18/11/24

1)Bubble sort in C++

#include <iostream>

using namespace std;

// perform bubble sort

void bubbleSort(int array[], int size) {

// loop to access each array element

for (int step = 0; step < size -1; ++step) {

// loop to compare array elements

for (int i = 0; i < size - step - 1; ++i) {

// compare two adjacent elements

// change > to < to sort in descending order

if (array[i] > array[i + 1]) {

// swapping elements if elements

// are not in the intended order

int temp = array[i];

array[i] = array[i + 1];

array[i + 1] = temp;

}

}

}

}

// print array

void printArray(int array[], int size) {

for (int i = 0; i < size; ++i) {

cout << " " << array[i];

}

cout << "\n";

}

int main() {

int data[] = {-2, 45, 0, 11, -9};

// find array's length

int size = sizeof(data) / sizeof(data[0]);

bubbleSort(data, size);

cout << "Sorted Array in Ascending Order:\n";

printArray(data, size);

}

2)Quick sort

#include <bits/stdc++.h>

using namespace std;

int partition(vector<int>& arr, int low, int high) {

// Choose the pivot

int pivot = arr[high];

// Index of smaller element and indicates

// the right position of pivot found so far

int i = low - 1;

// Traverse arr[;ow..high] and move all smaller

// elements on left side. Elements from low to

// i are smaller after every iteration

for (int j = low; j <= high - 1; j++) {

if (arr[j] < pivot) {

i++;

swap(arr[i], arr[j]);

}

}

// Move pivot after smaller elements and

// return its position

swap(arr[i + 1], arr[high]);

return i + 1;

}

// The QuickSort function implementation

void quickSort(vector<int>& arr, int low, int high) {

if (low < high) {

// pi is the partition return index of pivot

int pi = partition(arr, low, high);

// Recursion calls for smaller elements

// and greater or equals elements

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main() {

vector<int> arr = {10, 7, 8, 9, 1, 5};

int n = arr.size();

quickSort(arr, 0, n - 1);

for (int i = 0; i < n; i++) {

cout << arr[i] << " ";

}

return 0;

}

3)non repeating character

// C++ program to find the index of the first

// non repeating character using nested loop

#include <iostream>

using namespace std;

// Function to find the index of the first

// non-repeating character

char nonRepeatingChar(string &s) {

int n = s.length();

// Iterate over each character in the string

for (int i = 0; i < n; ++i) {

bool found = false;

// Check if the character repeats in the

// rest of the string

for (int j = 0; j < n; ++j) {

if (i != j && s[i] == s[j]) {

found = true;

break;

}

}

// If character does not repeat, return it

if (found == false)

return s[i];

}

// If no such character is found, return '$'

return '$';

}

int main() {

string s = "racecar";

cout << nonRepeatingChar(s);

return 0;

}

4) Edit distance

#include <bits/stdc++.h>

using namespace std;

// A Naive recursive C++ program to find minimum number

// of operations to convert s1 to s2

int editDistRec(string& s1, string& s2, int m, int n) {

// If first string is empty, the only option is to

// insert all characters of second string into first

if (m == 0) return n;

// If second string is empty, the only option is to

// remove all characters of first string

if (n == 0) return m;

// If last characters of two strings are same, nothing

// much to do. Get the count for

// remaining strings.

if (s1[m - 1] == s2[n - 1])

return editDistRec(s1, s2, m - 1, n - 1);

// If last characters are not same, consider all three

// operations on last character of first string,

// recursively compute minimum cost for all three

// operations and take minimum of three values.

return 1 + min({editDistRec(s1, s2, m, n - 1), // Insert

editDistRec(s1, s2, m - 1, n), // Remove

editDistRec(s1, s2, m - 1, n - 1)}); // Replace

}

// Wrapper function to initiate the recursive calculation

int editDist(string& s1, string& s2) {

return editDistRec(s1, s2, s1.length(), s2.length());

}

// Driver code

int main() {

// your code goes here

string s1 = "GEEXSFRGEEKKS";

string s2 = "GEEKSFORGEEKS";

// Output the result of the edit distance calculation

cout << editDist(s1, s2);

return 0;

}

5) K largest elements

// C++ program to find k largest elements in an

// array using sorting

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

vector<int> kLargest(vector<int> &arr, int k) {

// sort the given array in descending order

sort(arr.begin(), arr.end(), greater<int>());

// store the first k element in result array

vector<int> res(arr.begin(), arr.begin() + k);

return res;

}

int main() {

vector<int>arr = {1, 23, 12, 9, 30, 2, 50};

int k = 3;

vector<int> res = kLargest(arr, k);

for(int ele : res)

cout << ele << " ";

return 0;

}

6) form the largest number

string printLargest(vector<int> &arr) {  
        // code here  
          
         sort(begin(arr),end(arr),[]( int & s1, int  &s2){  
             string t1= to\_string(s1)+to\_string(s2);  
             string t2= to\_string(s2)+to\_string(s1);;  
             return t1> t2;  
         });  
           
         string temp="";  
           
         for(auto &it :arr ){  
             temp+=to\_string(it);  
         }  
           
         if(temp[0]=='0') return "0";  
        return temp;  
    }