INTODUCTION

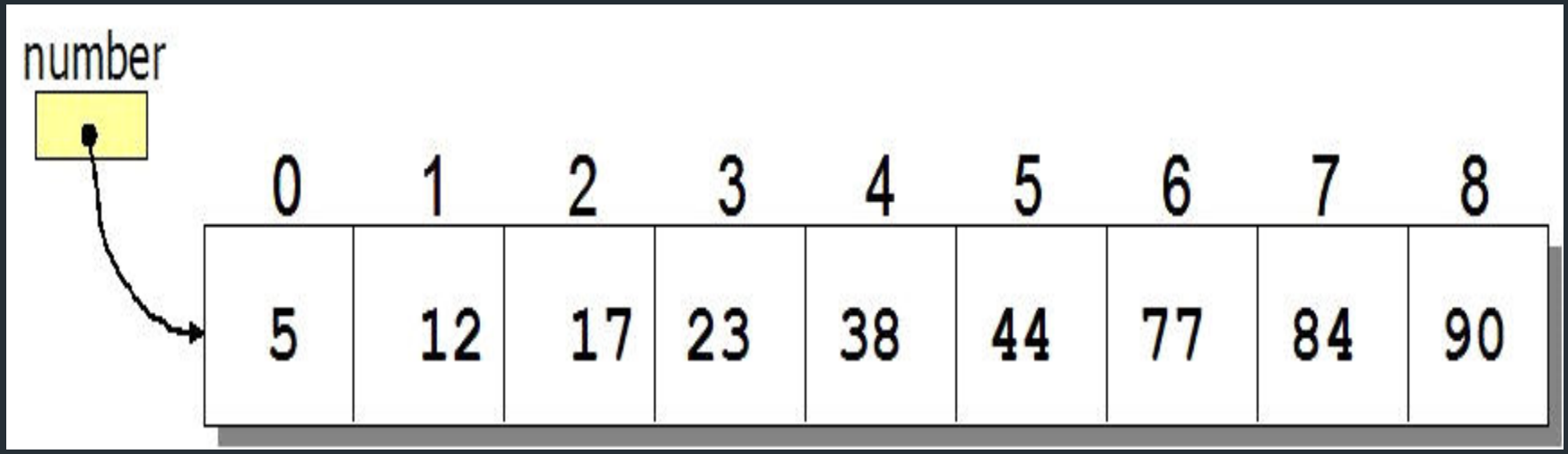
A **Binary search** algorithm finds the position of a specified input value (the search "key") within a sorted array . For binary search, the array should be arranged in ascending or descending order.

Why Binary Search Is Better Than Linear Search ?

A linear search works by looking at each element in a list of data until it either finds the target or reaches the end. This results in O(n) performance on a given list. A binary search comes with the prerequisite that the data must be sorted. We can use this information to decrease the number of items we need to look at to find our target. That is ,

Binary Search is fast as compared to linear search because of less number of computational operations .

Example :

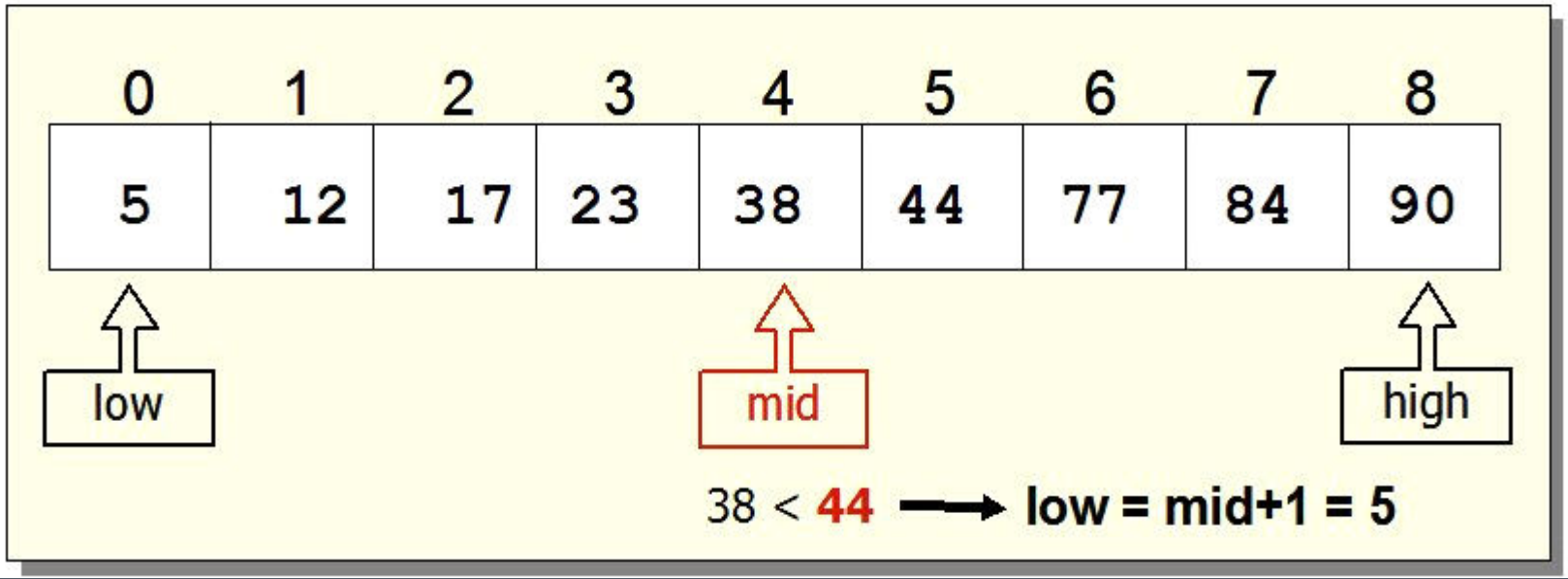


Step 1 :

search(44)

	low	high	mid
#1	0	8	4

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$

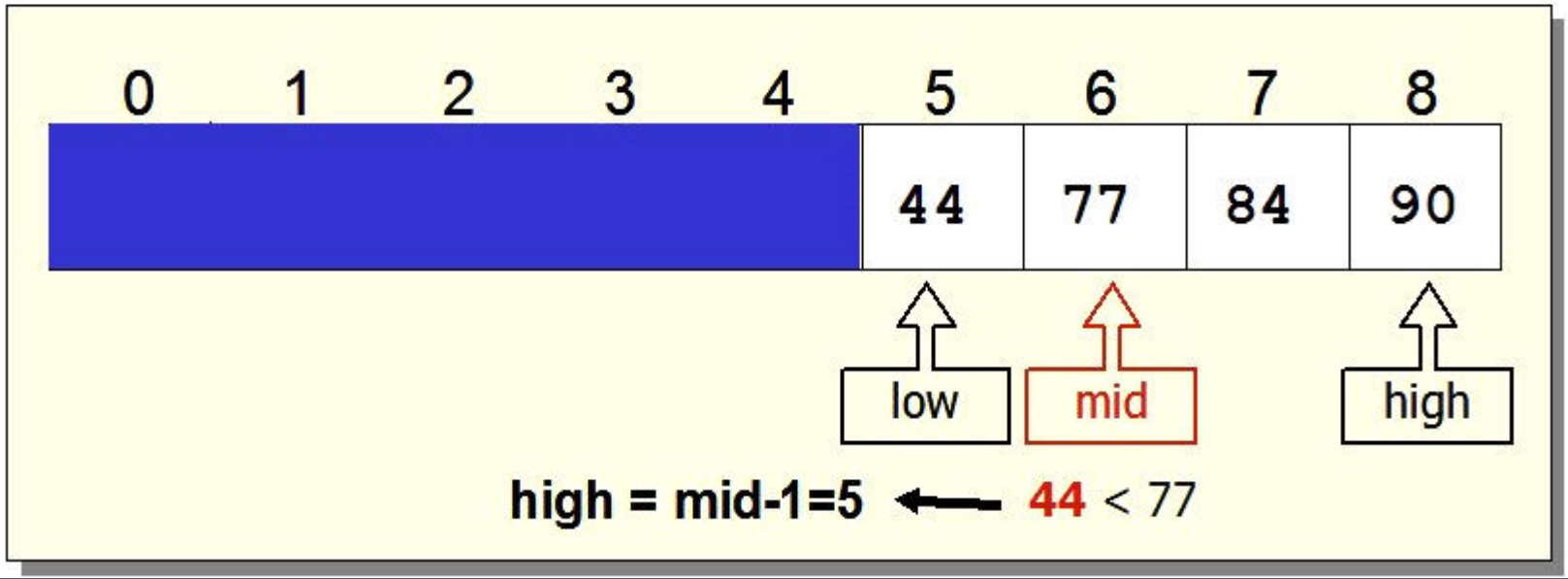


Step 2:

search(44)

	low	high	mid
#1	0	8	4
#2	5	8	6

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$

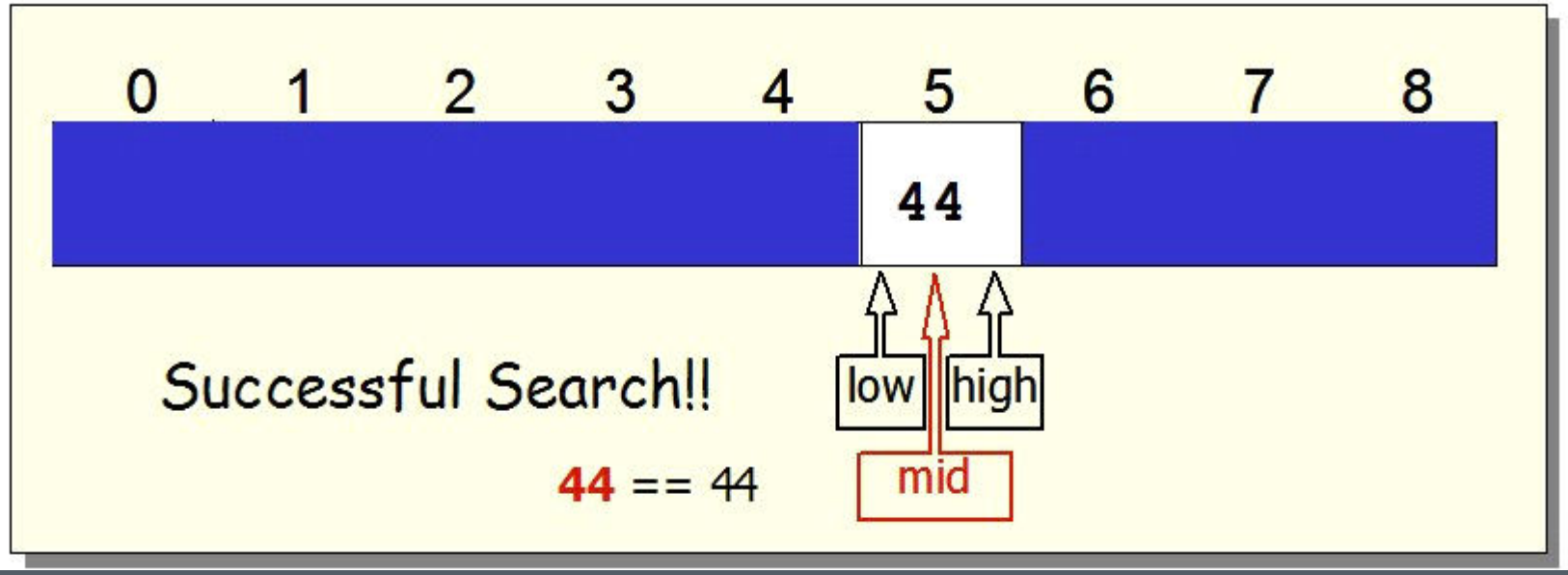


Step 3:

	low	high	mid
#1	0	8	4
#2	5	8	6
#3	5	5	5

search(44)

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$



ALGORITHM (Recursive)

```
recursivebinarysearch(int A[], int first, int last, int key)
 if last<first
    index=-1
else
int mid = (first + last) / 2
if key=A[mid]
     index=mid
else if key<A[mid]
     index= recursivebinarysearch(A, first, mid – 1, key)
 else
     index= recursivebinarysearch(A, mid + 1, last, key)
return index
```

BINARY SEARCH ALGORITHM

Algorithm is quite simple. It can be done either recursively or iteratively:

- Get the middle element;
- If the middle element equals to the searched value, the algorithm stops;
- Otherwise, two cases are possible:
- Searched value is less, than the middle element. In this case, go to the step 1 for the part of the array, before middle element.
- Searched value is greater, than the middle element. In this case, go to the step 1 for the part of the array, after middle element.
- Now, The iterations should stop when searched element is found & when sub array has no elements. In this case, we can conclude, that searched value is not present in the array.

ANALYSIS OF BINARY SEARCH

To analyze the binary search algorithm, we need to recall that each comparison eliminates about half of the remaining items from consideration. If we start with *n* items, about n/2 items will be left after the first comparison. After the second comparison, there will be about n/4. Then n/8, n/16, and so on.

When we split the list enough times, we end up with a list that has just one item. Either that is the item we are looking for or it is not. Either way, we are done. The number of comparisons necessary to get to this point is i where n/2^i=1. Solving for i gives us i=(log n). The maximum number of comparisons is log n. Therefore, the complexity of binary search is O(log n).

CORRECTNESS OF BINARY SEARCH

- **Lemma:** For all n>=0, if BinarySearch(E, first , last, k) is called, and the size is (last-first+1)=n and E[first], …E[last] are in ascending order, then it returns -1 if k doesn't occur in E within the range first, … last and it returns index such that k=E[index] otherwise

PROOF BY INDUCTION

- **Base case:**

  $$n = last - first + 1 = 0$$

- The array is empty, so first = last + 1
- Line 1 i.e. (first > last) is true , line 2 is reached and the algorithm correctly returns -1.
- Therefore, the element is not present in the array.
- **Inductive step:** For n>0, assume that the binary search(E, first, last ,k) satisfies the lemma on size k such that 0<=k<n, and first and the last are any indexes such that k=last-first+1