**DAY -18**

**ALGORITHMS**

**Searching Algorithms:**

1) Linear Search:

--> Time process is high

-->Checks each and every element

2) Interval Search: dividing into two half and starts searching(divide and concurring)

-->we can perform in sorted list

-->the no of iterations are less so time consuming process is less

Ex: Binary Search

**Linear Search:**

In Linear Search, we iterate over all the elements of the array and check if it the current element is equal to the target element. If we find any element to be equal to the target element, then return the index of the current element. Otherwise, if no element is equal to the target element, then return -1 as the element is not found. Linear search is also known as **sequential search**.

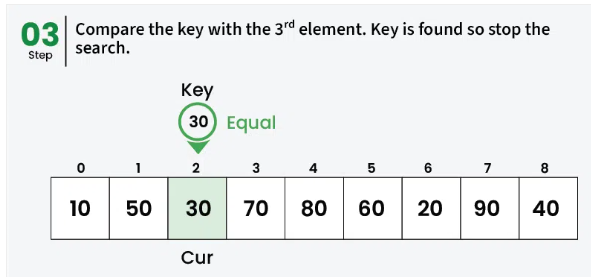
**For example:** Consider the array **arr[] = {10, 50, 30, 70, 80, 20, 90, 40}** and **key** = 30

A screenshot of a math test

Description automatically generated

A white paper with black text and numbers

Description automatically generated



**Binary Search:**

1.Iterative Method

2.Recursive Method

🡪The recursive method follows the divide and conquer approach.

🡪Binary Search Algorithm is a [searching algorithm](https://www.geeksforgeeks.org/searching-algorithms/) used in a sorted array by repeatedly dividing the search interval in half.

A screenshot of a computer

Description automatically generated

🡪**Binary Search using iterative method:**

We have two types:

1) iterative

2) recursive

It follows divide and conquer approach

- find the middle element mid of the array è [(low+high]/2]

- If x==mid then return mid else compare the element to be searched with m

- If x>mid, compare x with mid element

**Iteration method**

Do until the pointers low and high meet each other

Mid = (low+high)/2

If(x==arr[mid])

Return mid

Elif(x>arr[mid]) // x is on right side

Low = mid+1

Else // x is on left side

High = mid-1

Now doing it using dynamic memory allocation

BinarySearch(arr,x,low,high)

**SORTING**

Three basic design goals:

**1) Time complexity**

T(n) = k1\*(n^0)+k2+k3+k4+k5

= n(0)

= O(1)

S=d/t

T=d/s 250kb/20kb // ex

T(n)=d/n=n` \*d = O(n`)

Where n` = 1/n

**2) Space complexity**

**Sorting types:**

1) Bubble sort

2) Merge sort

3) Insertion

4) Quick

**Sorting**

Space used Stability

In place Out place Stable Unstable

Ex : Bubble Ex : Merge Ex : Insertion Ex : Quick

- In place does not require any extra space for sorting

- Out place requires an extra space for sorting

- In stable sorting the position of the container must not change

70 10 80 40 50 40 20

10 20 40 40 50 70 80

- In unstable sorting the position of the container change

70 10 80 40 50 40 20

10 20 40 40 50 70 80

**BUBBLE SORT**

- It is also known as Sinking sort

- We repeatedly compare each pair of adjacent items and swap them if they are in the wrong order

- If the first iteration is in the increasing order then last element(MAX) will be in right position

**To swap two elements in an array using bubble sort**

#include <stdio.h>

// Function to swap two elements

void swap(int \*x, int \*y) {

int temp = \*x;

\*x = \*y;

\*y = temp;

}

// Bubble Sort function

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

// Outer loop for number of passes

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

// Inner loop for each pass comparison

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

// Swap if the element is greater than the next element

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

swap(&arr[j], &arr[j + 1]);

}

}

}

}

**// Function to print the array**

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

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

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

}

printf("\n");

}

int main() {

int arr[] = {64, 34, 25, 12, 22, 11, 90};

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

printf("Original array: \n");

printArray(arr, n);

**// Call bubble sort**

bubbleSort(arr, n);

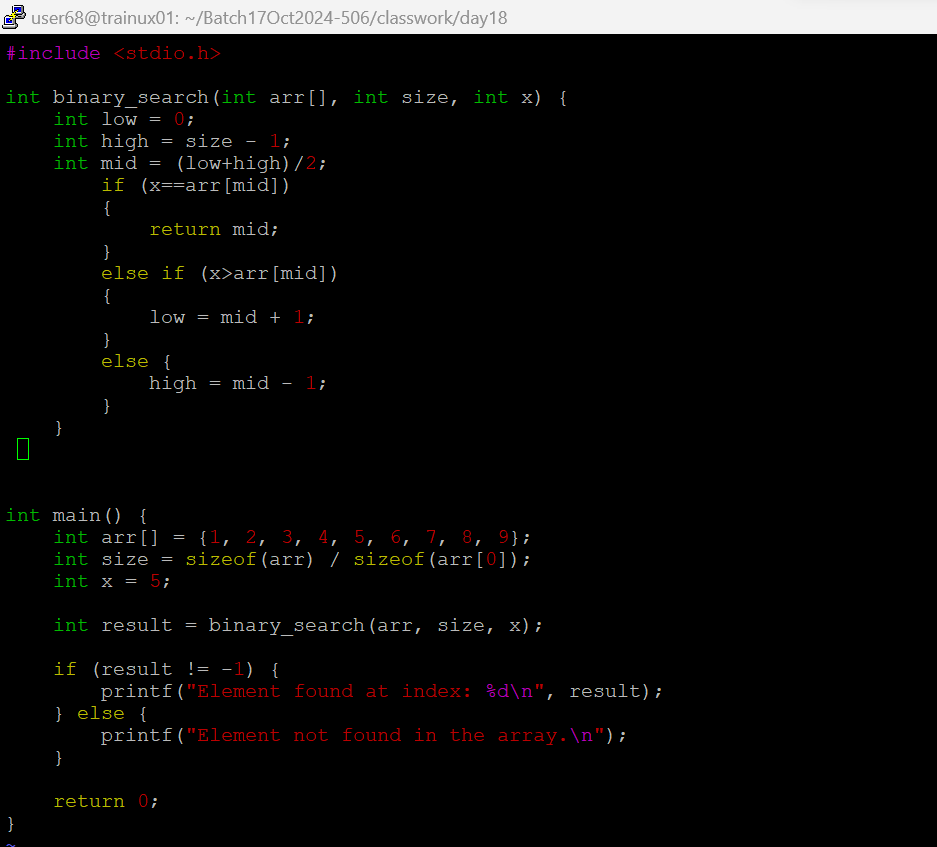
printf("Sorted array: \n");

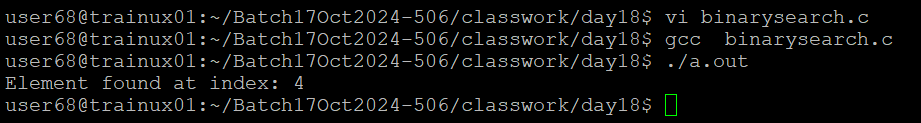
printArray(arr, n);

return 0;

}

Bubble will sort from right to left….Selection will sort from left to right





* **Binary search Using Recursive Method**

**Bubble**

**Selection**

**🡪insestion: (stable sort) the position of the sorting should not be changed.**

**🡪quick sort: (unstable sort)**

**🡪bubble sort : also referred as sinking sort. Swap them if it is in wrong order**