Binary Search

A **binary search** or **half-interval search** algorithm finds the position of a specified value (the input " **searchkey**") within a **sorted** array.

At each stage, the algorithm compares the Searched value (input **searchkey**) with the key value of the **middle** element of the array. If the keys match, then a matching element has been found so its index, or position, is returned.

Otherwise, if the Searched value (**searchkey**) is less than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the input searchkey is greater, on the sub-array to the right.

If the remaining array to be searched is reduced to zero, then the key cannot be found in the array and a special "**Not Found**" indication is returned.

A binary search halves the number of items to check with each iteration.

Binary Search Algorithm

1. Get Number of Array elements (**Size**)
2. Get Array elements (**List**)
3. Get search key (**searchkey** – Target -)
4. calculate the middle element.
5. If the middle element equals to the searched value, the algorithm stops; otherwise, two cases are possible:

   a) Searched value is less, than the middle element. In this case, go to the step 4 for the part of the array, before middle element.

b) Searched value is greater, than the middle element. In this case, go to the step 4 for the part of the array, after middle element.

```
first = 0
last = Size -1
while (first < =  last)
{
    mid = (first + last) div 2
    if(searchkey = list [mid] ) then
        return (mid)
    else
        if  (searchkey <  list [mid] )  then
           last = mid - 1
        else
            if(searchkey >  list [mid] ) then
                first  =  mid +1
}
return (-1)
```

Binary Search Function:

```
int BinarySearch (int Size , int List[ ], int searchkey )
{
    int first , last ,mid ;
    first = 0;
    last = Size -1;
    while (first < =  last)
    {
        mid = (first + last) / 2;
        if(searchkey == list [mid] )
            return mid ;
        else
            if  (searchkey <  list [mid] )
               last = mid – 1;
            else
                if(searchkey >  list [mid] )
                    first  =  mid +1;
   }
        return -1;    // failed to find search key
}
```

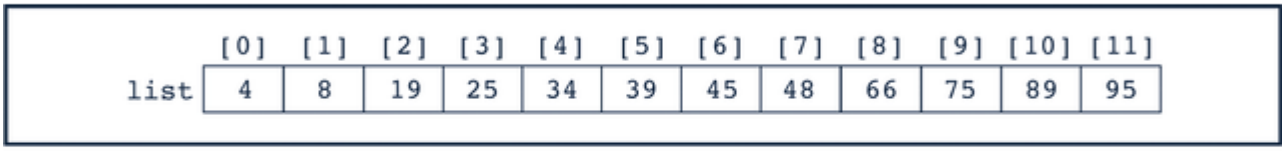
- **Benefit**

  Much more efficient than linear search

- **Disadvantage**

  Requires that list elements be sorted

Example 1:



Sorted list for a binary search

Values of `first`, `last`, and `middle` and the Number of Comparisons for Search Item 89

Iteration	`first`	`last`	`mid`	`list[mid]`	Number of Comparisons
1	0	11	5	39	2
2	6	11	8	66	2
3	9	11	10	89	1 (`found` is `true`)

Example 2:

Values of `first`, `last`, and `middle` and the Number of Comparisons for Search Item 22

Iteration	`first`	`last`	`mid`	`list[mid]`	Number of Comparisons
1	0	11	5	39	2
2	0	4	2	19	2
3	3	4	3	25	2
4	3	2	the loop stops (because `first > last`)		