1. Linear search:

#include <stdio.h>

int main()

{

int array[100], search, c, n;

printf("Enter number of elements in array\n");

scanf("%d", &n);

printf("Enter %d integer(s)\n", n);

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

scanf("%d", &array[c]);

printf("Enter a number to search\n");

scanf("%d", &search);

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

{

if (array[c] == search) /\* If required element is found \*/

{

printf("%d is present at location %d.\n", search, c+1);

break;

}

}

if (c == n)

printf("%d isn't present in the array.\n", search);

return 0;

}

Time Complexity: O(n)

Space Complexity: O(1)

2. Binary search:

#include <stdio.h>

int main()

{

int c, first, last, middle, n, search, array[100];

printf("Enter number of elements\n");

scanf("%d", &n);

printf("Enter %d integers\n", n);

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

scanf("%d", &array[c]);

printf("Enter value to find\n");

scanf("%d", &search);

first = 0;

last = n - 1;

middle = (first+last)/2;

while (first <= last) {

if (array[middle] < search)

first = middle + 1;

else if (array[middle] == search) {

printf("%d found at location %d.\n", search, middle+1);

break;

}

else

last = middle - 1;

middle = (first + last)/2;

}

if (first > last)

printf("Not found! %d isn't present in the list.\n", search);

return 0;

}

Time Complexities

Best case complexity: O(1)

Average case complexity: O(log n)

Worst case complexity: O(log n)

Space Complexity

The space complexity of the binary search is O(1).

3. Quick Sort:

#include<stdio.h>

void quicksort(int number[25],int first,int last){

int i, j, pivot, temp;

if(first<last){

pivot=first;

i=first;

j=last;

while(i<j){

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

i++;

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

j--;

if(i<j){

temp=number[i];

number[i]=number[j];

number[j]=temp;

}

}

temp=number[pivot];

number[pivot]=number[j];

number[j]=temp;

quicksort(number,first,j-1);

quicksort(number,j+1,last);

}

}

int main(){

int i, count, number[25];

printf("Enter some elements (Max. - 25): ");

scanf("%d",&count);

printf("Enter %d elements: ", count);

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

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

quicksort(number,0,count-1);

printf("The Sorted Order is: ");

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

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

return 0;

}

Time Complexity

Best O(n\*log n)

Worst O(n2)

Average O(n\*log n)

Space Complexity O(log n)

Stability No

4. Merge sort:

// Merge sort in C

#include <stdio.h>

// Merge two subarrays L and M into arr

void merge(int arr[], int p, int q, int r) {

// Create L ← A[p..q] and M ← A[q+1..r]

int n1 = q - p + 1;

int n2 = r - q;

int L[n1], M[n2];

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

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

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

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

// Maintain current index of sub-arrays and main array

int i, j, k;

i = 0;

j = 0;

k = p;

// Until we reach either end of either L or M, pick larger among

// elements L and M and place them in the correct position at A[p..r]

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

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

arr[k] = L[i];

i++;

} else {

arr[k] = M[j];

j++;

}

k++;

}

// When we run out of elements in either L or M,

// pick up the remaining elements and put in A[p..r]

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = M[j];

j++;

k++;

}

}

// Divide the array into two subarrays, sort them and merge them

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

if (l < r) {

// m is the point where the array is divided into two subarrays

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

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

// Merge the sorted subarrays

merge(arr, l, m, r);

}

}

// Print the array

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

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

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

printf("\n");

}

// Driver program

int main() {

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

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

mergeSort(arr, 0, size - 1);

printf("Sorted array: \n");

printArray(arr, size);

}

Time Complexity

Best O(n\*log n)

Worst O(n\*log n)

Average O(n\*log n)

Space Complexity O(n)

Stability Yes

5.Selection Sort:

// Selection 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 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]);

}

}

// function to 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[] = {20, 12, 10, 15, 2};

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

selectionSort(data, size);

printf("Sorted array in Acsending Order:\n");

printArray(data, size);

}

Selection Sort Complexity

Time Complexity

Best O(n2)

Worst O(n2)

Average O(n2)

Space Complexity O(1)

Stability No

6. Insertion Sort:

// Insertion sort in C

#include <stdio.h>

// Function to print an array

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

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

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

}

printf("\n");

}

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);

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

printArray(data, size);

}

Insertion Sort Complexity

Time Complexity

Best O(n)

Worst O(n2)

Average O(n2)

Space Complexity O(1)

Stability Yes

7. Bubble sort:

/ Bubble sort in C

#include <stdio.h>

// perform the bubble sort

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

// loop to access each array element

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

// loop to compare array elements

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

// compare two adjacent elements

// change > to < to sort in descending order

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

// swapping occurs if elements

// are not in the intended order

int temp = array[i];

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

array[i + 1] = temp;

}

}

}

}

// print array

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

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

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

}

printf("\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);

printf("Sorted Array in Ascending Order:\n");

printArray(data, size);

}

Bubble Sort Complexity

Time Complexity

Best O(n)

Worst O(n2)

Average O(n2)

Space Complexity O(1)

Stability Yes

8. 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);

}

Heap Sort Complexity

Time Complexity

Best O(nlog n)

Worst O(nlog n)

Average O(nlog n)

Space Complexity O(1)

Stability No

9. UPPer case:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int main()

{

char y,x;

printf("enter a letter\n");

x=getchar();

y=toupper(x);

putchar(y);

return 0;

}

10. #include<stdio.h>

long factorial (int n)

{

long result;

if (n==0)

return 1;

else

{

result=n \* factorial(n-1);

return result;

}

}

int main()

{

int x;

long fact;

scanf("%d", &x);

fact=factorial(x);

printf("%d", fact);

return 0;

}

11.