Experiment no:- 4

Name:- Swapnil Vijaykumar Pamu

Roll No:- 76

1) Title:- Binary Search Algorithm.

2) Theory of the Problem Statement:

Binary search is an efficient algorithm for finding an item in a sorted array. By repeatedly dividing the search interval in half, it significantly reduces the number of comparisons needed. If the target value matches the middle element, the search is complete; otherwise, the search continues in the appropriate half of the array until the element is found or the interval is empty. This efficiency makes binary search much faster than linear search for large datasets.

3) Algorithm in Pseudo Code format:-

-Pseudo code for bubble sort

```
1)  BinarySearch(A, n, x)
2)    low ← 0
3)    high ← n - 1
4)    while low ≤ high do
5)       mid ← (low + high) / 2
6)       if A[mid] = x then
7)          return mid
8)       else if A[mid] < x then
9)          low ← mid + 1
10)      else
11)         high ← mid - 1
12)   return -1
```

4) Analysis of the Algorithm: -

-Calculating the running time function:-

Best case :-

Number of comparison if target 'k' is found at mid then.

- Number of comparison

$T(n) = 1$

- Running Time :-

$T(n) = O(1)$

- Worst case Analysis

1. Initial Iterative size :- n
2. First Iterative :- $\frac{n}{2}$
3. Second Iterative :- $\frac{n}{4}$
4. K^{th} Iterative :- $\frac{n}{2^k}$

Continuous until interval size = 1.

$\therefore \frac{n}{2^k} = 1$

$\therefore n = 2^k$

Applying $\log_2 n$ both side

$k = \log_2 n$

$T(n) = \log_2 n$

Running Time function

$T(n) = O(\log n)$

5) Experiment and result:

Merge Sort (Code)

```
public class Main {
   public static int binarySearch(int[] arr, int n, int k) {
      int low = 0;
      int high = n - 1;
      while (low <= high) {
         int mid = (low + high) / 2;
         if (arr[mid] == k) {
            return mid;
         } else if (arr[mid] < k) {
            low = mid + 1;
         } else {
            high = mid - 1;
         }
      }
      return -1;
   }

   public static void main(String[] args) {
      int[] arr = {2, 3, 4, 10, 40};
      int n = arr.length;
      int k = 10;
      int result = binarySearch(arr, n, k);
      if (result != -1) {
         System.out.println("Element is present at index " +
(result+1));
      } else {
         System.out.println("Element is not present in array");
      }
   }
}
```

• **Result**

```
C:\Users\swapn\.jdks\openjdk-21.0.2\bin\java.exe "
Element is present at index 4

Process finished with exit code 0
```

6) Conclusion:

Merge Sort is a highly effective and reliable sorting algorithm renowned for its O(n log n) time complexity. Its divide-and-conquer approach ensures stable and efficient sorting even for large datasets. This makes it a popular choice in applications requiring both stable sorting and optimal performance.

Experiment no:- 4

Name:- Swapnil Vijaykumar Pamu

Roll No:- 76

1) Title:- Binary Search Algorithm.

2) Theory of the Problem Statement:

Binary search is an efficient algorithm for finding an item in a sorted array. By repeatedly dividing the search interval in half, it significantly reduces the number of comparisons needed. If the target value matches the middle element, the search is complete; otherwise, the search continues in the appropriate half of the array until the element is found or the interval is empty. This efficiency makes binary search much faster than linear search for large datasets.

3) Algorithm in Pseudo Code format:-

-Pseudo code for Binary Search

```
1)  BinarySearch(A, n, x)
2)    low ← 0
3)    high ← n - 1
4)    while low ≤ high do
5)       mid ← (low + high) / 2
6)       if A[mid] = x then
7)          return mid
8)       else if A[mid] < x then
9)          low ← mid + 1
10)      else
11)         high ← mid - 1
12)   return -1
```

4) Analysis of the Algorithm: -

-Calculating the running time function:-

Best case :-

Number of comparison if target 'k' is found at mid then.

- Number of comparison

$T(n) = 1$

- Running Time :-

$T(n) = O(1)$

- Worst case Analysis

1. Initial Iterative size :- n
2. First Iterative :- $\frac{n}{2}$
3. Second Iterative :- $\frac{n}{4}$
4. K^{th} Iterative :- $\frac{n}{2^k}$

Continuous until interval size = 1.

$\therefore \frac{n}{2^k} = 1$

$\therefore n = 2^k$

Applying $\log_2 n$ both side

$k = \log_2 n$

$T(n) = \log_2 n$

Running Time function

$T(n) = O(\log n)$

5) Experiment and result:

Binary Search(Code)

```
public class Main {
   public static int binarySearch(int[] arr, int n, int k) {
      int low = 0;
      int high = n - 1;
      while (low <= high) {
         int mid = (low + high) / 2;
         if (arr[mid] == k) {
            return mid;
         } else if (arr[mid] < k) {
            low = mid + 1;
         } else {
            high = mid - 1;
         }
      }
      return -1;
   }

   public static void main(String[] args) {
      int[] arr = {2, 3, 4, 10, 40};
      int n = arr.length;
      int k = 10;
      int result = binarySearch(arr, n, k);
      if (result != -1) {
         System.out.println("Element is present at index " +
(result+1));
      } else {
         System.out.println("Element is not present in array");
      }
   }
}
```

• **Result**

```
C:\Users\swapn\.jdks\openjdk-21.0.2\bin\java.exe "
Element is present at index 4

Process finished with exit code 0
```

6) Conclusion:

Binary search is a highly efficient algorithm for finding elements in a sorted array with a time complexity of O(log⁡n)O(\log n)O(logn). It significantly reduces the number of comparisons needed compared to linear search, making it ideal for large datasets.