Selection Sort-

Code in C

// C program for implementation of selection sort

#include <stdio.h>

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

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

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++)

printf("%d ", arr[i]);

printf("\n");

}

// Driver program to test above functions

int main()

{

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

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

selectionSort(arr, n);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}

Code in C++

// C++ program for implementation of selection sort

#include <bits/stdc++.h>

using namespace std;

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

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

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;

}

// Driver program to test above functions

int main()

{

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

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

selectionSort(arr, n);

cout << "Sorted array: \n";

printArray(arr, n);

return 0;

}

insertion sort

// C program for insertion sort

#include <math.h>

#include <stdio.h>

/\* Function to sort an array using insertion sort\*/

void insertionSort(int arr[], int n)

{

int i, key, j;

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

key = arr[i];

j = i - 1;

/\* Move elements of arr[0..i-1], that are

greater than key, to one position ahead

of their current position \*/

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

// A utility function to print an array of size n

void printArray(int arr[], int n)

{

int i;

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

printf("%d ", arr[i]);

printf("\n");

}

/\* Driver program to test insertion sort \*/

int main()

{

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

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

insertionSort(arr, n);

printArray(arr, n);

return 0;

}

// C++ program for insertion sort

#include <bits/stdc++.h>

using namespace std;

/\* Function to sort an array using insertion sort\*/

void insertionSort(int arr[], int n)

{

int i, key, j;

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

{

key = arr[i];

j = i - 1;

/\* Move elements of arr[0..i-1], that are

greater than key, to one position ahead

of their current position \*/

while (j >= 0 && arr[j] > key)

{

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

// A utility function to print an array of size n

void printArray(int arr[], int n)

{

int i;

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

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

cout << endl;

}

/\* Driver code \*/

int main()

{

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

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

insertionSort(arr, n);

printArray(arr, n);

return 0;

}

Bubble sort

// C program for implementation of Bubble sort

#include <stdio.h>

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

// A function to implement bubble sort

void bubbleSort(int arr[], int n)

{

int i, j;

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

// Last i elements are already in place

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

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

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

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

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

printf("%d ", arr[i]);

printf("\n");

}

// 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);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}

// C++ program for implementation of Bubble sort

#include <bits/stdc++.h>

using namespace std;

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

// A function to implement bubble sort

void bubbleSort(int arr[], int n)

{

int i, j;

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

// Last i elements are already in place

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

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

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

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

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

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

cout << endl;

}

// Driver code

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;

}

Shell sort

// Shell Sort in C programming

#include <stdio.h>

// Shell sort

void shellSort(int array[], int n) {

// Rearrange elements at each n/2, n/4, n/8, ... intervals

for (int interval = n / 2; interval > 0; interval /= 2) {

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

int temp = array[i];

int j;

for (j = i; j >= interval && array[j - interval] > temp; j -= interval) {

array[j] = array[j - interval];

}

array[j] = temp;

}

}

}

// Print an array

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

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

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

}

printf("\n");

}

// Driver code

int main() {

int data[] = {9, 8, 3, 7, 5, 6, 4, 1};

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

shellSort(data, size);

printf("Sorted array: \n");

printArray(data, size);

}

// C++ implementation of Shell Sort

#include <iostream>

using namespace std;

/\* function to sort arr using shellSort \*/

int shellSort(int arr[], int n)

{

// Start with a big gap, then reduce the gap

for (int gap = n/2; gap > 0; gap /= 2)

{

// Do a gapped insertion sort for this gap size.

// The first gap elements a[0..gap-1] are already in gapped order

// keep adding one more element until the entire array is

// gap sorted

for (int i = gap; i < n; i += 1)

{

// add a[i] to the elements that have been gap sorted

// save a[i] in temp and make a hole at position i

int temp = arr[i];

// shift earlier gap-sorted elements up until the correct

// location for a[i] is found

int j;

for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)

arr[j] = arr[j - gap];

// put temp (the original a[i]) in its correct location

arr[j] = temp;

}

}

return 0;

}

void printArray(int arr[], int n)

{

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

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

}

int main()

{

int arr[] = {12, 34, 54, 2, 3}, i;

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

cout << "Array before sorting: \n";

printArray(arr, n);

shellSort(arr, n);

cout << "\nArray after sorting: \n";

printArray(arr, n);

return 0;

}

Merge sort

/\* C and C++ program for Merge Sort \*/

#include <stdio.h>

#include <stdlib.h>

// Merges two subarrays of arr[].

// First subarray is arr[l..m]

// Second subarray is arr[m+1..r]

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

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

/\* create temp arrays \*/

int L[n1], R[n2];

/\* Copy data to temp arrays L[] and R[] \*/

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

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

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

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

/\* Merge the temp arrays back into arr[l..r]\*/

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

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

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

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy the remaining elements of L[], if there

are any \*/

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

/\* Copy the remaining elements of R[], if there

are any \*/

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

/\* l is for left index and r is right index of the

sub-array of arr to be sorted \*/

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

{

if (l < r) {

// Same as (l+r)/2, but avoids overflow for

// large l and h

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

// Sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

/\* UTILITY FUNCTIONS \*/

/\* Function to print an array \*/

void printArray(int A[], int size)

{

int i;

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

printf("%d ", A[i]);

printf("\n");

}

/\* Driver program to test above functions \*/

int main()

{

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

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

printf("Given array is \n");

printArray(arr, arr\_size);

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

printf("\nSorted array is \n");

printArray(arr, arr\_size);

return 0;

}

Quick sort

/\* C implementation QuickSort \*/

#include<stdio.h>

// A utility function to swap two elements

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

{

int t = \*a;

\*a = \*b;

\*b = t;

}

/\* This function takes last element as pivot, places

the pivot element at its correct position in sorted

array, and places all smaller (smaller than pivot)

to left of pivot and all greater elements to right

of pivot \*/

int partition (int arr[], int low, int high)

{

int pivot = arr[high]; // pivot

int i = (low - 1); // Index of smaller element

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

{

// If current element is smaller than the pivot

if (arr[j] < pivot)

{

i++; // increment index of smaller element

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

}

}

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

return (i + 1);

}

/\* The main function that implements QuickSort

arr[] --> Array to be sorted,

low --> Starting index,

high --> Ending index \*/

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

/\* pi is partitioning index, arr[p] is now

at right place \*/

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

// Separately sort elements before

// partition and after partition

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

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

printf("%d ", arr[i]);

printf("n");

}

// Driver program to test above functions

int main()

{

int arr[] = {10, 7, 8, 9, 1, 5};

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

quickSort(arr, 0, n-1);

printf("Sorted array: n");

printArray(arr, n);

return 0;

}

/\* C++ implementation of QuickSort \*/

#include <bits/stdc++.h>

using namespace std;

// A utility function to swap two elements

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

{

int t = \*a;

\*a = \*b;

\*b = t;

}

/\* This function takes last element as pivot, places

the pivot element at its correct position in sorted

array, and places all smaller (smaller than pivot)

to left of pivot and all greater elements to right

of pivot \*/

int partition (int arr[], int low, int high)

{

int pivot = arr[high]; // pivot

int i = (low - 1); // Index of smaller element

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

{

// If current element is smaller than the pivot

if (arr[j] < pivot)

{

i++; // increment index of smaller element

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

}

}

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

return (i + 1);

}

/\* The main function that implements QuickSort

arr[] --> Array to be sorted,

low --> Starting index,

high --> Ending index \*/

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

/\* pi is partitioning index, arr[p] is now

at right place \*/

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

// Separately sort elements before

// partition and after partition

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

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

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

cout << endl;

}

// Driver Code

int main()

{

int arr[] = {10, 7, 8, 9, 1, 5};

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

quickSort(arr, 0, n - 1);

cout << "Sorted array: \n";

printArray(arr, n);

return 0;

}

Heap sort

// Heap Sort in C

#include <stdio.h>

// Function to swap the the position of two elements

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

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

// Find largest among root, left child and right child

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && arr[left] > arr[largest])

largest = left;

if (right < n && arr[right] > arr[largest])

largest = right;

// Swap and continue heapifying if root is not largest

if (largest != i) {

swap(&arr[i], &arr[largest]);

heapify(arr, n, largest);

}

}

// Main function to do heap sort

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

// Build max heap

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

// Heap sort

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

swap(&arr[0], &arr[i]);

// Heapify root element to get highest element at root again

heapify(arr, i, 0);

}

}

// Print an array

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

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

printf("%d ", arr[i]);

printf("\n");

}

// Driver code

int main() {

int arr[] = {1, 12, 9, 5, 6, 10};

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

heapSort(arr, n);

printf("Sorted array is \n");

printArray(arr, n);

}

// C++ program for implementation of Heap Sort

#include <iostream>

using namespace std;

// To heapify a subtree rooted with node i which is

// an index in arr[]. n is size of heap

void heapify(int arr[], int n, int i)

{

int largest = i; // Initialize largest as root

int l = 2\*i + 1; // left = 2\*i + 1

int r = 2\*i + 2; // right = 2\*i + 2

// If left child is larger than root

if (l < n && arr[l] > arr[largest])

largest = l;

// If right child is larger than largest so far

if (r < n && arr[r] > arr[largest])

largest = r;

// If largest is not root

if (largest != i)

{

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

// Recursively heapify the affected sub-tree

heapify(arr, n, largest);

}

}

// main function to do heap sort

void heapSort(int arr[], int n)

{

// Build heap (rearrange array)

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

// One by one extract an element from heap

for (int i=n-1; i>0; i--)

{

// Move current root to end

swap(arr[0], arr[i]);

// call max heapify on the reduced heap

heapify(arr, i, 0);

}

}

/\* A utility function to print array of size n \*/

void printArray(int arr[], int n)

{

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

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

cout << "\n";

}

// Driver program

int main()

{

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

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

heapSort(arr, n);

cout << "Sorted array is \n";

printArray(arr, n);

}