**SEARCHING**

**1)C PROGRAM TO PERFORM A LINEAR SEARCH ON AN ARRAY OF INTEGERS.**

**CODE:**

#include <stdio.h>

int linearSearch(int arr[], int size, int target) {

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

if (arr[i] == target) {

return i;

}

}

return -1;

}

int main() {

int size, target, result;

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

scanf("%d", &size);

int arr[size];

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

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

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

}

printf("Enter the target element: ");

scanf("%d", &target);

result = linearSearch(arr, size, target);

if (result != -1) {

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

}

else {

printf("Element not found in the array\n");

}

return 0;

}

**2) IMPLEMENT A BINARY SEARCH TO FIND THE TARGET VALUE IN A SORTED ARRAY.**

**CODE:**

#include <stdio.h>

int binarySearch(int arr[], int size, int target) {

int left = 0, right = size - 1;

while (left <= right) {

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

if (arr[mid] == target) {

return mid;

}

if (arr[mid] < target) {

left = mid + 1;

}

else {

right = mid - 1;

}

}

}

int main() {

int size, target, result;

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

scanf("%d", &size);

int arr[size];

printf("Enter the elements of the sorted array in ascending order:\n");

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

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

}

printf("Enter the target element: ");

scanf("%d", &target);

result = binarySearch(arr, size, target);

if (result != -1) {

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

}

else {

printf("Element not found in the array\n");

}

return 0;

}

**SORTING**

**1) GIVE AN ARRAY OF INTEGERS, IMPLEMENT BUBBLE SORT TO SORT IT IN ASCENDING ORDER.**

**CODE:**

#include <stdio.h>

void printArr(int\* a, int n) {

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

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

}

printf("\n");

}

void bubbleSort(int\* a, int n) {

int temp;

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

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

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

temp = a[j];

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

a[j + 1] = temp;

}

}

}

}

int main() {

int a[] = {5, 3, 8, 4, 2};

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

printf("Original array:\n");

printArr(a, n);

bubbleSort(a, n);

printf("Sorted array:\n");

printArr(a, n);

return 0;

}

**2) WRITE A C PROGRAM TO SORT AN ARRAY OF INTEGERS IN DESCENDING 0RDER USING SELECTION SORT.**

**CODE:**

#include <stdio.h>

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

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

int maxIdx = i;

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

if (arr[j] > arr[maxIdx]) {

maxIdx = j;

}

}

int temp = arr[i];

arr[i] = arr[maxIdx];

arr[maxIdx] = temp;

}

}

int main() {

int arr[] = {5, 3, 8, 4, 2};

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

selectionSortDescending(arr, n);

printf("Sorted array in descending order: ");

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

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

}

return 0;

}

**3) IMPLEMENT INSERTION SORT TO SORT AN ARRAY OF INTEGERS IN ASCENDING ORDER.**

**CODE:**

#include <stdio.h>

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

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

int key = arr[i];

int j = i - 1;

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

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

j = j - 1;

}

arr[j + 1] = key;

}

}

int main() {

int arr[] = {5, 3, 8, 4, 2};

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

insertionSort(arr, n);

printf("Sorted array in ascending order: ");

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

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

}

return 0;

}

**4) IMPLEMENT MERGE SORT TO SORT AN ARRAY OF INTEGERS.**

**CODE:**

#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int L[n1], R[n2];

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

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

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

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

int i = 0, j = 0, k = left;

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 left, int right) {

if (left < right) {

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

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

int main() {

int arr[] = {5, 3, 8, 4, 2};

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

mergeSort(arr, 0, n - 1);

printf("Sorted array in ascending order: ");

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

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

}

return 0;

}

**5) IMPLEMENT QUICK SORT TO SORT AN ARRAY OF INTEGERS.**

**CODE:**

#include <stdio.h>

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

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

int pivot = arr[high];

int i = (low - 1);

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

if (arr[j] < pivot) {

i++;

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

}

}

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

return (i + 1);

}

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

if (low < high) {

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

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main() {

int arr[] = {5, 3, 8, 4, 2};

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

quickSort(arr, 0, n - 1);

printf("Sorted array in ascending order: ");

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

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

}

return 0;

}

**6) IMPLEMENT HEAP SORT TO SORT AN ARRAY OF INTEGERS.**

**CODE:**

#include <stdio.h>

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);

}

}

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[] = {5, 3, 8, 4, 2};

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

heapSort(arr, n);

printf("Sorted array in ascending order: ");

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

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

}

return 0;

}