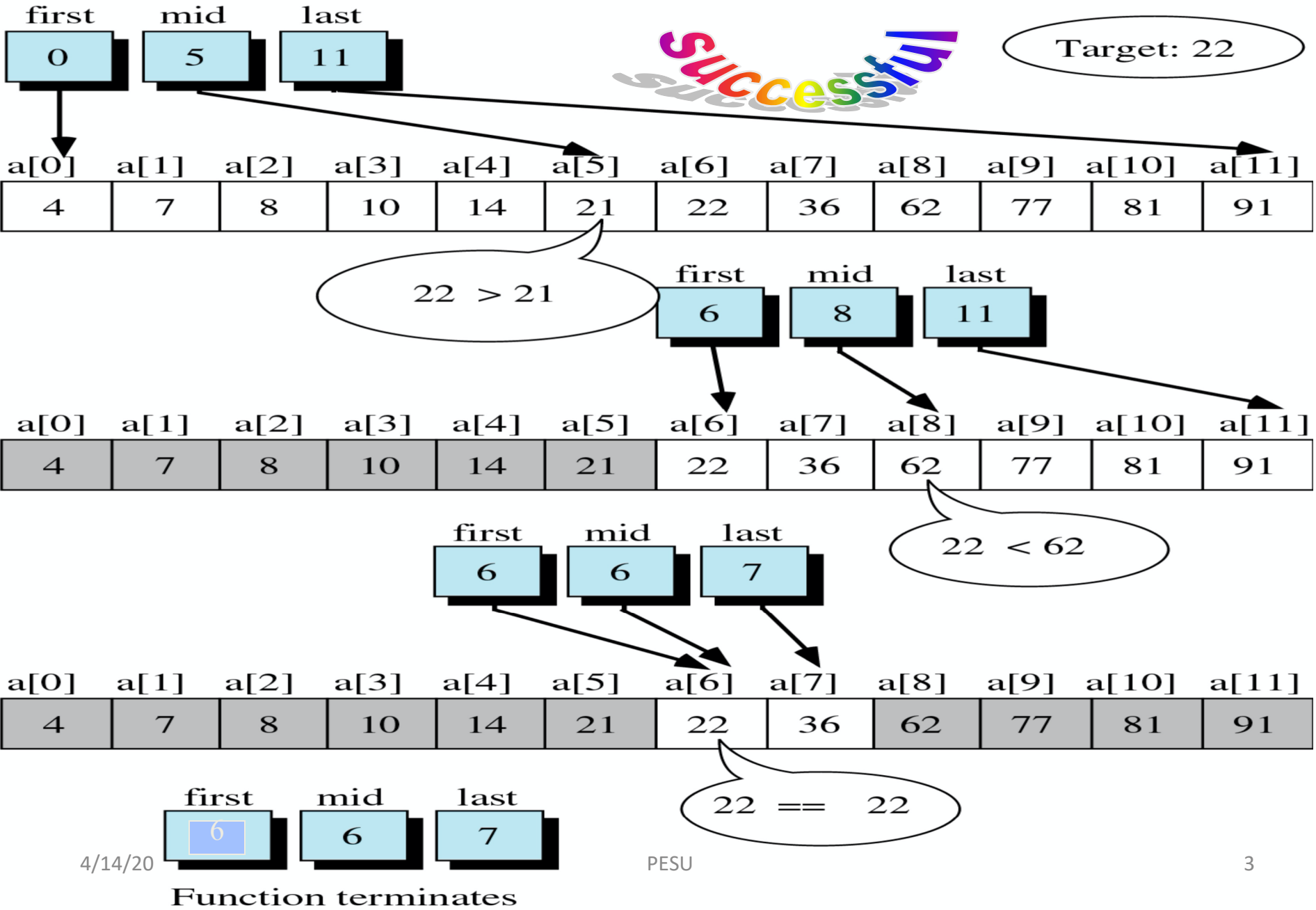
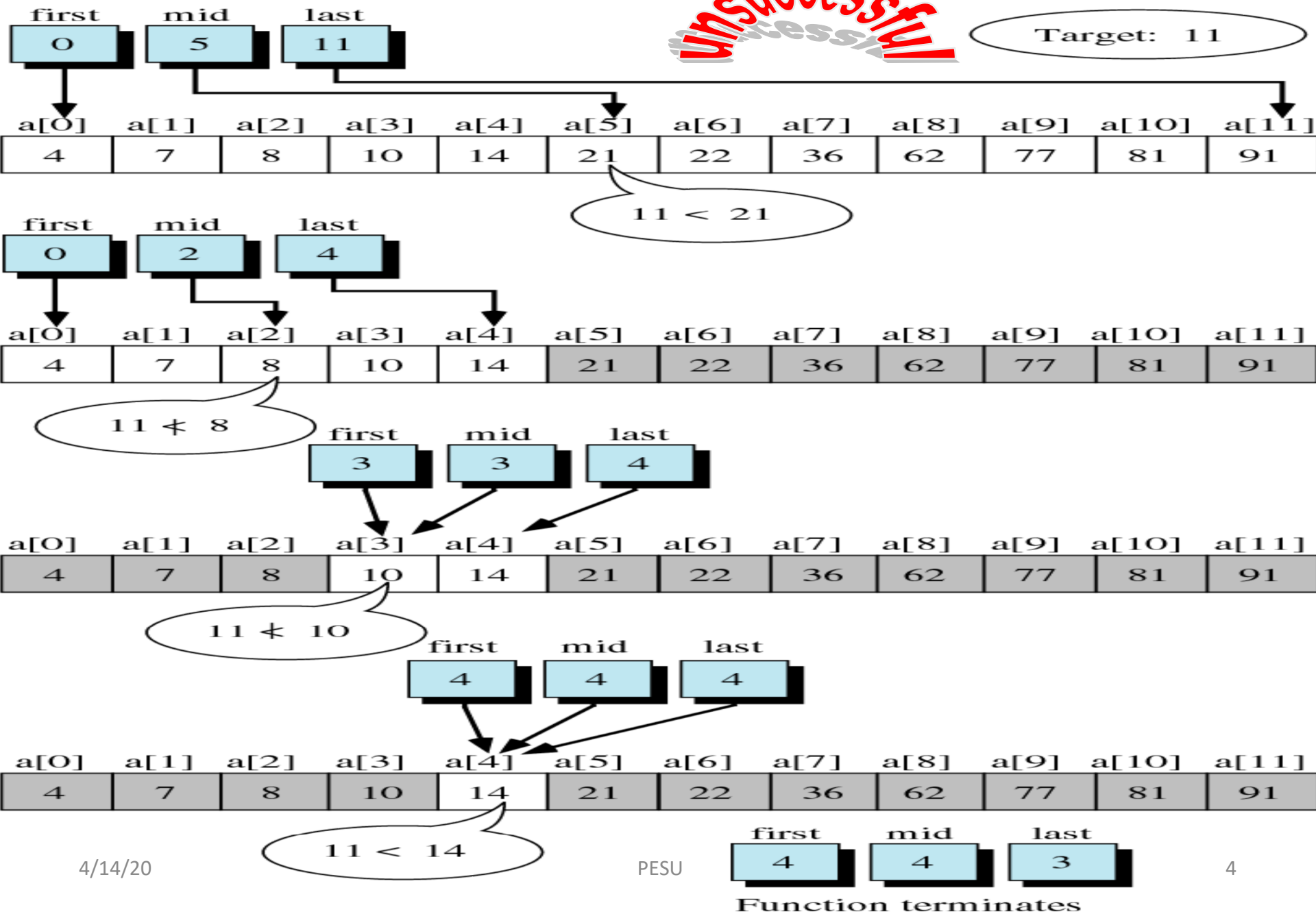
Unit :IV

Binary Search

- Search an ordered array of integers for a value and return its index if the value is found. Otherwise, return -1.

- Binary search skips over parts of the array if the search value cannot possibly be there.


first
mid
last
0
5
11
Successful
Target: 22
a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11]
4 7 8 10 14 21 22 36 62 77 81 91
22 > 21
first
mid
last
6
8
11
a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11]
4 7 8 10 14 21 22 36 62 77 81 91
22 < 62
first
mid
last
6
6
7
a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11]
4 7 8 10 14 21 22 36 62 77 81 91
22 == 22
first
mid
last
6
6
7
Function terminates


unsuccessful
Target: 11
first
mid
last
0
5
11
a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11]
4 7 8 10 14 21 22 36 62 77 81 91
11 < 21
first
mid
last
0
2
4
a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11]
4 7 8 10 14 21 22 36 62 77 81 91
11 ≠ 8
first
mid
last
3
3
4
a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11]
4 7 8 10 14 21 22 36 62 77 81 91
11 ≠ 10
first
mid
last
4
4
4
a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11]
4 7 8 10 14 21 22 36 62 77 81 91
11 < 14
first
mid
last
4
4
3
Function terminates

Binary Search

- Binary search is based on the “divide-and-conquer” strategy which works as follows:
  - Start by looking at the middle element of the array
    - 1. If the value it holds is lower than the search element, eliminate the first half of the array from further consideration.
    - 2. If the value it holds is higher than the search element, eliminate the second half of the array from further consideration.
  - Repeat this process until the element is found, or until the entire array has been eliminated.

Binary Search

- Algorithm:

```
Set first and last boundary of array to be searched
Repeat the following:
   Find middle element between first and last boundaries;
   if (middle element contains the search value)
          return middle_element position;
   else if (first >= last )
          return -1;
   else if (value < the value of middle_element)
          set last to middle_element position - 1;
   else
          set first to middle_element position + 1;
```

Binary Search

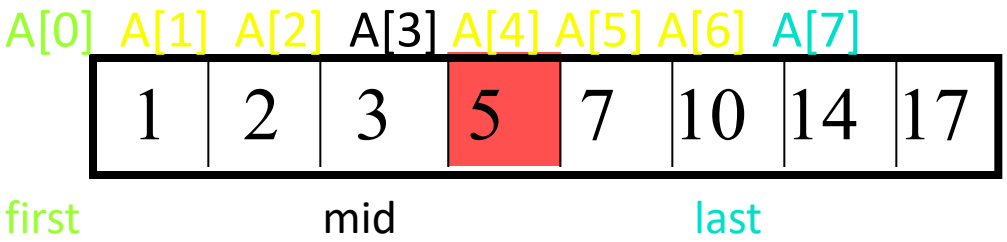
```
// Searches an ordered array of integers
int bsearch(int data[], // input: array
            int size,   // input: array size
            int value   // input: value to find
            )           // output: if found,return index
{                       //          otherwise, return -1
       int first, middle, last;
        first = 0;
        last = size - 1;
       while (true) {
          middle = (first + last) / 2;
          if (data[middle] == value)
               return middle;
          else if (first >= last)
               return -1;
          else if (value < data[middle])
              last = middle - 1;
          else
              first = middle + 1;
     }
}
```

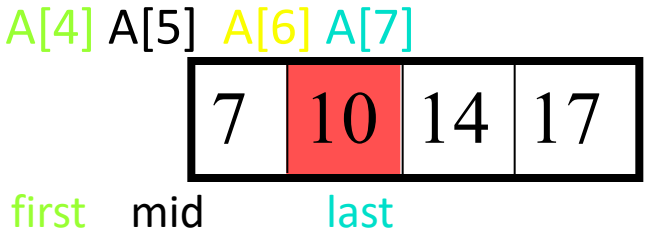
Binary Search

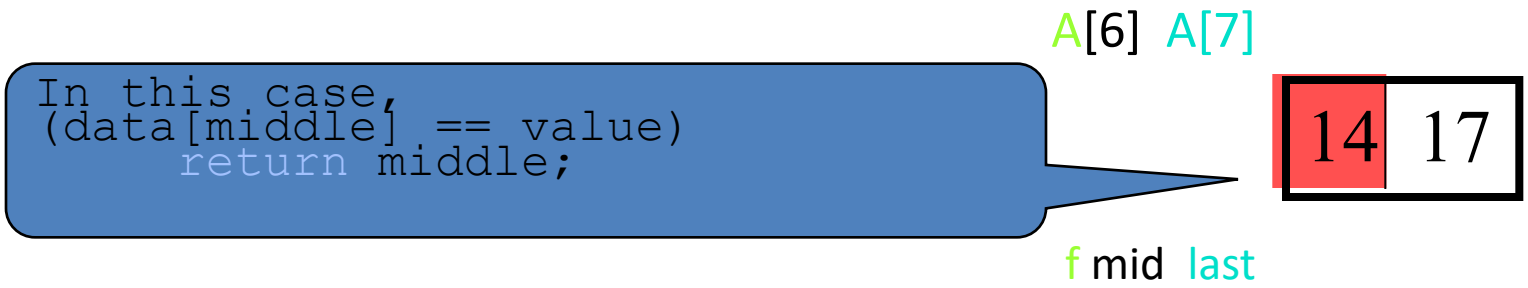
```
#include <stdio.h>
int main() {
   int array_size = 8;
   int list[array_size]={1,2,3,5,7,10,14,17};

int search_value;

   printf( "Enter search value: “);
   scanf(%d,&search_value);

   x=bsearch(list,array_size,search_value)

   printf(x);
   return 0;
}
```

Example: binary search

- 14 ?

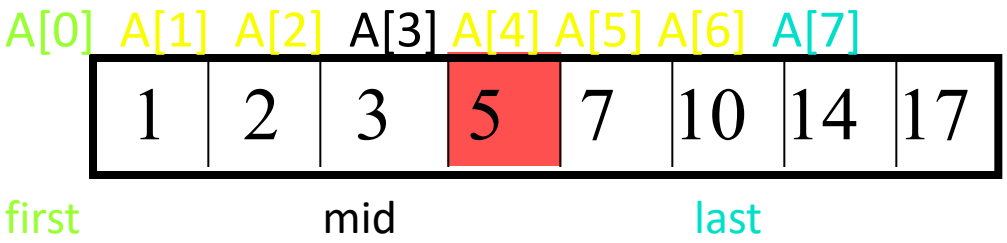


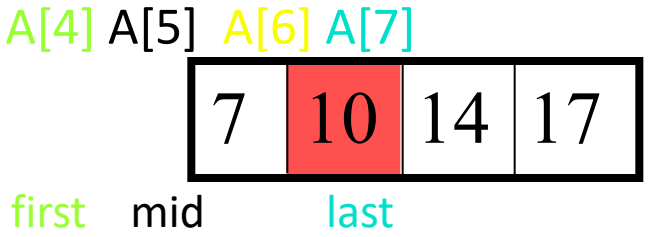




Example: binary search

- 8 ?






```
In this case, (first
   == last)
    return -1;
```

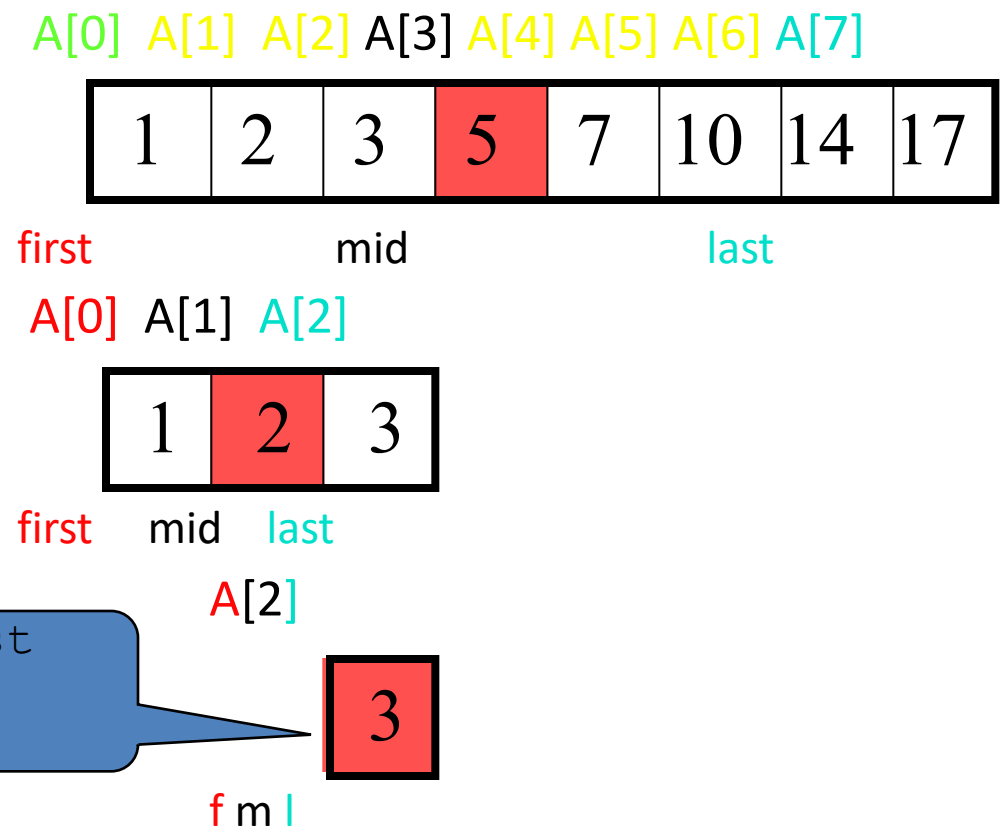
A[4]

7

f m l

Example: binary search

- 4 ?



C program to search a word using binary search technique

```
#include <stdio.h>          //standard input output functions
#include<string.h>                  //console functions
#define max 20                       //define max as 20

void search(char [][20],int,char[]);       //search function

void main()                 //main function
{
   char string[max][20],t[20],word[20];       //variables
   int i, j, n;
   printf("Enter the number of words: \n");
   scanf("%d", &n);                          //getting number of words
   printf("Enter the words: \n");
   for (i = 0; i < n; i++)                   //entering words
         scanf("%s",string[i]);
```

```
   /*  sorting elements as for binary search elements should be sorted */
   for (i = 1; i < n; i++)
   {
     for (j = 1; j < n; j++)
     {                                    //if the previous string is greater than next
        if (strcmp(string[j - 1], string[j]) > 0)
        {                                 //swap their positions
           strcpy(t, string[j - 1]);
            strcpy(string[j - 1], string[j]);
           strcpy(string[j], t);
        }
     }
   }


 printf("Input words \n");                 //displaying the words
    for (i = 0; i < n; i++)
        printf("%s\n", string[i]);

   printf("Enter the element to be searched: \n");
   scanf("%s",word);                          //entering the word to be searched
   search(string,n,word);                    //calling search function
}
```

```
/*  Binary searching begins */
void search(char string [][20],int n,char word[ ])
{
   int lb, mid, ub;
   lb = 0;                        //lower bound to 0
   ub = n;                         //upper bound to n
   do
   {
      mid = (lb + ub) / 2;           //finding the mid of the array

      if ((strcmp(word,string[mid]))<0)      //compare the word with mid
         ub = mid - 1;                       //if small then decrement ub

      else if ((strcmp(word,string[mid]))>0)
         lb = mid + 1;                       //if greater then increment lb

    /*repeat the process till lb doesn't becomes ub and string is found */
   } while ((strcmp(word,string[mid])!=0) && lb <= ub);

   if ((strcmp(word,string[mid]))==0)            //if string is found
        printf("SEARCH SUCCESSFUL \n");
   else                                 //if not found
        printf("SEARCH FAILED \n");
}
```