# Selection Sort

* Find the minimum and swap it

Code:

for(int i=0; i<=n-2; i++){

int mini = i;

for(int j=i+1; j<n; j++){

if(arr[mini]>arr[j]) mini = j;

}

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

}

**Time complexity: O(n2)** ------🡪 In best, average, worst

Space Complexity: O(1)

# Bubble Sort

* Find the max in a range and by swapping push max to the last

Code:

for(int i=n-1; i>=1; i--){

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

if(arr[j] > arr[j+1]){

swap(arr[j], arr[j+1]);

}

}

}

**Time complexity: O(n2)** in every case

**We can Optimize it:** by check if our array is not swapping in any iteration then we can say sorted.

for(int i=n-1; i>=1; i--){

bool checkswap=true;

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

if(arr[j] > arr[j+1]){

swap(arr[j], arr[j+1]);

checkswap=false;

}

}

if(checkswap) break;

}

**Time Complexity:**

**O(n) ------------🡪 Best case**

**O(n2) ----------🡪 Average and Worst Case**

**Space complexity: O(1)**

**Here is the Recursive Solution:**

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

if(n == 1) return;

bool notswapped=true;

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

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

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

notswapped=false;

}

}

if(notswapped) return;

bubbleSort(arr, n-1);

}

**Space Complexity: O(n)**  auxiliary stack space.

**Reference:**

[**https://takeuforward.org/arrays/recursive-bubble-sort-algorithm/**](https://takeuforward.org/arrays/recursive-bubble-sort-algorithm/)

# Insertion Sort

* Take an element and place it in correct order

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

int j=i;

while(j>0 && arr[j] < arr[j-1]){

swap(arr[j], arr[j-1]);

j--;

}

}

**Time Complexity:**

**O(n) ---------🡪 in Best Case**

**O(n2) -------🡪 in both Average and Worst Case**

**Space Complexity: O(1)**

**Here is the Recursive Solution:**

void fun(int arr[], int i, int n){

if(i == n) return;

int j=i;

while(j>0 && arr[j] < arr[j-1]){

int temp = arr[j];

arr[j] = arr[j-1];

arr[j-1] = temp;

j--;

}

fun(arr, i+1, n);

}

**Space Complexity: O(n)**  auxiliary stack space.

**Reference:**

[**https://takeuforward.org/arrays/recursive-insertion-sort-algorithm/**](https://takeuforward.org/arrays/recursive-insertion-sort-algorithm/)

**References:**

<https://www.youtube.com/watch?v=HGk_ypEuS24&t=1197s>

<https://takeuforward.org/strivers-a2z-dsa-course/strivers-a2z-dsa-course-sheet-2/>

# Merge Sort

* Divide the array into sub array until it becomes size 1 and then merge it

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

    vector<int> temp;

    int left=low;

    int right=mid+1;

    while(left<=mid && right<=high){

        if(arr[left] <= arr[right]){

            temp.push\_back(arr[left]);

            left++;

        }

        else{

            temp.push\_back(arr[right]);

            right++;

        }

    }

    while(left <= mid){

        temp.push\_back(arr[left]);

        left++;

    }

    while(right <= high){

        temp.push\_back(arr[right]);

        right++;

    }

    for(int i=low; i<=high; i++){

        arr[i] = temp[i-low];

    }

}

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

        if(low == high) return;

        int mid = (low+high)/2;

        mSort(arr, low, mid);

        mSort(arr, mid+1, high);

        merge(arr, low, mid, high);

}

void mergeSort(vector < int > & arr, int n) {

    mSort(arr, 0, n-1);

}

**Time Complexity: O(nlog(n)) [in merge function it requires n operation and as we are dividing our array into half everytime.]**

**Space Complexity: O(n) [in final merge temp can take a n size memory]**

**Reference:**

<https://takeuforward.org/data-structure/merge-sort-algorithm/>

<https://www.youtube.com/watch?v=ogjf7ORKfd8>

# Quick Sort (Divide and Conquer)

* Pick a pivot and place it in the correct place
* Smaller on the left and Larger on the right

#include <bits/stdc++.h>

int findpivot(vector<int> &v, int low, int high){

int pivot = v[low];

int i = low;

int j = high;

while(i < j){

while(v[i]<=pivot && i<high) i++;

while(v[j]>pivot && j>low) j--;

if(i < j) swap(v[i], v[j]);

}

swap(v[low], v[j]);

return j;

}

void qs(vector<int> &v, int low, int high){

if(low < high){

int pivot = findpivot(v, low, high);

qs(v, low, pivot-1);

qs(v, pivot+1, high);

}

}

vector<int> quickSort(vector<int> arr)

{

qs(arr, 0, arr.size()-1);

return arr;

}

**Time Complexity: O(nlog(n)) [same reason as merge]**

**Space Complexity: O(1) [as we are not using any space except stack space]**

**Reference:**

<https://www.youtube.com/watch?v=WIrA4YexLRQ>

<https://takeuforward.org/data-structure/quick-sort-algorithm/>