Part A

Aim:

1. Design and analysis algorithms for following problems.
   Linear Search, Binary Search
2. Understand best-case, average-case and worst-case complexities

Prerequisite: Any programming language

Outcome: Algorithms and their implementation

Theory:

Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored. Based on the type of search operation, these algorithms are generally classified into two categories:

1. Sequential Search: In this, the list or array is traversed sequentially and every element is checked. For example: Linear Search.
2. Interval Search: These algorithms are specifically designed for searching in sorted data-structures. These type of searching algorithms are much more efficient than Linear Search as they repeatedly target the center of the search structure and divide the search space in half. For Example: Binary Search.

Cases to analyse an algorithm:

1) The Worst Case
2) Average Case
3) Best Case

Worst Case Analysis

In the worst-case analysis, we calculate upper bound on running time of an algorithm. We must know the case that causes maximum number of operations to be executed. For Linear Search, the worst case happens when the element to be searched (x in the above code) is not present in the array. When x is not present, the search() functions compares it with all the elements of arr[] one by one. Therefore, the worst-case time complexity of linear search would be Θ(n).

Average Case Analysis

In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by total number of inputs. We must know (or predict) distribution of cases. For the linear search problem, let us assume that all cases are uniformly distributed (including the case of x not being present in array). So we sum all the cases and divide the sum by (n+1). Following is the value of average case time complexity.

Average Case Time =

$$\frac{\sum_{i=1}^{n+1} \theta(i)}{(n+1)}$$

$= \frac{\theta((n+1)*(n+2)/2)}{(n+1)}$

$= \Theta(n)$

Best Case Analysis (Bogus)

In the best-case analysis, we calculate lower bound on running time of an algorithm. We must know the case that causes minimum number of operations to be executed. In the linear search problem, the best case occurs when x is present at the first location. The number of operations in the best case is constant (not dependent on n). So time complexity in the best case would be $\Theta(1)$

Procedure:

1. Design algorithm and find best, average and worst-case complexity
2. Implement algorithm in any programming language.
3. Paste output

Practice Exercise:

S.no	Query statement
1	Design and analysis algorithm for Linear Search
2	Design and analysis algorithm for Binary Search.
3	For both the algorithm, give case-wise complexity

Instructions:

1. Design, analysis and implement the algorithms.
2. Paste the snapshot of the output in input & output section.

Part B

Algorithm: Linear search

Input : List/ array of n number of elements and the element to be searched.

Output:Print Index of search element if found else element not found.

```
start
LinearSearch(arr[],sear):
     i=0;
     for( i=0;i<n;i++)
          if( sear==arr[i])
               print element found at index i
               break
     if(i==n-1)
          print element not found
```

CODE:

```java
import java.util.Scanner;
class Linearsearch{
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        int sear=sc.nextInt();
        int arr[]=new int[n];
        int i,flag;
        for(i=0;i<n;i++){
            arr[i]=sc.nextInt();
        }
        for(i=0;i<n;i++){
            if(sear==arr[i]){
                break;
            }
            if(i==n-1){
                System.out.println("Element not found");
                return ;
            }
        }
        System.out.println("element found at index "+i);



    }
}
```

Input & Output:

```
Enter size of array
6
Enter search element
3
Enter array elements
54 21 3 78 66 11
element found at index 2
```

```
Enter size of array
6
Enter search element
3
Enter array elements
4 32 11 67 34 99
Element not found
```

Time complexity:

Best case time complexity:

If in case the search element is 1st element then the number of comparisons would be only 1 as the for loop will run only once. So the best case time complexity for the above algorithm is O(1).

Worst case time complexity:

If in case the search element is in the last position then the number of comparisons would be n as for loop runs n times . So worst case time complexity would be O(n).

Average case time complexity:

Average case time complexity=(Complexity of all possible cases)/ number of cases

So for n elements all cases of finding an element would be 1,2,3,4,5….n

Therefore the formula becomes, (1+2+3+4+5+....+n)/n = n(n+1)/2n= (n+1)/2 =O(n).

Therefore, Average case time complexity is almost equal to the worst case time complexity so the average case time complexity will be O(n).

2)Algorithm: Binary search

Input: List/ array of n number of elements and the element to be searched.

Output: Print Index of search element if found else element not found.

```
Start
  BinarySearch(arr[] , sear):
   start=0
   end=  size - 1
   while(start  <  end):
              mid = (start + end)/2
              if( sear = = arr[mid])
                        return mid;
         else if( sear <  arr[mid] )
                        high = mid - 1
          else if(sear > arr[mid])
                    low = mid + 1
  return -1;
End.
```

Code:

```java
import java.util.Scanner;
class Binarysearch{
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter size of array");
        int n=sc.nextInt();
        System.out.println("Enter search element");
        int sear=sc.nextInt();
        int arr[]=new int[n];
        int s=0,e=n-1,m,i,flag;
        System.out.println("Enter array elements");
        for(i=0;i<n;i++){
            arr[i]=sc.nextInt();
```

```
        }
        m=(s+e)/2;
        while(s<=e){
            if(arr[m]==sear){
                System.out.println("element found at index "+m);
                break;

            }
            else if(sear>arr[m])
            {
                s=m+1;
            }
            else if(sear<arr[m]){
                e=m-1;
            }
            m=(s+e)/2;
        }
        if(s>e){
            System.out.println("Search element not found");
        }

    }
}
```

Input and Output:

```
Enter size of array
6
Enter search element
44
Enter array elements
23 24 27 34 37 38
Search element not found
```

```
Enter size of array
6
Enter search element
45
Enter array elements
34 36 38 41 45 55
element found at index 4
```

Time complexities:

Best case time complexity:

If in case the search element is the first element in the array then it takes only one iteration . so, the best case time complexity for linear search is O(1).

Worst case time complexity:

If in case search element is at last position and it took k iterations to search the element then

n/2^(k-1) = 1

n = 2^(k-1)

Applying log on both sides gives , log n= log 2^(k-1)

log n=k-1 log 2
k-1=log n
k=(log n)+1 → O(log n)
So time complexity is O(log n) at worst case

Average case time complexity: (Complexity of all possible cases)/ number of cases
Average case time complexity is almost equal to the worst case time complexity so the average case time complexity will be O(n).

Observation & Learning:
I have observed that best case and worst case time complexities will not be the same for all algorithms .It varies from algorithm to algorithm.
I have also observed that average time complexity is almost equal to the worst case time complexity for most of the algorithms.

Conclusion:
I have learned how to find best case , average case and worst case time complexities of an algorithm and implemented the algorithms of linear search and binary search in java programming language.

Questions:

1. Out of best-case, average-case and worst-case complexity of an algorithm, which one is most useful for analysis of an algorithm and why?

Answer

1) Worst case time complexity is most useful for analysis of an algorithm because if we prepare for the worst thing to happen we can handle any other situation with ease.
So we need to design algorithms which perform good in the worst case and those algorithms will automatically work good in best case.