# Unit 2

Program for Binary search

# Aim:

- To implement Binary Search Algorithm

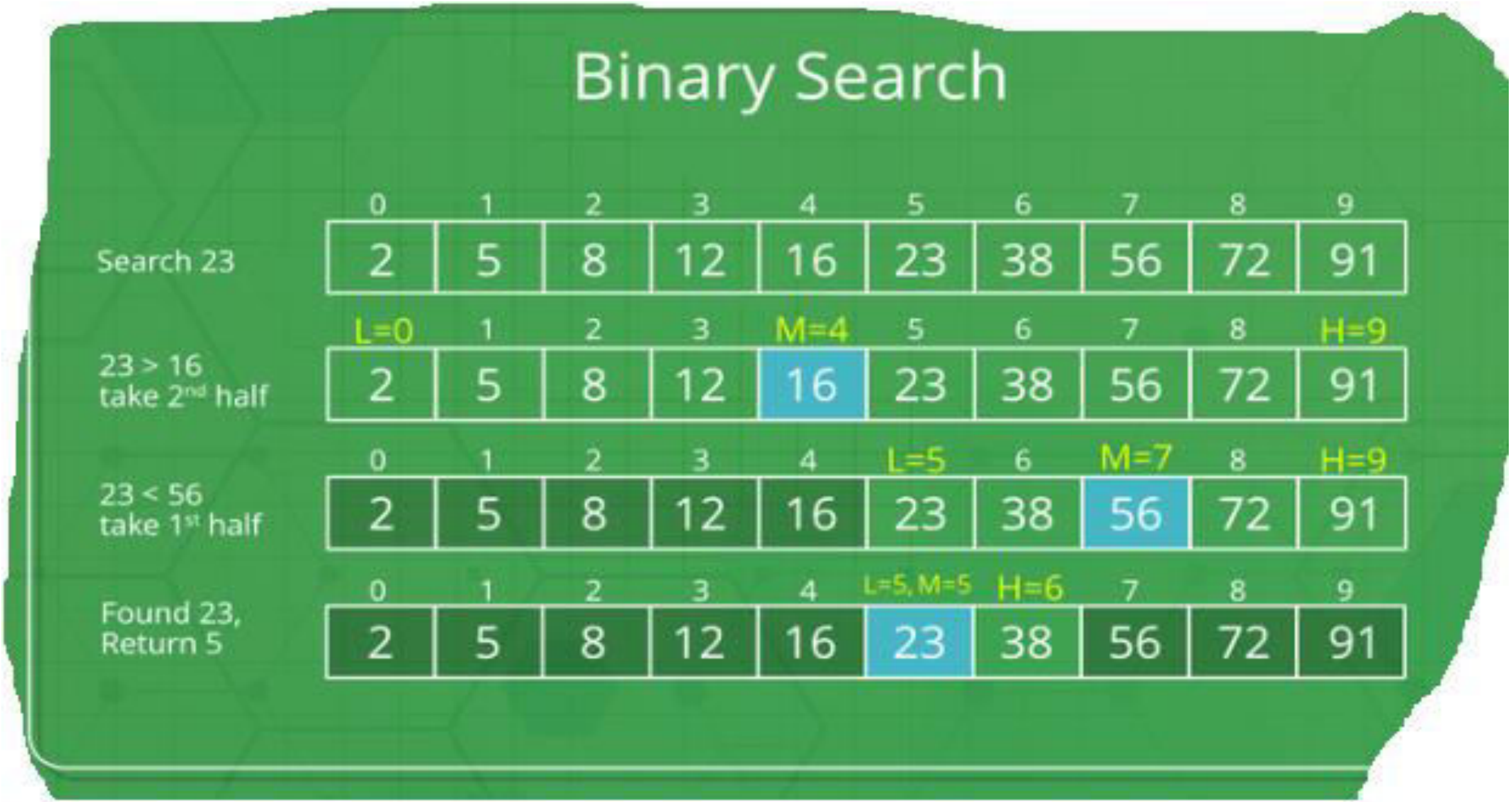
# Theory:

- Binary search is the search technique that works efficiently on sorted lists. Hence, to search an element into some list using the binary search technique, we must ensure that the list is sorted.

- Binary search follows the divide and conquer approach in which the list is divided into two halves, and the item is compared with the middle element of the list.
- If the match is found then, the location of the middle element is returned.
- Otherwise, we search into either of the halves depending upon the result produced through the match.

# Binary Search Logic

- Step1:Begin with the mid element of the whole array as a search key.
- Step 2: If the value of the search key is equal to the item then return an index of the search key.
- Step3:Or if the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
- Step4:Otherwise, narrow it to the upper half.
- Step5:Repeatedly check from the second point until the value is found or the interval is empty.




*Example of Binary Search Algorithm*

- Binary search is a searching algorithm that works efficiently with a sorted list.
- The mechanism  of binary search can be better understood by an analogy of a telephone directory.
- When we are  searching for a particular name in a directory, we first open the directory from the middle and then decide whether to look for the name in the first part of the directory or in the second part of the directory.
- Again, we open some page in the middle and the whole process is repeated until we finally find the right name.

- Take another analogy. How do we find words in a dictionary? We first open the dictionary somewhere in the middle.
- Then, we compare the first word on that page with the desired word whose meaning we are looking for. If the desired word comes before the word on the page, we look in the first half of the dictionary, else we look in the second half.
- Again, we open a page in the first half of the dictionary and compare the first word on that page with the desired word and repeat the same procedure until we finally get the word.
- The same mechanism is applied in the binary search

Now, let us consider how this mechanism is applied to search for a value in a sorted array.

Consider an array A[] that is declared and initialized as

int A[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

and the value to be searched is VAL = 9.

The algorithm will proceed in the following manner.

BEG = 0, END = 10, MID = (0 + 10)/2 = 5

Now, VAL = 9 and A[MID] = A[5] = 5

A[5] is less than VAL, therefore, we now search for the value in the second half of the array. So, we change the values of BEG and MID.

Now, BEG = MID + 1 = 6, END = 10, MID = (6 + 10)/2 =16/2 = 8

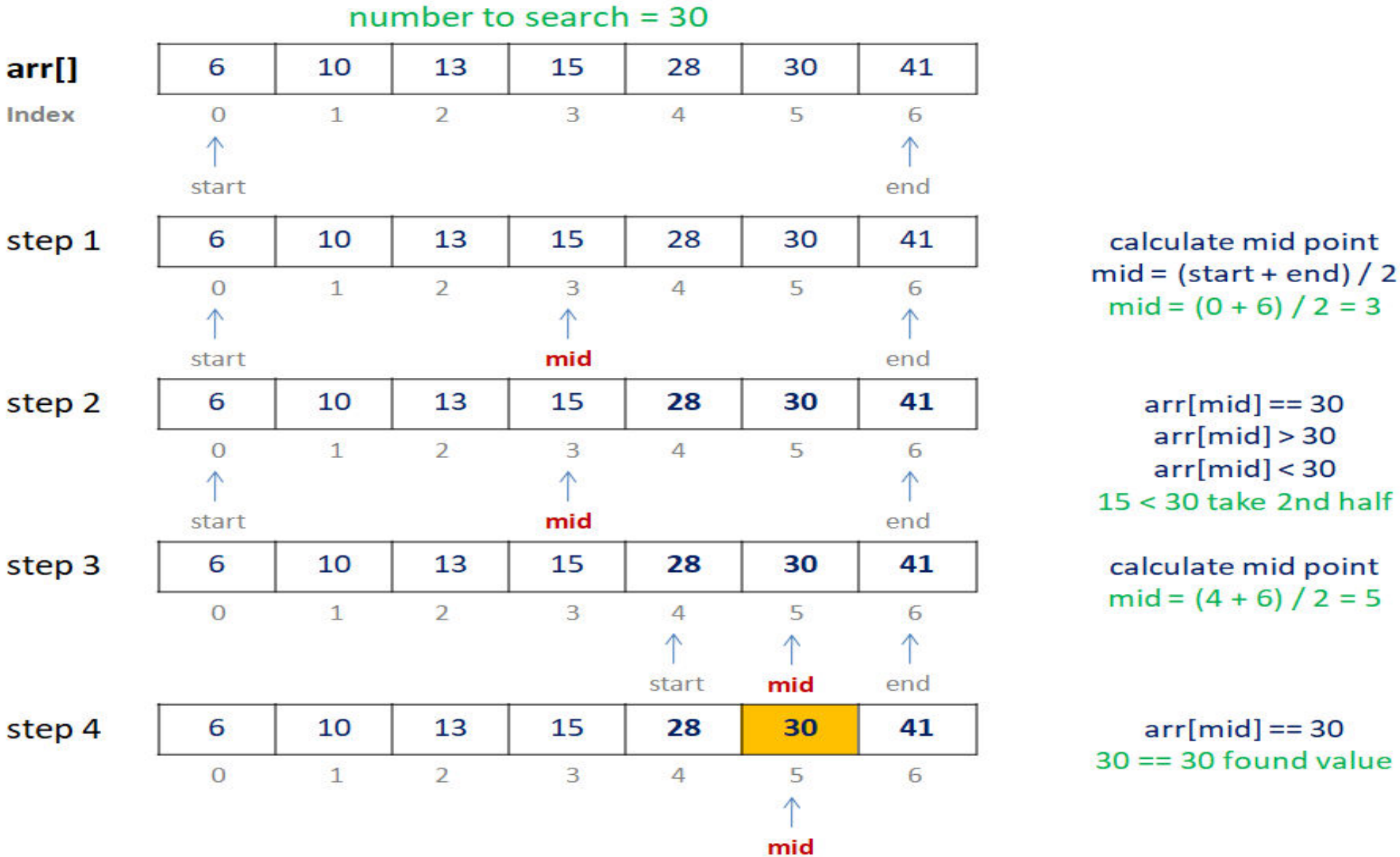
VAL = 9 and A[MID] = A[8] = 8

A[8] is less than VAL, therefore, we now search for the value in the second half of the segment. So, again we change the values of BEG and MID.

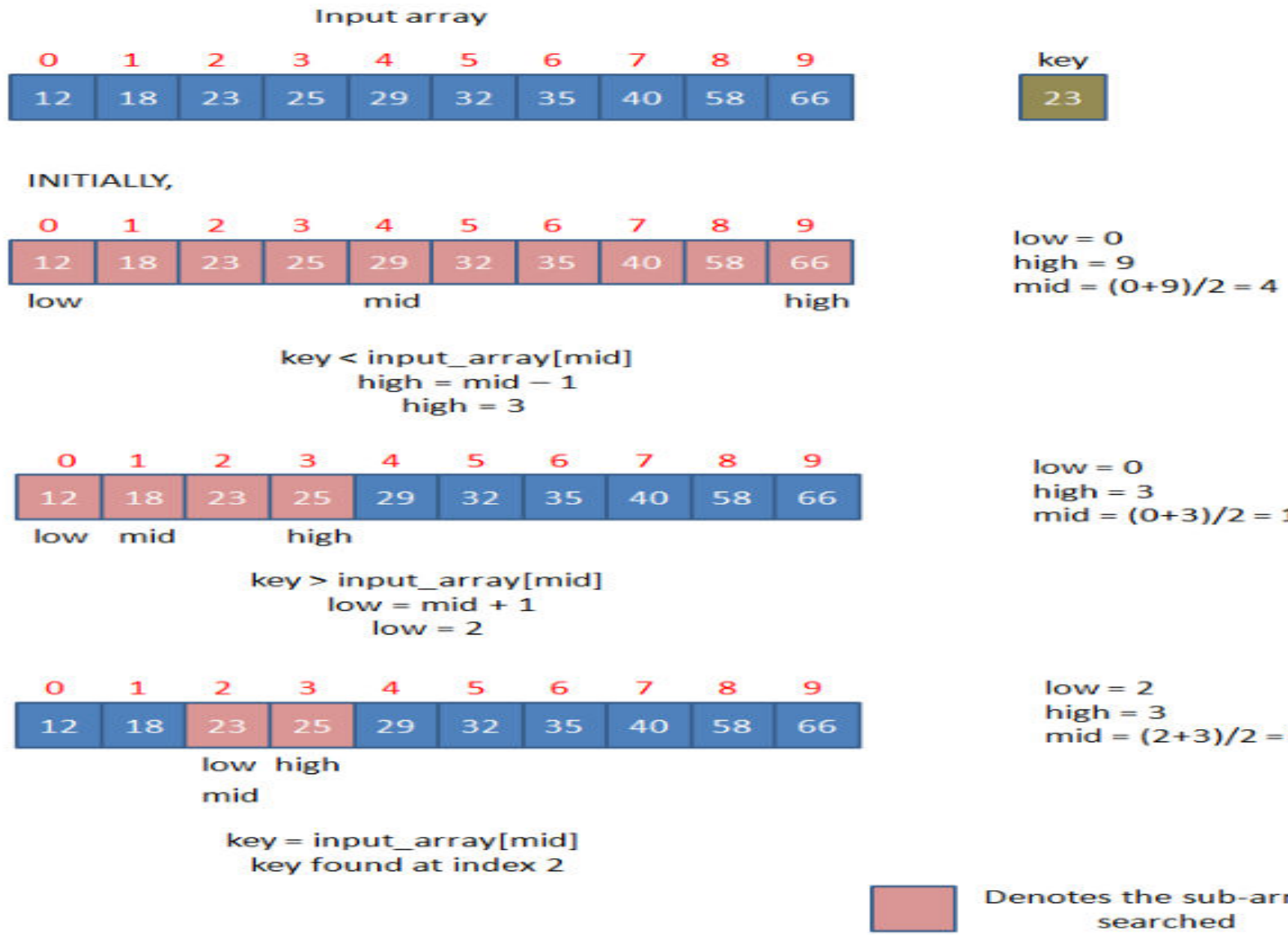
Now, BEG = MID + 1 = 9, END = 10, MID = (9 + 10)/2 = 9

Now, VAL = 9 and A[MID] = 9.

- In this algorithm, we see that BEG and END are the beginning and ending positions of the segment that we are looking to search for the element.
- MID is calculated as (BEG + END)/2.
- Initially, BEG = lower_bound and END = upper_bound.
- The algorithm will terminate when A[MID] = VAL.
- When the algorithm ends, we will set POS = MID.
- POS is the position at which the value is present in the array.
- However, if VAL is not equal to A[MID], then the values of BEG, END, and MID will be changed depending on whether VAL is smaller or greater than A[MID].
- (a) If VAL < A[MID], then VAL will be present in the left segment of the array. So, the value of END will be changed as END = MID – 1.
- (b) If VAL > A[MID], then VAL will be present in the right segment of the array. So, the value of BEG will be changed as BEG = MID + 1.
- Finally, if VAL is not present in the array, then eventually, END will be less than BEG. When this happens, the algorithm will terminate and the search will be unsuccessful.


number to search = 30
arr[]
6 10 13 15 28 30 41
Index
0 1 2 3 4 5 6
start
end
step 1
6 10 13 15 28 30 41
0 1 2 3 4 5 6
start
mid
end
calculate mid point
mid = (start + end) / 2
mid = (0 + 6) / 2 = 3
step 2
6 10 13 15 28 30 41
0 1 2 3 4 5 6
start
mid
end
arr[mid] == 30
arr[mid] > 30
arr[mid] < 30
15 < 30 take 2nd half
step 3
6 10 13 15 28 30 41
0 1 2 3 4 5 6
start
mid
end
calculate mid point
mid = (4 + 6) / 2 = 5
step 4
6 10 13 15 28 30 41
0 1 2 3 4 5 6
mid
arr[mid] == 30
30 == 30 found value

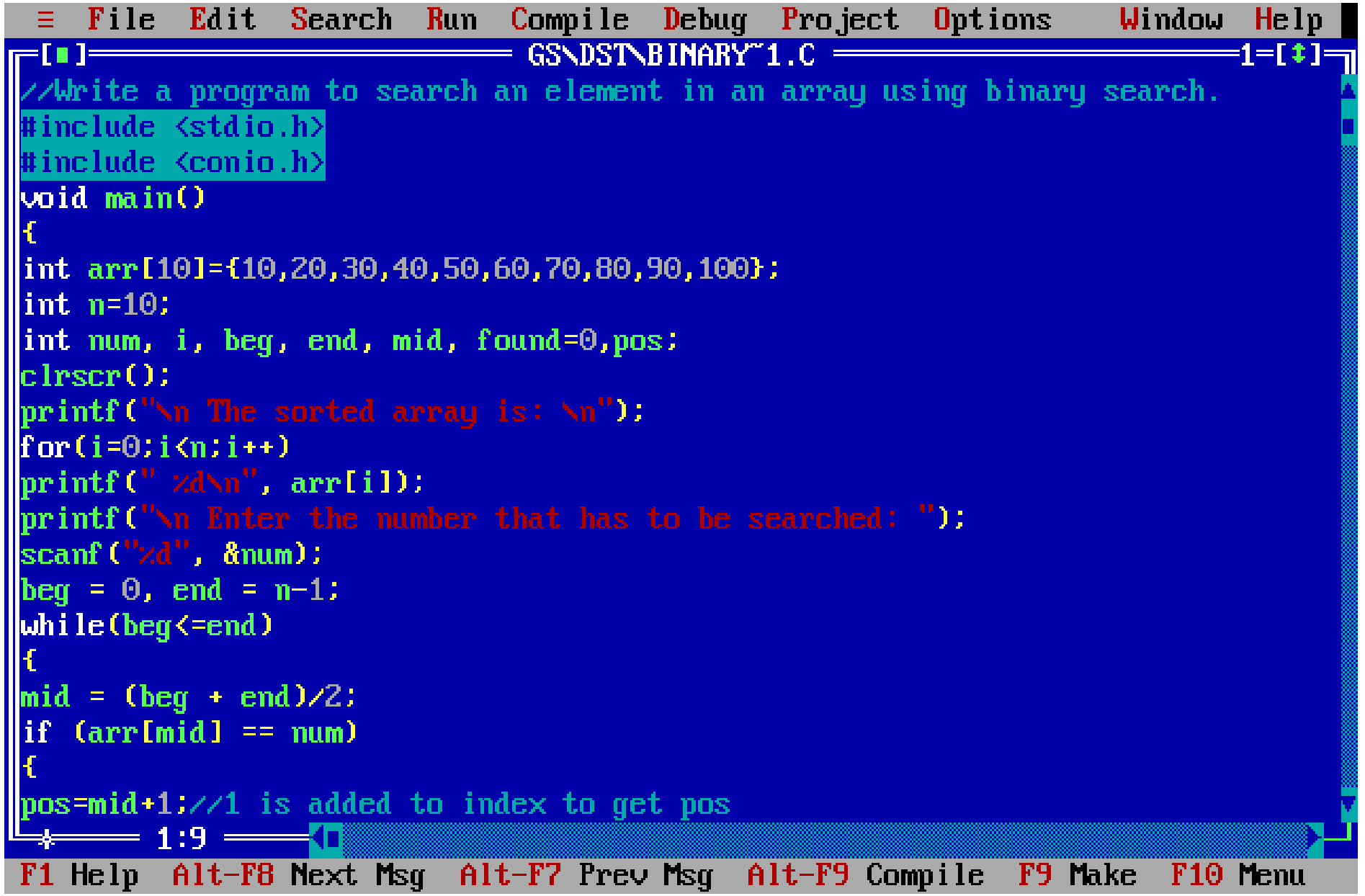
Let `input_array` - {12, 18, 23, 25, 29, 32, 35, 40, 58, 66} and `key = 18`



```
BINARY_SEARCH(A, lower_bound, upper_bound, VAL)
Step 1: [INITIALIZE] SET BEG = lower_bound
        END = upper_bound, POS = - 1
Step 2: Repeat Steps 3 and 4 while BEG <= END
Step 3:            SET MID = (BEG + END)/2
Step 4:            IF A[MID] = VAL
                         SET POS = MID
                         PRINT POS
                         Go to Step 6
                   ELSE IF A[MID] > VAL
                         SET END = MID - 1
                   ELSE
                         SET BEG = MID + 1
                   [END OF IF]
        [END OF LOOP]
Step 5: IF POS = -1
            PRINT "VALUE IS NOT PRESENT IN THE ARRAY"
        [END OF IF]
Step 6: EXIT
```

**Figure 14.2** Algorithm for binary search

# Source Code


File Edit Search Run Compile Debug Project Options Window Help
GS\DST\BINARY~1.C
//Write a program to search an element in an array using binary search.
#include <stdio.h>
#include <conio.h>
void main()
{
int arr[10]={10,20,30,40,50,60,70,80,90,100};
int n=10;
int num, i, beg, end, mid, found=0,pos;
clrscr();
printf("\n The sorted array is: \n");
for(i=0;i<n;i++)
printf(" %d\n", arr[i]);
printf("\n Enter the number that has to be searched: ");
scanf("%d", &num);
beg = 0, end = n-1;
while(beg<=end)
{
mid = (beg + end)/2;
if (arr[mid] == num)
{
pos=mid+1;//1 is added to index to get pos
1:9
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
if (arr[mid] == num)
{
pos=mid+1;//1 is added to index to get pos
printf("\n %d is present in the array at position %d", num, pos);
found =1;
break;
}
else if (arr[mid]>num)
end = mid-1;
 else
beg = mid+1;
        }
if (beg > end && found == 0)
printf("\n %d does not exist in the array", num);
getch();
}
```

# output

```
The sorted array is:
10
20
30
40
50
60
70
80
90
100

Enter the number that has to be searched: 100

100 is present in the array at position 10
```

```
The sorted array is:
10
20
30
40
50
60
70
80
90
100

Enter the number that has to be searched: 300

300 does not exist in the array_
```