**Selection Sort**

#include <bits/stdc++.h>

using namespace std;

// Function for Selection sort

void selectionSort(int arr[], int n)

{

int i, j, min\_idx;

// One by one move boundary of

// unsorted subarray

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

// Find the minimum element in

// unsorted array

min\_idx = i;

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

if (arr[j] < arr[min\_idx])

min\_idx = j;

}

// Swap the found minimum element

// with the first element

if (min\_idx != i)

swap(arr[min\_idx], arr[i]);

}

}

// Function to print an array

void printArray(int arr[], int size)

{

int i;

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

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

cout << endl;

}

}

int main()

{

int arr[] = { 64, 25, 12, 22, 11 };

int n = sizeof(arr) / sizeof(arr[0]);

// Function Call

selectionSort(arr, n);

cout << "Sorted array: \n";

printArray(arr, n);

return 0;

}

**Bubble Sort**

#include <bits/stdc++.h>

using namespace std;

// An optimized version of Bubble Sort

void bubbleSort(int arr[], int n)

{

int i, j;

bool swapped;

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

swapped = false;

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

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

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

swapped = true;

}

}

// If no two elements were swapped

// by inner loop, then break

if (swapped == false)

break;

}

}

// Function to print an array

void printArray(int arr[], int size)

{

int i;

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

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

}

// Driver program to test above functions

int main()

{

int arr[] = { 64, 34, 25, 12, 22, 11, 90 };

int N = sizeof(arr) / sizeof(arr[0]);

bubbleSort(arr, N);

cout << "Sorted array: \n";

printArray(arr, N);

return 0;

}

**Merge Sort**

#include <bits/stdc++.h>

using namespace std;

// Merges two subarrays of array[].

// First subarray is arr[begin..mid]

// Second subarray is arr[mid+1..end]

void merge(int array[], int const left, int const mid,

int const right)

{

int const subArrayOne = mid - left + 1;

int const subArrayTwo = right - mid;

// Create temp arrays

auto \*leftArray = new int[subArrayOne],

\*rightArray = new int[subArrayTwo];

// Copy data to temp arrays leftArray[] and rightArray[]

for (auto i = 0; i < subArrayOne; i++)

leftArray[i] = array[left + i];

for (auto j = 0; j < subArrayTwo; j++)

rightArray[j] = array[mid + 1 + j];

auto indexOfSubArrayOne = 0, indexOfSubArrayTwo = 0;

int indexOfMergedArray = left;

// Merge the temp arrays back into array[left..right]

while (indexOfSubArrayOne < subArrayOne

&& indexOfSubArrayTwo < subArrayTwo) {

if (leftArray[indexOfSubArrayOne]

<= rightArray[indexOfSubArrayTwo]) {

array[indexOfMergedArray]

= leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

}

else {

array[indexOfMergedArray]

= rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

}

indexOfMergedArray++;

}

// Copy the remaining elements of

// left[], if there are any

while (indexOfSubArrayOne < subArrayOne) {

array[indexOfMergedArray]

= leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

indexOfMergedArray++;

}

// Copy the remaining elements of

// right[], if there are any

while (indexOfSubArrayTwo < subArrayTwo) {

array[indexOfMergedArray]

= rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

indexOfMergedArray++;

}

delete[] leftArray;

delete[] rightArray;

}

// begin is for left index and end is right index

// of the sub-array of arr to be sorted

void mergeSort(int array[], int const begin, int const end)

{

if (begin >= end)

return;

int mid = begin + (end - begin) / 2;

mergeSort(array, begin, mid);

mergeSort(array, mid + 1, end);

merge(array, begin, mid, end);

}

// UTILITY FUNCTIONS

// Function to print an array

void printArray(int A[], int size)

{

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

cout << A[i] << " ";

cout << endl;

}

int main()

{

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

int arr\_size = sizeof(arr) / sizeof(arr[0]);

cout << "Given array is \n";

printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

cout << "\nSorted array is \n";

printArray(arr, arr\_size);

return 0;

}