Searching Via Divide and Conquer

Student’s Name

Institutional Affiliation

Searching Via Divide and Conquer

**Part 1**

#include <iostream>

using namespace std;

int main ()

{

//variables and array declarations

 int songs, i, myArray[30], number, top, bottom, centre;

 cout<<"Enter the number of songs in the list:";

 cin>>songs;

 // input of the number of elements

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

 {

 cout<<"Enter song number "<<(i+1)<<"=";

 cin>>myArray[i];

 //input of the songs via the loop.

 }

 cout<<"Which song number do you want to search:";

 cin>>number;

 // this allows you to input the song from the list to be searched within the array

 top = 0;

 bottom = songs 1;

 centre = (top+bottom)/2;

 // initialization and dividing the array into 2 sections for searching purposes

 while (top <= bottom)

 {

 if(myArray[centre] < number)

 {

 top = centre + 1;

 // this section searches the first part of the array

 }

 else if(myArray[centre] == number)

 {

 cout<<"The song number:"<<number<<" found in the Song List at the location "<<centre + 1<<"\n";

//if the song to be searched is in the first section, then it will display a message on the screen that the song has been found

 break;

 }

 else {

 bottom = centre - 1;

 // this searches the last section of the array to find if the song is in the second part

 }

 centre = (top + bottom)/2;

 }

 if(top > bottom)

 {

 cout<<"The song number:"<<number<<" not found in the Song List";

 }

 return 0;

}

**Screenshot of the Results**

**Part 2**

1st step = Time (n) = Time (n / 2) + 1

2nd step = Time (n / 2) =Time (n/4) + 1-----(T (n / 4) = Time (n / 2 ^ 2)

3rd step = Time(n/4) =Time (n / 8) + 1------[ T(n/8) = T(n/2^3)

kth step = Time (n / 2 ^ k-1)=Time (n / 2 ^ k) + 1x(k times)

When we add all the equations, we obtain Time (n) = Time (n / 2 ^ k) + k times 1

= n /2 ^ k= 1 (So how many times we need to divide by 2 until we have only one element remaining)

=> n=2 ^ k

=> log n=k [(taken log (base 2) on both sides)

Put k= log n in e.q. [final]

Time (n) = Time (1) + log n

This algorithm uses is in worst case just like it is the case of binary search algorithm in which its time complexity is at worst logarithmic time, making O (log n) comparisons, where n is the number of elements in the array, the O is Big O notation, and log is the logarithm. Its derivation can be described in the section above.