Sequential Search:

#include <iostream>

using namespace std;

int search(int array[], int n, int x) {

// Going through array sequencially

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

if (array[i] == x)

return i;

return -1;

}

int main() {

int array[] = {2, 4, 0, 1, 9};

int x = 1;

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

int result = search(array, n, x);

(result == -1) ? cout << "Element not found" : cout << "Element found at index: " << result;

}

---

Element found at index: 3%

Time Complexity: O(n)

---

Binary Search:

#include <iostream>

using namespace std;

int binarySearch(int array[], int x, int low, int high) {

if (high >= low) {

int mid = low + (high - low) / 2;

// If found at mid, then return it

if (array[mid] == x)

return mid;

// Search the left half

if (array[mid] > x)

return binarySearch(array, x, low, mid - 1);

// Search the right half

return binarySearch(array, x, mid + 1, high);

}

return -1;

}

int main(void) {

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

int x = 4;

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

int result = binarySearch(array, x, 0, n - 1);

if (result == -1)

printf("Not found");

else

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

}

----

Element is found at index 1%

Time Complexity: O(1)

---

Bubble Sort

#include <iostream>

using namespace std;

// perform bubble sort

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

// loop to access each array element

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

// check if swapping occurs

int swapped = 0;

// loop to compare two elements

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

// compare two array elements

// change > to < to sort in descending order

if (array[i] > array[i + 1]) {

// swapping occurs if elements

// are not in intended order

int temp = array[i];

array[i] = array[i + 1];

array[i + 1] = temp;

swapped = 1;

}

}

// no swapping means the array is already sorted

// so no need of further comparison

if (swapped == 0)

break;

}

}

// print an array

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

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

cout << " " << array[i];

}

cout << "\n";

}

int main() {

int data[] = {-2, 45, 0, 11, -9};

// find the array's length

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

bubbleSort(data, size);

cout << "Sorted Array in Ascending Order:\n";

printArray(data, size);

}

---

Sorted Array in Ascending Order:

-9 -2 0 11 45

Time Complexity: O(n)

---

Selection Sort:

#include <iostream>

using namespace std;

// function to swap the the position of two elements

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

// function to print an array

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

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

cout << array[i] << " ";

}

cout << endl;

}

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

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

int min\_idx = step;

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

// To sort in descending order, change > to < in this line.

// Select the minimum element in each loop.

if (array[i] < array[min\_idx])

min\_idx = i;

}

// put min at the correct position

swap(&array[min\_idx], &array[step]);

}

}

// driver code

int main() {

int data[] = {20, 12, 10, 15, 2};

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

selectionSort(data, size);

cout << "Sorted array in Acsending Order:\n";

printArray(data, size);

}

---

Sorted array in Acsending Order:

2 10 12 15 20

Time Complexity: O(n^2)

---

Insertion Sort:

#include <iostream>

using namespace std;

// Function to print an array

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

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

cout << array[i] << " ";

}

cout << endl;

}

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

for (int step = 1; step < size; step++) {

int key = array[step];

int j = step - 1;

// Compare key with each element on the left of it until an element smaller than

// it is found.

// For descending order, change key<array[j] to key>array[j].

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

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

--j;

}

array[j + 1] = key;

}

}

// Driver code

int main() {

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

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

insertionSort(data, size);

cout << "Sorted array in ascending order:\n";

printArray(data, size);

}

-----

Sorted array in ascending order:

1 3 4 5 9

Time Complexity: O(n)