1. BUBBLE SORT:

#include <stdio.h>

#include <stdlib.h>

void main()

{

int arr[1000], n, i, j, temp, t = 0;

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

scanf("%d", &n);

if (n <= 0) {

printf("Invalid size:\n");

return;

}

srand(1); // Seed for reproducible random numbers

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

arr[i] = rand() % 5000;

t++;

}

printf("Before sorting:\n");

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

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

t++;

}

printf("\n");

// \*\*Bubble Sort Algorithm\*\*

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

int swapped = 0; // Optimization: track if a swap was made

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

t++;

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

// Swap adjacent elements if they are in the wrong order

temp = arr[j];

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

arr[j + 1] = temp;

swapped = 1;

t++;

}

}

// If no swaps were made, the array is already sorted

if (swapped == 0) {

break;

}

}

printf("After sorting:\n");

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

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

t++;

}

printf("\n");

printf("The total number of operations (t) is: %d\n", t);

}

1. FUNNEL SORT:

#include <stdio.h>

#include <stdlib.h>

#define MAX 1000

// Merge function for funnel sorting

void funnel\_merge(int arr[], int left, int mid, int right) {

int i = left, j = mid + 1, k = 0;

int temp[right - left + 1];

while (i <= mid && j <= right) {

if (arr[i] <= arr[j])

temp[k++] = arr[i++];

else

temp[k++] = arr[j++];

}

while (i <= mid) temp[k++] = arr[i++];

while (j <= right) temp[k++] = arr[j++];

for (i = left, k = 0; i <= right; i++, k++)

arr[i] = temp[k];

}

// Recursive Funnel Sort (Divide & Merge)

void funnel\_sort(int arr[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

funnel\_sort(arr, left, mid);

funnel\_sort(arr, mid + 1, right);

funnel\_merge(arr, left, mid, right);

}

}

int main() {

int arr[MAX], n, i;

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

scanf("%d", &n);

if (n <= 0 || n > MAX) {

printf("Invalid size:\n");

return 1;

}

srand(1); // Seed for reproducible random numbers

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

arr[i] = rand() % 5000;

printf("Before sorting:\n");

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

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

printf("\n");

// Apply Funnel Sort

funnel\_sort(arr, 0, n - 1);

printf("After sorting:\n");

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

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

printf("\n");

return 0;

}

1. INSERTION:

#include <stdio.h>

#include <stdlib.h>

void main() {

int arr[1000], n, i, j, temp, t = 0;

// Taking user input for number of elements

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

scanf("%d", &n);

// Checking for invalid input

if (n <= 0) {

printf("Invalid size:\n");

return;

}

srand(1); // Seeding for reproducibility

// Generating random numbers

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

arr[i] = rand() % 5000;

t++;

}

// Displaying array before sorting

printf("Before sorting:\n");

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

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

t++;

}

printf("\n");

// Insertion Sort in Ascending Order

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

t++;

temp = arr[i];

j = i - 1;

// Moving elements greater than temp one position ahead

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

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

j--;

t++;

}

arr[j + 1] = temp;

t++;

}

// Displaying array after sorting

printf("After sorting (Ascending Order):\n");

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

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

t++;

}

printf("\n");

// Printing total number of operations

printf("The total number of operations (t) is: %d\n", t);

}

1. SELECTION:

#include <stdio.h>

#include <stdlib.h>

void main() {

int arr[1000], n, i, j, min\_idx, temp, t = 0;

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

scanf("%d", &n);

if (n <= 0) {

printf("Invalid size:\n");

return;

}

srand(1); // Seed for reproducible random numbers

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

arr[i] = rand() % 5000;

t++;

}

printf("Before sorting:\n");

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

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

t++;

}

printf("\n");

// \*\*Selection Sort Algorithm\*\*

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

t++;

min\_idx = i;

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

t++;

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

min\_idx = j;

}

}

// Swap the found minimum element with the first element of the unsorted part

if (min\_idx != i) {

temp = arr[i];

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

arr[min\_idx] = temp;

t++;

}

}

printf("After sorting:\n");

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

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

t++;

}

printf("\n");

printf("The total number of operations (t) is: %d\n", t);

}

1. LINEAR SEARCH:

#include <stdio.h>

void main() {

int a[50], key, n, i, flag = 0;

printf("Enter the number of elements you want to store in the array:\n");

scanf("%d", &n);

printf("Enter the array elements:\n");

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

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

}

printf("Enter the key:\n");

scanf("%d", &key);

// Linear Search

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

if (key == a[i]) {

flag = 1;

break;

}

}

if (flag == 1)

{

printf("Key found %d \n",i);

}

else

{

printf("Key not found\n");

}

}

1. BINARY SEARCH:

//recursive

#include <stdio.h>

// Recursive binary search function

int binarySearchRecursive(int arr[], int left, int right, int key)

{

if (left > right)

{

return -1; // Element not found

}

int mid = left + (right - left) / 2; // Avoid overflow

if (arr[mid] == key)

{

return mid; // Element found

}

else if (arr[mid] > key)

{

return binarySearchRecursive(arr, left, mid - 1, key); // Search left half

}

Else

{

return binarySearchRecursive(arr, mid + 1, right, key); // Search right half

}

}

int main() {

int size, key;

// User input for array size

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

scanf("%d", &size);

int arr[size];

// User input for sorted array

printf("Enter %d sorted elements:\n", size);

{

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

{

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

}

}

// User input for search key

printf("Enter the element to search: ");

scanf("%d", &key);

// Perform recursive binary search

int result = binarySearchRecursive(arr, 0, size - 1, key);

if (result != -1)

printf("Element found at index %d\n", result);

else

printf("Element not found\n");

return 0;

}

// Iterative binary search function

#include <stdio.h>

// Iterative binary search function

int binarySearchIterative(int arr[], int size, int key) {

int left = 0, right = size - 1;

while (left <= right) {

int mid = left + (right - left) / 2; // Avoid overflow

if (arr[mid] == key)

return mid; // Element found

else if (arr[mid] > key)

right = mid - 1; // Search left half

else

left = mid + 1; // Search right half

}

return -1; // Element not found

}

int main() {

int size, key;

// User input for array size

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

scanf("%d", &size);

int arr[size];

// User input for sorted array

printf("Enter %d sorted elements:\n", size);

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

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

}

// User input for search key

printf("Enter the element to search: ");

scanf("%d", &key);

// Perform iterative binary search

int result = binarySearchIterative(arr, size, key);

if (result != -1)

printf("Element found at index %d\n", result);

else

printf("Element not found\n");

return 0;

}

1. MERGE SORT

#include <stdio.h>

void mergesort(int[], int, int);

void merge(int[], int, int, int);

void partition(int[], int, int);

int main() {

int i, n, a[50];

printf("Enter the number of elements you want to store in the array: ");

scanf("%d", &n);

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

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

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

partition(a, 0, n - 1);

printf("Sorted elements are:\n");

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

printf("%d\t", a[i]);

return 0;

}

// Function to divide the array into two halves

void partition(int a[], int low, int high) {

int mid;

if (low < high) {

mid = (low + high) / 2;

partition(a, low, mid);

partition(a, mid + 1, high);

merge(a, low, mid, high);

}

}

// Function to merge two sorted subarrays

void merge(int a[], int low, int mid, int high) {

int i, j, k;

int leftSize = mid - low + 1;

int rightSize = high - mid;

int left[leftSize], right[rightSize];

// Copy elements into temporary arrays

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

left[i] = a[low + i];

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

right[j] = a[mid + 1 + j];

// Merge the two subarrays

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = low; // Initial index of merged subarray

while (i < leftSize && j < rightSize) {

if (left[i] <= right[j]) {

a[k] = left[i];

i++;

} else {

a[k] = right[j];

j++;

}

k++;

}

// Copy remaining elements of left subarray

while (i < leftSize) {

a[k] = left[i];

i++;

k++;

}

// Copy remaining elements of right subarray

while (j < rightSize) {

a[k] = right[j];

j++;

k++;

}

}

1. QUICKSORT

#include <stdio.h>

#include <stdlib.h> // For rand() and srand()

#include <time.h> // For seeding random numbers

void quicksort(int a[], int first, int last) {

int i, j, pivot, temp;

if (first < last) {

pivot = first;

i = first;

j = last;

while (i < j) {

while (a[i] <= a[pivot] && i < last)

i++;

while (a[j] > a[pivot])

j--;

if (i < j) {

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

temp = a[pivot];

a[pivot] = a[j];

a[j] = temp;

quicksort(a, first, j - 1);

quicksort(a, j + 1, last);

}

}

int main() {

int i, n, a[50], range;

// Seeding random function

srand(time(0));

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

scanf("%d", &n);

printf("Enter the maximum value for random numbers: ");

scanf("%d", &range);

// Generating random numbers

printf("Generated random array:\n");

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

a[i] = rand() % range; // Random number between 0 and range-1

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

}

printf("\n");

quicksort(a, 0, n - 1);

printf("Sorted elements are:\n");

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

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

printf("\n");

return 0;

}

1. FIBONACCI SERIES:

#include <stdio.h>

int fibonacci(int n) {

return (n <= 1) ? n : fibonacci(n - 1) + fibonacci(n - 2);

}

int main() {

int n;

printf("Enter number of terms: ");

scanf("%d", &n);

printf("Fibonacci Series (Recursive): ");

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

printf("%d ", fibonacci(i));

return 0;

}

#include <stdio.h>

void fibonacci\_iterative(int n) {

int a = 0, b = 1, temp;

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

printf("%d ", a);

temp = a + b;

a = b;

b = temp;

}

}

int main() {

int n;

printf("Enter number of terms: ");

scanf("%d", &n);

printf("Fibonacci Series (Iterative): ");

fibonacci\_iterative(n);

return 0;

}

1. Factorial:

#include <stdio.h>

long long factorial(int n) {

return (n <= 1) ? 1 : n \* factorial(n - 1);

}

int main() {

int n;

printf("Enter a number: ");

scanf("%d", &n);

printf("Factorial (Recursive) of %d = %lld\n", n, factorial(n));

return 0;

}

#include <stdio.h>

long long factorial\_iterative(int n) {

long long fact = 1;

for (int i = 2; i <= n; i++)

fact \*= i;

return fact;

}

int main() {

int n;

printf("Enter a number: ");

scanf("%d", &n);

printf("Factorial (Iterative) of %d = %lld\n", n, factorial\_iterative(n));

return 0;

}

11 .HeapSort

#include <stdio.h>

// Function to heapify a subtree rooted at index i

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

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;

if (largest != i) {

int temp = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

heapify(arr, n, largest);

}

}

// Heap Sort function

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

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

heapify(arr, n, i);

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

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, i, 0);

}

}

int main() {

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

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

printf("Original array: ");

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

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

printf("\n");

heapSort(arr, n);

printf("Sorted array: ");

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

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

printf("\n");

return 0;

}