**19. Vectors**

Vector Is a Dynamic Array.

#include<iostream>

#include<vector> // New header file include

using namespace std;

int main()

{

    vector<int> v;      // initialise vectors

    v.push\_back(1);

    v.push\_back(2);

    v.push\_back(3);

    for(int i=0; i<v.size(); i++){  // print vector

        cout<<v[i]<<" ";

    }cout<<endl;

    vector<int>::iterator it; // print for iterator.

    for(it=v.begin(); it!=v.end(); it++){

        cout<<\*it<<" ";

    }cout<<endl;

    for(auto element:v){    // print for auto keyword.

        cout<<element<<" ";

    }cout<<endl;

    v.pop\_back();   // 1 2

    vector<int> v2 (3,50);

    // 50 50 50

    swap(v,v2);

    for(auto element:v){

        cout<<element<<" "; // 50 50 50

    }cout<<endl;

    for(auto element:v2){

        cout<<element<<" "; // 1 2

    }cout<<endl;

    sort(v2.begin(), v2.end());

    return 0;

}

**Pairs:**

#include<bits/stdc++.h>

using namespace std;

bool myCompure(pair<int, int> p1, pair<int, int> p2){

    return p1.first<p2.first;

}

int main(){

    int arr[] = {10,16,7,14,5,3,12,9};

    vector < pair<int, int> > v;

    // for(int i=0; i<(sizeof(arr)/sizeof(arr[0])); i++){

    //     pair<int, int> p;

    //     p.first = arr[i];

    //     p.second = i;

    // }

    for(int i=0; i<(sizeof(arr)/sizeof(arr[0])); i++){

        v.push\_back(make\_pair(arr[i],i));

    }

    sort(v.begin(), v.end(), myCompure);

    for(int i=0; i<v.size(); i++){

        arr[v[i].second]=i;

    }

    for(int i=0; i<v.size(); i++){

        cout<<arr[i]<<" ";

    }cout<<endl;

    return 0;

}

**Output:** 4 7 2 6 1 0 5 3

**19.1 Merge Sort:**

T(n) = 2T(n/2) + θ(n)

**Time Complexity:** O(n log(n)),

**Auxiliary Space:** O(n).

#include <bits/stdc++.h>

using namespace std;

void merge(int arr[], int l, int mid, int r){

    int n1 = mid - l + 1;

    int n2 = r - mid;

    int a[n1], b[n2]; // temp arr

    for (int i=0; i<n1; i++){

        a[i] = arr[l+i];

    }

    for (int i=0; i<n2; i++){

        b[i] = arr[mid+1+i];

    }

    int i = 0; // Initial index of first subarray

    int j = 0; // Initial index of second subarray

    int k = l; // Initial index of merged subarray

    while (i < n1 && j < n2){

        if (a[i] <= b[j]){

            arr[k] = a[i];

            i++;

        }

        else{

            arr[k] = b[j];

            j++;

        }

        k++;

    }

    while(i < n1){

        arr[k] = a[i];

        i++; k++;

    }

    while(j < n2){

        arr[k] = b[j];

        j++; k++;

    }

}

void mergeSort(int arr[], int l, int r){

    if (l < r){

        int mid = (l + r) / 2;

        mergeSort(arr, l, mid);

        mergeSort(arr, mid + 1, r);

        merge(arr, l, mid, r);

    }

}

int main(){

    int arr[] = {12, 11, 13, 5, 6, 7};

    mergeSort(arr, 0, 5);

    for (int i=0; i<6; i++){

        cout << arr[i] << " ";

    }

    cout << endl;

    return 0;

}

**Output:** 5 6 7 11 12 13

**19.2 Quick Sort:**

T(n) = 2T(n/2) + θ(n)

**Time Complexity:** O(n log(n)),

**Auxiliary Space:** O(n2).

#include <bits/stdc++.h>

using namespace std;

void swap(int arr[], int i, int j){

    int temp=arr[i];

    arr[i]=arr[j];

    arr[j]=temp;

}

int partition(int arr[], int l, int r){

    int pivot = arr[r];

    int i = l-1;

    for (int j=l; j<r; j++){

        if(arr[j]<pivot){

            i++;

            swap(arr, i, j);

        }

    }

    swap(arr,i+1,r);

    return i+1;

}

void quickSort(int arr[], int l, int r){

    if (l < r){

        int pi = partition(arr,l,r);

        quickSort(arr, l, pi-1);

        quickSort(arr, pi+1, r);

    }

}

int main(){

    int arr[] = {12, 11, 13, 5, 6, 7};

    quickSort(arr, 0, 5);

    for (int i=0; i<6; i++)

        cout << arr[i] << " ";

    cout << endl;

    return 0;

}

**Output:** 5 6 7 11 12 13

**19.3 Count Inversion - Merge Sort**

#include<bits/stdc++.h>

using namespace std;

long long merge(int arr[], int l, int mid, int r){

    long long inv = 0;

    int n1 = mid-l+1;

    int n2 = r - mid;

    int a[n1];

    int b[n2];

    for(int i=0; i<n1; i++){

        a[i] = arr[l+i];

    }

    for(int i=0; i<n2; i++){

        b[i] = arr[mid+i+1];

    }

    int i = 0, j = 0, k = l;

    while(i<n1 and j<n2){

        if(a[i] <= b[j]){

            arr[k] = a[i];

            k++; i++;

        }

        else{

            arr[k] = b[j];

            inv += n1-i;    // important

            k++; i++;

        }

    }

    while(i<n1){

        arr[k] = a[i];

        k++; i++;

    }

    while(j<n2){

        arr[k] = b[j];

        k++; j++;

    }

    return inv;

}

long long mergeSort(int arr[], int l, int r){

    long long inv = 0;

    if(l<r){

        int mid = (l+r)/2;

        inv += mergeSort(arr,l,mid);

        inv += mergeSort(arr,mid+1,r);

        inv += merge(arr,l,mid,r);

    }

}

int main(){

    int n; cin>>n;

    int arr[n];

    for(int i=0;i<n;i++){

        cin>>arr[i];

    }

    cout<<mergeSort(arr,0,n-1);

    return 0;

}

**20.1 Count Sort:**

**Time Complexity:** O(n + k)

Where, n is no. of element in the input array & k is the range of input.

**Auxiliary Space:** O(n + k)

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

void countSort(int arr[], int n){

    int k = arr[0];

    for (int i = 0; i < n; i++){

        k = max(k, arr[i]);

    }

    int count[10] = {0};

    for (int i = 0; i < n; i++){

        count[arr[i]]++;

    }

    for (int i = 1; i <= k; i++){

        count[i] += count[i - 1];

    }

    int output[n];

    for (int i = n - 1; i >= 0; i--){

        output[--count[arr[i]]] = arr[i];

    }

    for (int i = 0; i < n; i++){

        arr[i] = output[i];

    }

}

int main(){

    int arr[] = {1, 3, 2, 3, 4, 1, 6, 4, 3};

    countSort(arr, 9);

    for (int i = 0; i < 9; i++){

        cout << arr[i] << " ";

    }

    cout << endl;

    return 0;

}

**Output:** 1 1 2 3 3 3 4 4 6

**20.2 DNF Sort:**

#include<bits/stdc++.h>

using namespace std;

void swap(int arr[], int i, int j){

    int temp=arr[i];

    arr[i]=arr[j];

    arr[j]=temp;

}

void dnfsort(int arr[], int n){

    int low = 0;

    int mid = 0;

    int high = n-1;

    while(mid<=high){

        if(arr[mid]==0){

            swap(arr, low, mid);

            low++; mid++;

        }

        else if(arr[mid]==1){

            mid++;

        }

        else{

            swap(arr, mid, high);

            high--;

        }

    }

}

int main(){

    int arr[] = {1,0,2,1,0,1,2,1,2};

    dnfsort(arr,9);

    for(int i=0; i<9; i++){

        cout<<arr[i]<<" ";

    }

    cout<<endl;

    return 0;

}

**Output:** 0 0 1 1 1 1 2 2 2

**20.3 Wave Sort:**

**arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4]...**

#include<bits/stdc++.h>

using namespace std;

void swap(int arr[], int i, int j){

    int temp = arr[i];

    arr[i] = arr[j];

    arr[j] = temp;

}

void wavesort(int arr[], int n){

    for(int i=1; i<n; i+=2){  // increased by 2

        if(arr[i]>arr[i-1]){

            swap(arr, i, i-1);

        }

        if(arr[i]>arr[i+1] && i<=n-2){

            swap(arr, i, i+1);

        }

    }

}

int main(){

    int arr[] = {1,3,4,7,5,6,2};

    wavesort(arr, 7);

    for(int i=0; i<7; i++)

        cout<<arr[i]<<" ";

    return 0;

}

Output: 3 1 7 4 6 2 5