

**Arrays sort function**

***sorts:***

**01- Bubble sort**

**02- Insertion sort**

**03- Selection sort**

**04- Bubble sort recursion**

**05- Insertion sort recursion**

**06- Selection sort recursion**

**07- Odd even sort**

**08- Merge sort**

**09- Quick sort**

1. Arrays: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Create a file with an extension of .cpp or .c**

**Main function**

**And display function**

**void display(int array[], int size)**

**{**

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

**printf("%d, ", array[i]);**

**}**

**int main()**

**{**

**int array[] = {1, 5, -6, -1, 18, 45, 2, 2, 3, 4, 19};**

**int size = sizeof(array) / sizeof(array[0]);**

**printf("\nArray before sorting:\n");**

**display(array, size);**

**nameOfTheSortFunction(array, size); // import function name here**

**printf("\nArray after sorting:\n");**

**display(array, size);**

**return 0;**

**}**

****

**1- Arrays: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

***sorts:***

1. **Bubble sort**

**void swap(int array[], int i, int j)**

**{**

**int temp = array[i];**

**array[i] = array[j];**

**array[j] = temp;**

**}**

**void bubble\_sort(int array[], int size)**

**{**

**int i, j, temp;**

**for (i = 0; i < size - 1; i++)**

**{**

**for (j = i + 1; j < size; j++)**

**{**

**if (array[i] > array[j])**

**{**

**swap(array, i, j);**

**}**

**}**

**}**

**}**

1. **Insertion sort**

**void insertion\_sort(int a[], int size)**

**{**

**int i, j, current;**

**for (i = 1; i < size; i++)**

**{**

**current = a[i];**

**j = i - 1;**

**while (j >= 0 && a[j] > current)**

**{**

**a[j + 1] = a[j];**

**j--;**

**}**

**a[j + 1] = current;**

**}**

**}**

1. **Selection sort**

**void selection\_sort(int a[], int size)**

**{**

**int i, j, min, temp;**

**for (i = 0; i < size - 1; i++)**

**{**

**min = i;**

**for (j = i + 1; j < size; j++)**

**{**

**if (a[min] > a[j])**

**min = j;**

**}**

**if (min != i)**

**{**

**temp = a[i];**

**            a[i] = a[min];**

**a[min] = temp;**

**}**

**}**

**}**

**04- Bubble sort recursion**

**void swap(int array[], int j)**

**{**

**int temp = array[j];**

**array[j] = array[j + 1];**

**array[j + 1] = temp;**

**}**

**void bubble\_recursion\_sort(int array[], int size)**

**{**

**int i, j, temp;**

**if(size >= 0)**

**{**

**for (j = 0; j < size - 1; j++)**

**{**

**if (array[j] > array[j + 1])**

**{**

**swap(array, j);**

**}**

**}**

**return bubble\_recursion\_sort(array, size - 1);**

**}**

**}**

**05- Insertion sort recursion**

**void insertion\_recursion\_sort(int a[], int size)**

**{**

**int i, j, last;**

**if (size <= 1)**

**return;**

**insertion\_recursion\_sort(a, size - 1);**

**last = a[size - 1];**

**j = size - 2;**

**while (j >= 0 && a[j] > last)**

**{**

**a[j + 1] = a[j];**

**j--;**

**}**

**a[j + 1] = last;**

**}**

****

**06- Selection sort recursion**

**void selection\_recursion\_sort(int a[], int start, int size)**

**{**

**int i, j, location, temp;**

**if (start >= size)**

**{**

**return;**

**}**

**location = start;**

**for (i = start + 1; i < size; i++)**

**{**

**if (a[location] > a[i])**

**{**

**location = i;**

**}**

**}**

**if (location != start)**

**{**

**temp = a[start];**

**a[start] = a[location];**

**a[location] = temp;**

**}**

**return selection\_recursion\_sort(a, start + 1, size);**

**}**

**07- Odd even sort**

**void odd\_even\_sort(int a[], int size)**

**{**

**int i, j, flag, temp;**

**do**

**{**

**flag = 0;**

**// even sorting**

**for (i = 0; i < size - 1; i += 2)**

**{**

**if (a[i] > a[i + 1])**

**{**

**temp = a[i];**

**a[i] = a[i + 1];**

**a[i + 1] = temp;**

**flag = 1;**

**}**

**}**

**// odd sorting**

**for (i = 1; i < size - 1; i += 2)**

**{**

**if (a[i] > a[i + 1])**

**{**

**temp = a[i];**

**a[i] = a[i + 1];**

**a[i + 1] = temp;**

**flag = 1;**

**}**

**}**

**} while (flag == 1);**

**}**

****

**08- Merge sort**

**void merge(int arr[], int l, int m, int r)**

**{**

**int i, j, k;**

**int n1 = m - l + 1;**

**int n2 = r - m;**

**int L[n1], R[n2];**

**for (i = 0; i < n1; i++)**

**L[i] = arr[l + i];**

**for (j = 0; j < n2; j++)**

**R[j] = arr[m + 1 + j];**

**i = 0;**

**j = 0;**

**k = l;**

**while (i < n1 && j < n2)**

**{**

**if (L[i] <= R[j])**

**{**

**arr[k] = L[i];**

**i++;**

**}**

**else**

**{**

**arr[k] = R[j];**

**j++;**

**}**

**k++;**

**}**

**while (i < n1)**

**{**

**arr[k] = L[i];**

**i++;**

**k++;**

**}**

**while (j < n2)**

**{**

**arr[k] = R[j];**

**j++;**

**k++;**

**}**

**}**

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

**{**

**if (l < r)**

**{**

**int m = l + (r - l) / 2;**

**mergeSort(arr, l, m);**

**mergeSort(arr, m + 1, r);**

**merge(arr, l, m, r);**

**}**

**}**

****

**09- Quick sort**

**#include <stdio.h>**

**void swap(int \*a, int \*b)**

**{**

**int t = \*a;**

**\*a = \*b;**

**\*b = t;**

**}**

**int partition(int array[], int low, int high)**

**{**

**int pivot = array[high];**

**int i = (low - 1);**

**for (int j = low; j < high; j++)**

**{**

**if (array[j] <= pivot)**

**{**

**i++;**

**swap(&array[i], &array[j]);**

**}**

**}**

**swap(&array[i + 1], &array[high]);**

**return (i + 1);**

**}**

**void quickSort(int array[], int low, int high)**

**{**

**if (low < high)**

**{**

**int pi = partition(array, low, high);**

**quickSort(array, low, pi - 1);**

**quickSort(array, pi + 1, high);**

**}**

**}**

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

**{**

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

**printf("%d  ", array[i]);**

**printf("\n");**

**}**

**int main()**

**{**

**int data[] = {8, 7, 2, 1, 0, 9, 6};**

**int n = sizeof(data) / sizeof(data[0]);**

**printf("Unsorted Array\n");**

**printArray(data, n);**

**quickSort(data, 0, n - 1);**

**printf("Sorted array in ascending order: \n");**

**printArray(data, n);**

**}**

****

**Created by Mohammed Elidrissi Laoukili**

**لا اله الا الله**