BINARY SEARCH

Aim- Write a C program to implement Binary Search on array of Strings

Problem Statement – Given a array of strings implement binary search find a given string in the array ,and return the index at which the element was found

INPUT - The number of elements in the array = 9

Array Elements – DOG , CAT ,PIG ,ANT ,ELEPHANT , ZEBRA ,LION ,TIGER ,OX

ELEMENT TO BE FOUND – OX

OUTPUT – the element was found at index 5

ALGORITHM –

I] Algorithm BinarySearch (low,high,x)

//Given a global array a[i:l] of elements in assending order ,i<=x<=l, determine whether x is present

// if present return low ,such that x=a[low] else return -1

{

if(low=high) then {

if(a[low]= x) then {return low ; }

else { return -1 ; } }

mid := floor((low+high)/2) ;

if(x=a[mid]) then {

return BinarySearch(low,mid-1,x);}

else {return BinarySearch(mid+1,high,x) ;}

}

ii] Algorithm MergeSort (low,high)

```
// Given a global array arr[low:high} and a global temporary array b and 0<=low<=high
{ if (low<=high )  then {
 mid:= floor ((low+high)/2);
MergeSort (low,mid) ;
MergeSort(mid+1,high);
Merge (low,mid,high) ; } }
```

iii] Algorithm Merge (low,mid,high)

```
//Given two global arrays a,b and 0<=low<=mid<=high
{
i:=low ;
j:= mid +1 ;
k:= low ;
while ((i<=mid) and (j<=high)) do {
if ( arr[i] <= arr [j] ) then {
b[k] := arr[i] ;
i:= i+1 ;
} else {
e[k] := arr[j] ;
j := j+1 ;
}
X:= x+1 ; }
While (i<=mid) do {
b[k] : = arr[i] ;
i:= i+1 ; k:=k+1 ;}
while (j<=high) do {
b[k] := arr[j] ;
j:=j+1 ; k:=k+1 ;}}
```

Space and time complexity :

I. Algorithm BinarySearch

Time Complexity:

i) **Best Case:**

- **O(1)**
- This occurs when the target element x is found at the mid-point on the first comparison.

ii) **Worst Case:**

- **O(log n)**
- This happens when the algorithm keeps dividing the array into two halves until the target element is found or determined to be absent.

iii) **Average Case:**

- **O(log n)**
- On average, the binary search requires logarithmic time as the array is halved with each recursive call.

Space Complexity:

i) **Best Case:**

- **O(log n)**
- This accounts for the recursive stack depth in the best-case scenario.

ii) **Worst Case:**

- **O(log n)**
- This happens when the recursion reaches its maximum depth, proportional to the logarithmic size of the array.

iii) **Average Case:**

- **O(log n)**
- On average, the recursive stack grows logarithmically with the size of the array.

II. Algorithm MergeSort

Time Complexity:

i) **Best Case:**

- **O(n log n)**

- Even if the array is already sorted, the algorithm still recursively divides the array and merges it, leading to a time complexity of O(n log n).

ii) **Worst Case:**

- **O(n log n)**
- The algorithm always performs the same number of comparisons and divisions regardless of the input order.

iii) **Average Case:**

- **O(n log n)**
- On average, MergeSort divides the array and merges the sorted parts in logarithmic time for every level, with n operations at each level.

Space Complexity:

i) **Best Case:**

- **O(n)**
- The temporary array b requires linear additional space to store merged elements.

ii) **Worst Case:**

- **O(n)**
- Even in the worst case, the temporary array b is of size n, requiring linear additional space.

iii) **Average Case:**

- **O(n)**
- On average, the same temporary array is used, resulting in linear space usage.

III. Algorithm Merge

Time Complexity:

i) **Best Case:**

- **O(n)**
- Merging two sorted subarrays of total size n requires linear time in all cases.

ii) **Worst Case:**

- **O(n)**
- The merge process is always linear, irrespective of the input.

iii) **Average Case:**

- **O(n)**
- On average, merging two arrays of total size n takes linear time.

Space Complexity:

i) **Best Case:**

- **O(n)**
- A temporary array b of size n is used for merging.

ii) **Worst Case:**

- **O(n)**
- The same temporary array b is required regardless of the case.

iii) **Average Case:**

- **O(n)**
- On average, merging uses linear additional space for the temporary array.

Recurance Equation :

I. Algorithm BinarySearch

The recurrence equation for Binary Search is:

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

II. Algorithm MergeSort

The recurrence equation for MergeSort is:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

III. Algorithm Merge

No recurrence equation exists for the Merge algorithm itself since it is not recursive. Instead, it is part of MergeSort and is a linear process.

PROGRAM –

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#define MAX 15
#define MAX_LEN 100
char a[MAX][MAX_LEN];

void merge(int min, int max) {
   int mid = (min + max) / 2;
   int i = min;
   int j = mid + 1;
   int k = 0;
   char temp[MAX][MAX_LEN];

   while (i <= mid && j <= max) {
      if (strcmp(a[i], a[j]) < 0) {
         strcpy(temp[k], a[i]);
         k++;
         i++;
      } else {
         strcpy(temp[k], a[j]);
         k++;
         j++;
      }
   }

   while (i <= mid) {
      strcpy(temp[k], a[i]);
      k++;
      i++;
   }

   while (j <= max) {
      strcpy(temp[k], a[j]);
      k++;
      j++;
   }

   for (i = min, k = 0; i <= max; i++, k++) {
      strcpy(a[i], temp[k]);
   }
}

void mergesort(int min, int max) {

   int mid;
   if (min < max) {
      mid = (min + max) / 2;
      mergesort(min, mid);
      mergesort(mid + 1, max);
      merge(min, max);
   }

}

int binary_search(char key[], int low, int high, clock_t
start, clock_t *end, double *cpu_time_used) {


   int mid;
   while (low <= high) {
      mid = (low + high) / 2;
      int cmp = strcmp(key, a[mid]);
      if (cmp == 0) {
```

```
            return mid;
        }
        if (cmp < 0) {
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }



    return -1;
}

void display_elements(int n) {
    if (n == 0) {
    printf("Array is empty. Please enter the array first.\n");
    return;
  }
    printf("The elements of the array are: ");
    for (int i = 0; i < n; i++) {


        printf("| %10s ", a[i]);
    }
    printf("|\n");
}


int main() {


     printf
("*****************************************
*********");
    printf ("\n Roll number: 23B-CO-010\n");
    printf (" PR Number - 202311390\n");
    printf("*************************************
***************\n\n\n");


    int i, n=0, result, choice;
    char key[MAX_LEN];
     clock_t start, end;
     double cpu_time_used;


    do {
        printf("\nMenu:\n");
        printf("1. Enter elements of the array\n");
        printf("2. Display elements of the array\n");
        printf("3. Search for an element\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);


        switch (choice) {
            case 1:
                printf("Enter the number of elements in the
array: ");
                scanf("%d", &n);
                for (i = 0; i < n; i++) {
                    printf("Enter element %d: ", i + 1);
                    scanf("%s", a[i]);
                }
                mergesort(0, n - 1);
                break;
            case 2:
            if (n == 0) {
           printf("Array is empty. Please enter the array
first.\n");
           break ;
     }
                display_elements(n);
                break;
            case 3:


             start = clock();
```

```
            printf("Enter the element to be searched: ");
            scanf("%s", key);
            result = binary_search(key, 0, n - 1, start, &end,
&cpu_time_used);
            if (result == -1) {
               printf("Element not found\n");
            } else {
               printf("Element found at index %d\n", result);
            }
             end = clock();
             cpu_time_used = ((double) (end - start)) /
CLOCKS_PER_SEC;
             printf("Time taken by Binary Search: %f
seconds\n", cpu_time_used);
               break;
            case 4:
               printf("Exiting...\n");
               break;
            default:
               printf("Invalid choice. Please try again.\n");
         }
      } while (choice != 4);
      return 0;
}
```

INPUT –

```
**************************************************
 Roll number: 23B-CO-010
 PR Number - 202311390
**************************************************



Menu:
1. Enter elements of the array
2. Display elements of the array
3. Search for an element
4. Exit
Enter your choice: 1
Enter the number of elements in the array: 9
Enter element 1: DOG
Enter element 2: CAT
Enter element 3: PIG
Enter element 4: ANT
Enter element 5: ELEPHANT
Enter element 6: ZEBRA
Enter element 7: LION
Enter element 8: TIGER
Enter element 9: OX

Menu:
1. Enter elements of the array
2. Display elements of the array
3. Search for an element
4. Exit
Enter your choice: 2
The elements of the array are: |       ANT |       CAT |       DOG |   ELEPHANT |      LION |        OX |       PIG |     TIGER |     ZEBRA |

Menu:
1. Enter elements of the array
2. Display elements of the array
3. Search for an element
4. Exit
Enter your choice: 3
Enter the element to be searched: OX
```

OUTPUT –

```
Element found at index 5
```

TIME TAKEN –

```
Time taken by Binary Search: 4.473000 seconds
```

CONCLUSION - Binary search on array of strings was successfully executed without errors