// C program for implementation of selection sort

#include <stdio.h>

void selectionSort(int arr[], int n)

{

int i=0, loc=0, temp=0;

// One by one move boundary of unsorted subarray

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

{

// Find the minimum element in unsorted array

loc=min(arr,i,n) ;

temp=arr[loc];

arr[loc]=arr[loc];

arr[i]=temp;

}

}

int min(up\_bound,low\_bound,arr)

{

int min = low\_bound;

while(low\_bound<up\_bound)

{

if(arr[low\_bound < arr[min]])

min=low\_bound;

low\_bound++;

}

return min;

}

/\* 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;

}

// SORTING MENU DRIVEN

|  |
| --- |
| #include <stdio.h> |
|  | #include <stdlib.h> |
|  |  |
|  | void bubbleSort(int[], int); |
|  | void insertionSort(int[], int); |
|  | void quickSort(int[], int, int); |
|  | void mergeSort(int[], int, int); |
|  |  |
|  | int main() { |
|  | int n, arr[100]; |
|  | int i; |
|  | int op, cont; |
|  | do { |
|  | printf("Enter size of array: "); |
|  | scanf("%d", &n); |
|  | printf("Enter values of array:\n"); |
|  | for(i = 0; i < n; i++) { |
|  | scanf("%d", &arr[i]); |
|  | } |
|  | printf("Enter sorting algorithm to use:\n1: Bubble\n2: Insertion\n3: Quick sort\n4: Merge sort\n"); |
|  | scanf("%d", &op); |
|  |  |
|  | switch(op) { |
|  | case 1: |
|  | bubbleSort(arr, n); |
|  | break; |
|  | case 2: |
|  | insertionSort(arr, n); |
|  | break; |
|  | case 3: |
|  | quickSort(arr, 0, n - 1); |
|  | break; |
|  | case 4: |
|  | mergeSort(arr, 0, n - 1); |
|  | break; |
|  | default: |
|  | printf("Invalid option!"); |
|  | } |
|  | for(i = 0; i < n; i++) { |
|  | printf("%d ", arr[i]); |
|  | } |
|  | printf("\nContinue? 1/0:\t"); |
|  | scanf("%d", &cont); |
|  | } while(cont == 1); |
|  |  |
|  | return 0; |
|  | } |
|  |  |
|  | void bubbleSort(int arr[], int size) { |
|  | int i, j, temp; |
|  | for(i = 0; i < size - 1; i++) { |
|  | for(j = 0; j < size - i - 1; j++) { |
|  | if (arr[j] > arr[j + 1]) { |
|  | //swap |
|  | temp = arr[j]; |
|  | arr[j] = arr[j + 1]; |
|  | arr[j + 1] = temp; |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void insertionSort(int arr[], int size) { |
|  | int i, j, key; |
|  | for(i = 1; i < size; i++) { |
|  | key = arr[i]; |
|  | for(j = i; j > 0 && arr[j - 1] > key; j--) { |
|  | arr[j] = arr[j - 1]; |
|  | } |
|  | arr[j] = key; |
|  | } |
|  | } |
|  |  |
|  | int partition(int arr[], int offset, int size) { |
|  | int x = arr[size]; |
|  | int i = offset - 1; |
|  | int j; |
|  | int temp; |
|  | for(j = offset; j < size; j++) { |
|  | if (arr[j] <= x) { |
|  | i++; |
|  | temp = arr[i]; |
|  | arr[i] = arr[j]; |
|  | arr[j] = temp; |
|  | } |
|  | } |
|  | temp = arr[i + 1]; |
|  | arr[i + 1] = arr[size]; |
|  | arr[size] = temp; |
|  | return i + 1; |
|  | } |
|  |  |
|  | void quickSort(int arr[], int offset, int size) { |
|  | int pivot; |
|  | if (offset < size) { |
|  | pivot = partition(arr, offset, size); |
|  | quickSort(arr, offset, pivot - 1); |
|  | quickSort(arr, pivot + 1, size); |
|  | } |
|  | } |
|  |  |
|  | void merge(int arr[], int offset, int mid, int size) { |
|  | int n1 = mid - offset + 1; |
|  | int n2 = size - mid; |
|  | int\* l = (int\*)calloc(n1 + 1, sizeof(int)); |
|  | int\* r = (int\*)calloc(n2 + 1, sizeof(int)); |
|  | int i, j, k; |
|  | for(i = 0; i < n1; i++) { |
|  | \*(l + i) = arr[offset + i]; |
|  | } |
|  | for(j = 0; j < n2; j++) { |
|  | \*(r + j) = arr[mid + j + 1]; |
|  | } |
|  | \*(l + n1) = 32767; |
|  | \*(r + n2) = 32767; |
|  | i = j = 0; |
|  | for(k = offset; k <= size; k++) { |
|  | if(\*(l + i) <= \*(r + j)) { |
|  | arr[k] = \*(l + i); |
|  | i++; |
|  | } |
|  | else { |
|  | arr[k] = \*(r + j); |
|  | j++; |
|  | } |
|  | } |
|  | free(l); |
|  | free(r); |
|  | } |
|  |  |
|  | void mergeSort(int arr[], int offset, int size) { |
|  | if (offset < size) { |
|  | int mid = (offset + size) / 2; |
|  | mergeSort(arr, offset, mid); |
|  | mergeSort(arr, mid + 1, size); |
|  | merge(arr, offset, mid, size); |
|  | } |
|  | } |

// SORTING

#include<stdio.h>

#include<stdlib.h>

void display(int a[],int n);

void bubble\_sort(int a[],int n);

void selection\_sort(int a[],int n);

void insertion\_sort(int a[],int n);

int main()

{

int n,choice,i;

char ch[20];

printf("Enter no. of elements : ");

scanf("%d",&n);

int arr[n];

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

{

printf("Enter %d Element : ",i+1);

scanf("%d",&arr[i]);

}

printf("select option for Sorting : \n");

while(1)

{

printf("\n1. Bubble Sort\n2. Selection Sort\n3. Insertion Sort\n4. Display Array.\n5. Exit the Program.\n");

printf("\nEnter your Choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

bubble\_sort(arr,n);

break;

case 2:

selection\_sort(arr,n);

break;

case 3:

insertion\_sort(arr,n);

break;

case 4:

display(arr,n);

break;

case 5:

return 0;

default:

printf("\n Select 1-5 option ----\n");

}

}

return 0;

}

void display(int arr[],int n)

{

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

{

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

}

}

void bubble\_sort(int arr[],int n)

{

int i,j,temp;

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

{

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

{

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

{

temp=arr[j];

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

arr[j+1]=temp;

}

}

}

printf("After Bubble sort Elements are : ");

display(arr,n);

}

void selection\_sort(int arr[],int n)

{

int i,j,temp;

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

{

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

{

if(arr[i]>arr[j])

{

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

}

printf("After Selection sort Elements are : ");

display(arr,n);

}

void insertion\_sort(int arr[],int n)

{

int i,j,min;

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

{

min=arr[i];

j=i-1;

while(min<arr[j] && j>=0)

{

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

j=j-1;

}

arr[j+1]=min;

}

printf("After Insertion sort Elements are : ");

display(arr,n);

}

//Sort

// implement program for selection and insertion sort

#include <stdio.h>

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

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

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

}

printf("\n");

}

void insertionSort(int arr[], int size) {

int i, j, key;

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

key = arr[i];

j = i - 1;

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

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

j = j - 1;

}

arr[j + 1] = key;

printf("Step %d: ", i);

printArray(arr, size);

}

}

void selectionSort(int arr[], int size) {

int i, j, min\_idx, temp;

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

min\_idx = i;

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

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

min\_idx = j;

}

}

temp = arr[i];

arr[i] = arr[min\_idx];

arr[min\_idx] = temp;

printf("Step %d: ", i + 1);

printArray(arr, size);

}

}

int main() {

int choice, size;

printf("Enter the number of elements in the array: ");

scanf("%d", &size);

int arr[size];

printf("Enter the elements of the array: ");

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

scanf("%d", &arr[i]);

}

do {

printf("\nMenu:\n");

printf("1. Insertion Sort\n");

printf("2. Selection Sort\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("\nApplying Insertion Sort:\n");

insertionSort(arr, size);

break;

case 2:

printf("\nApplying Selection Sort:\n");

selectionSort(arr, size);

break;

case 3:

printf("Exiting the program.\n");

break;

default:

printf("Invalid choice! Please try again.\n");

}

} while (choice != 3);

    return 0;

}