**Searching**

1. **Linear Search**

#include <stdio.h>

void linearSearch(int \*arr, int n, int key){

int found=0;

int i;

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

{

if (key == arr[i])

{

found = 1;

break;

}

}

if (found == 1)

printf("Element is present in the array at position %d",i+1);

else

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

}

void main()

{ int num;

int i, key;

printf("Enter the number of elements ");

scanf("%d", &num);

int arr[num];

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

{

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

}

printf("Enter the element to be searched ");

scanf("%d", &key);

linearSearch(arr,num,key);

}

Analysis of linear search

Worst case:

O(n)

Best case:

O(1)

1. **Binary Search**

#include <stdio.h>

void binarySearch(int \*arr, int n, int key){

int low = 0;

int high = n - 1;

int mid = (low+high)/2;

while (low <= high) {

if(arr[mid] < key)

low = mid + 1;

else if (arr[mid] == key) {

printf("%d found at location %d", key, mid+1);

break;

}

else

high = mid - 1;

mid = (low + high)/2;

}

if(low > high)

printf("Not found! %d isn't present in the list", key);

}

int main()

{

int i, low, high, mid, n, key, arr[100];

printf("Enter number of elements:");

scanf("%d",&n);

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

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

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

printf("Enter value to find");

scanf("%d", &key);

binarySearch(arr,n,key);

return 0;

}

Analysis of Binary Search

Worst case:

O(log n)

Best case:

The best case of binary search is when the first comparison/guess is correct(the key item is equal to the mid of array). It means, regardless of the size of the list/array, we'll always get the result in constant time. So the best case complexity is **O(1)**.

**Sorting**

1. **Bubble sort**

#include <stdio.h>

void printArray(int \*arr,int n){

printf("Array Elements are:");

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

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

}

}

int bubbleSort(int \*arr, int n){

int temp=0;

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

{

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

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

temp=arr[j];

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

arr[j+1]=temp;

}

}

}

}

int main()

{

int arr[100];

printf("Input Array Elements:");

for(int i=0;i<10;i++){

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

}

printArray(arr,10);

bubbleSort(arr,10);

printArray(arr,10);

return 0;

}

Analysis of Bubble sort

Worst Case:

Len = n

Number of passes = n-1

Total number of comparisons = n-1 + n-2 + n-3 + …. + 3 + 2 + 1

n(n-1)/2 = O(n2)

Best case:

It will do one pass and do all the comparison through all the elements so it is O(n)

If an array is already in sorted order, and bubble sort makes no swaps, the algorithm can terminate after one pass.

1. **Insertion Sort**

#include <stdio.h>

void printArray(int \*arr,int n){

printf("Array Elements are:");

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

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

}

}

int insertionSort(int \*arr, int n){

int j, i, key;

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

{

key=arr[i];

j=i-1;

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

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

j--;

}

arr[j+1]=key;

}

}

int main()

{

int arr[100];

printf("Input Array Elements:");

for(int i=0;i<10;i++){

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

}

printArray(arr,10);

insertionSort(arr,10);

printArray(arr,10);

return 0;

}

Analysis of Selection sort

Worst Case:

Len = n

Total number of passes = n-1

Total Possible Comparison/ swap = 1 + 2 + 3 + …. + n-1

n(n-1)/2 = O(n2)

Best case:

n-1 = O(n)

The best case input is an array that is already sorted. In this case insertion sort has a linear running time (i.e., O(**n**)). During each iteration, the first remaining element of the input is only compared with the right-most element of the sorted subsection of the array.

1. **Selection Sort**

#include <stdio.h>

void printArray(int \*arr,int n){

printf("Array Elements are:");

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

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

}

}

int selectionSort(int \*arr, int n){

int indexOfMin, temp;

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

{

indexOfMin=i;

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

if(arr[j]<arr[indexOfMin]){

indexOfMin=j;

}

}

temp=arr[i];

arr[i]=arr[indexOfMin];

arr[indexOfMin]=temp;

}

}

int main()

{

int arr[100];

printf("Input Array Elements:");

for(int i=0;i<10;i++){

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

}

printArray(arr,10);

selectionSort(arr,10);

printArray(arr,10);

return 0;

}

Analysis of Selection sort

Worst Case:

Len = n

Number of passes = n-1

In each pass total number of comparison = 1 + 2 + 3 + …. + n-1

n(n-1)/2 = O(n2)

Best case:

O(n2)