int i = 0; i < ans.size(); i++) {
        if (i == ans.size() - 1)
            cout << ans[i];
        else
            cout << ans[i] << ", ";
    }
    cout << "]";
}
```

Q) 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)

```
#include <iostream>
#include <vector>

struct TernaryNode {
    size_t height{0};
    size_t weight{0};
    size_t maxHeight{0};

    TernaryNode *left{nullptr};
    TernaryNode *mid{nullptr};
    TernaryNode *right{nullptr};

    bool operator<(const TernaryNode &other) {
        return this->height < other.height &&
               this->weight < other.weight;
    }

    bool operator>(const TernaryNode &other) {
        return this->height > other.height &&
               this->weight > other.weight;
    }
};

TernaryNode* buildTTree(std::vector<std::pair<size_t, size_t> > &staff,
                        size_t index = 0) {

    TernaryNode *node = new TernaryNode();
    node->height = staff[index].first;
    node->weight = staff[index].second;

    if(index == staff.size() - 1) {
        return node;
    }

    TernaryNode *subNode = buildTTree(staff, index + 1);

    if(*subNode < *node) {
        node->left = subNode;
        node->maxHeight = std::max(node->maxHeight, subNode->maxHeight
+ 1);
    } else if(*subNode > *node) {
        node->right = subNode;
        node->maxHeight = std::max(node->maxHeight, subNode->maxHeight
+ 1);
    } else {
        node->mid = subNode;
        node->maxHeight = std::max(node->maxHeight,
subNode->maxHeight);
    }
}

size_t calcMaxTowerHt(std::vector<std::pair<size_t, size_t> > &staff) {
    TernaryNode *node = buildTTree(staff);
    return node->maxHeight;
}

int main()
{
    std::vector<std::pair<size_t, size_t> > circusStaff{{60, 230}, {50,
130}, {56, 240},
                                                        {52, 220}, {57,
200}, {53, 160},
                                                       {58, 210}, {51,
140}, {55, 180},
                                                       {59, 220}};

    std::cout << calcMaxTowerHt(circusStaff);
    return 0;
}
```

Q) 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

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

struct Node {
    int data;
    Node *left, *right;
    int leftSize;
};

Node* newNode(int data)
{
    Node *temp = new Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    temp->leftSize = 0;
    return temp;
}

Node* track(Node*& root, int data)
{
    if (!root)
        return newNode(data);

    if (data <= root->data) {
        root->left = track(root->left, data);
        root->leftSize++;
    }
    else
        root->right = track(root->right, data);
    return root;
}

int getRank(Node* root, int x)
{
    if (root->data == x)
        return root->leftSize;

    if (x < root->data) {
        if (!root->left)
            return -1;
        else
            return getRank(root->left, x);
    }

    else {
        if (!root->right)
            return -1;
        else {
            int rightSize = getRank(root->right, x);
              if(rightSize == -1 ) return -1;
            return root->leftSize + 1 + rightSize;
        }
    }
}

int main()
{
    int arr[] = { 5, 1, 4, 4, 5, 9, 7, 13, 3 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 1;
    Node* root = NULL;
    for (int i = 0; i < n; i++)
        root = track(root, arr[i]);
    cout << "Rank of " << x << " in stream is: "
         << getRank(root, x) << endl;
    x = 3;
    cout << "Rank of " << x << " in stream is: "
         << getRank(root, x) << endl;
    x = 4;
    cout << "Rank of " << x << " in stream is: "
         << getRank(root, x) << endl;
    return 0;
}
```