# Searching And Sorting Techniques

Prof. Harish D.G.
Dept. of Computer and IT
College of Engineering,Pune
www.harishgadade.com

# Searching

- Need of Searching and Sorting
- Basic Concepts
  - Sort Orders
  - Sort Stability
  - Sorting Efficiency
  - Passes
- Searching Techniques
- Sorting Techniques

# Sorting Stability

| Name | Subject | Marks |
|---|---|---|
| Vihaan | FDSP | 78 |
| Amit | DC | 65 |
| Vihaan | DC | 56 |
| Ritesh | DC | 45 |
| Rohit | FDSP | 87 |

a. Unsorted List

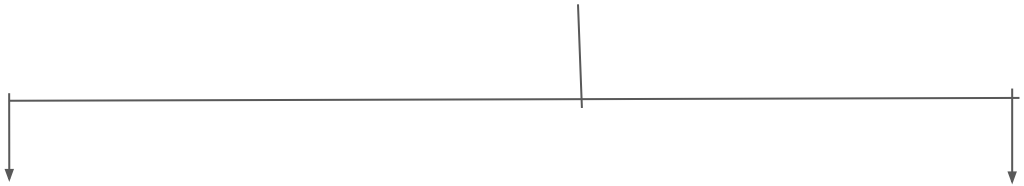
| Name | Subject | Marks |
|---|---|---|
| Amit | DC | 65 |
| Ritesh | DC | 45 |
| Rohit | FDSP | 87 |
| Vihaan | FDSP | 78 |
| Vihaan | DC | 56 |

b. Sorted List (Stable)
e.g. Bubble, seleion, insertion, merge sort

| Name | Subject | Marks |
|---|---|---|
| Amit | DC | 65 |
| Ritesh | DC | 45 |
| Rohit | FDSP | 87 |
| Vihaan | DC | 56 |
| Vihaan | FDSP | 78 |

c. Sorted List (UnStable)
e.g. Quik, shell, radix sort

# Searching Teachniques



## Linear Search

- Sequential Search, start from first element
- It Compares searching element with each item in the list.
- Stop either item found or end of the array is encountered
- Search may be successful or Unsuccessful

## Binary Search

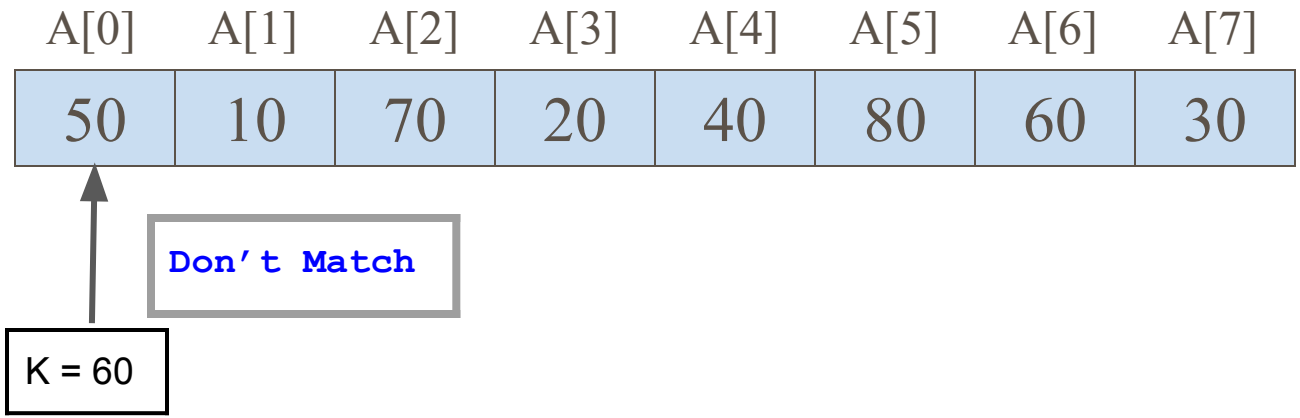
- Data Must be in Sorted Order
- Find the mid item from the list, say mid
- Compare searching key, say K, with mid.
- If key, k<mid, then searching key K may present to the left side of array;
- Otherwise, searching key, K may present right side of an array
- Continue above two steps till search is successful or unsuccessful.

# Linear Search

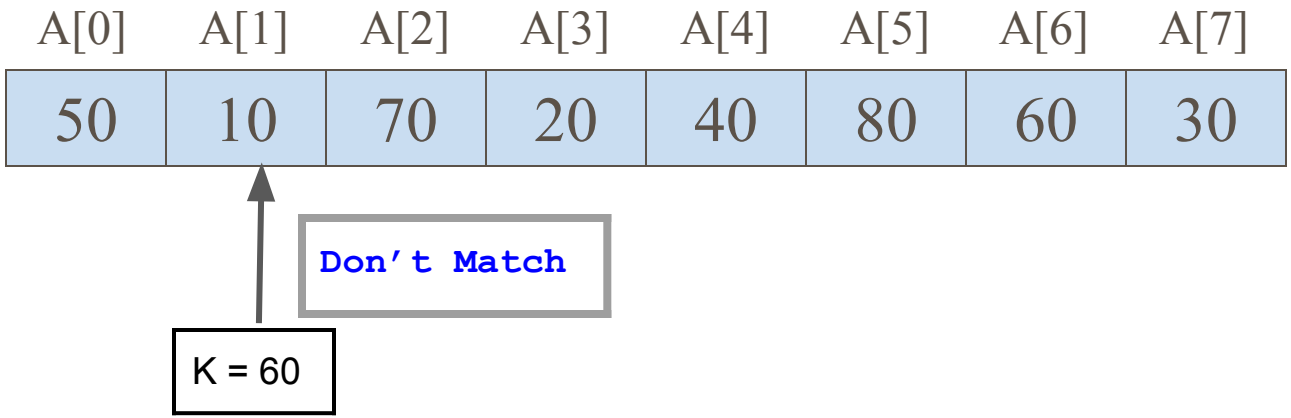
| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] | A[7] |
|---|---|---|---|---|---|---|---|
| 50 | 10 | 70 | 20 | 40 | 80 | 60 | 30 |

**Searching Key ( K ) = 60 ?**

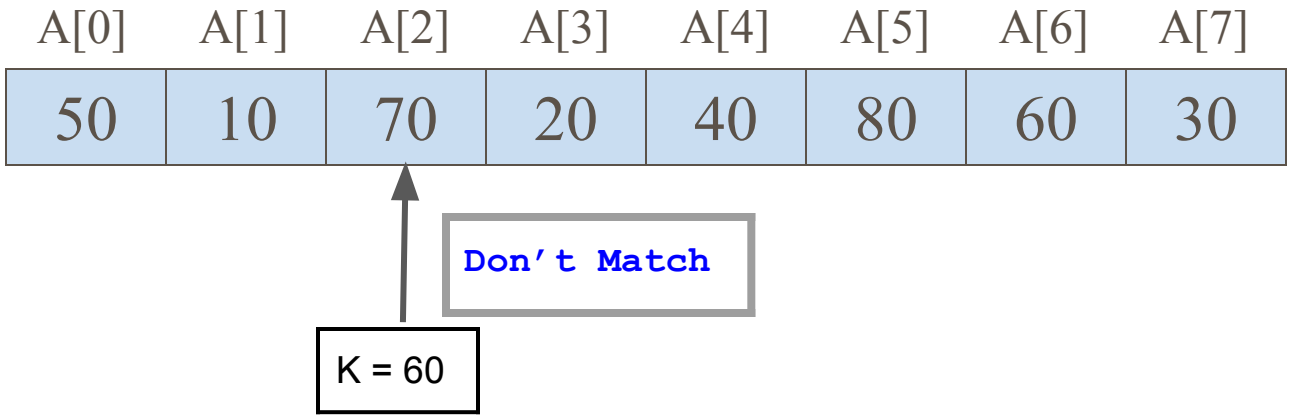
# Linear Search



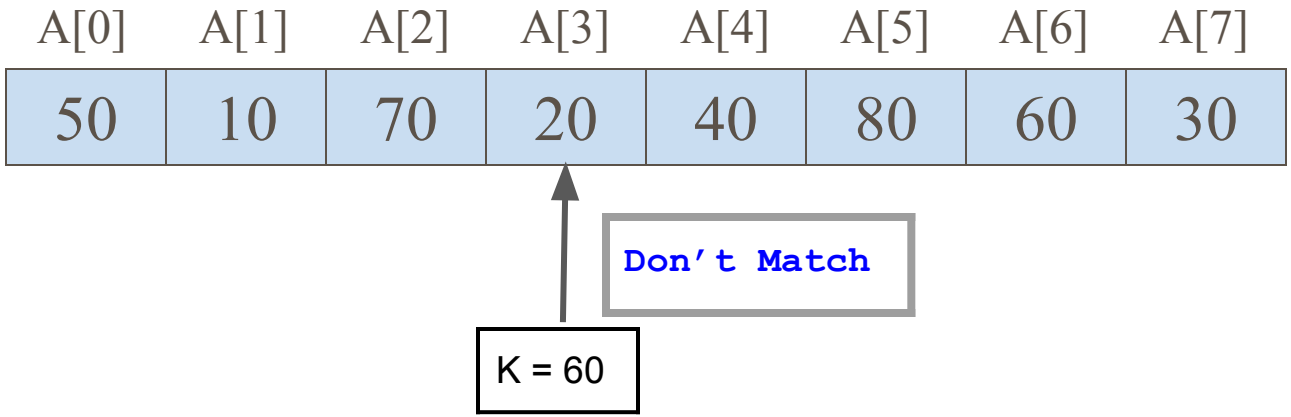
# Linear Search



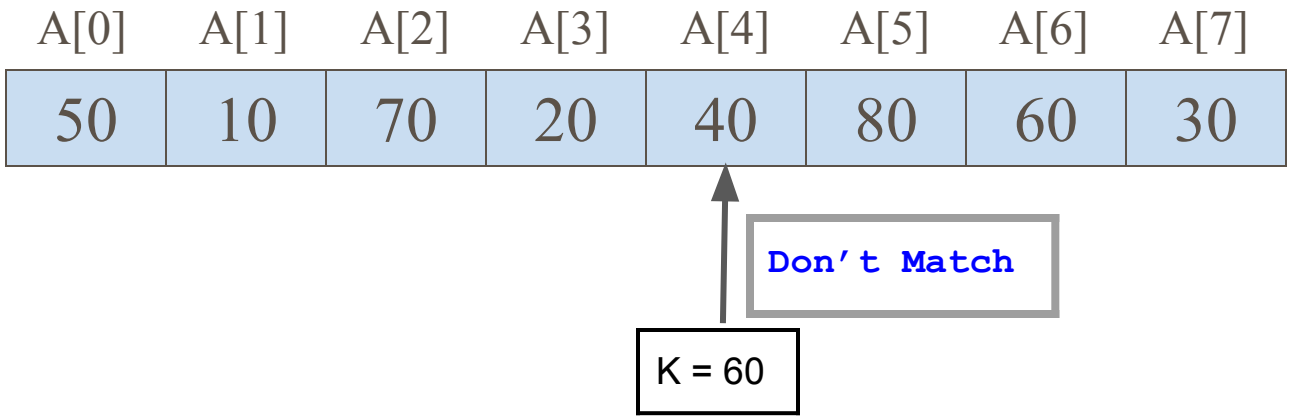
# Linear Search



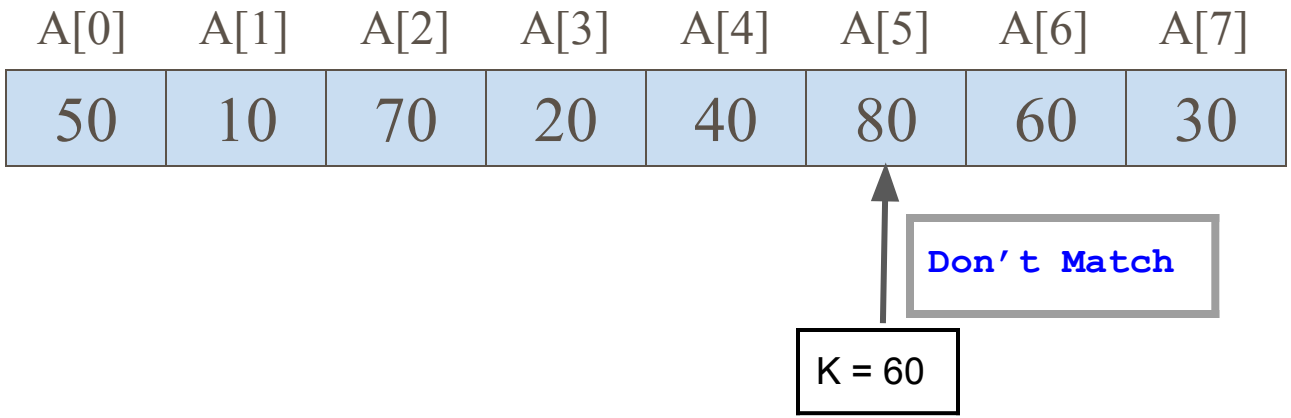
# Linear Search



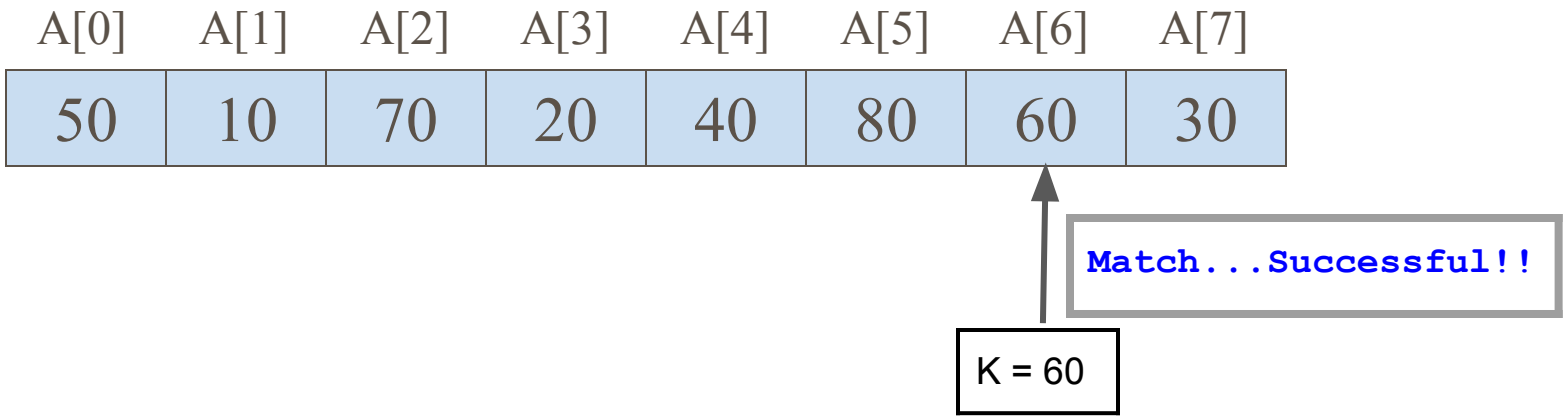
# Linear Search



# Linear Search



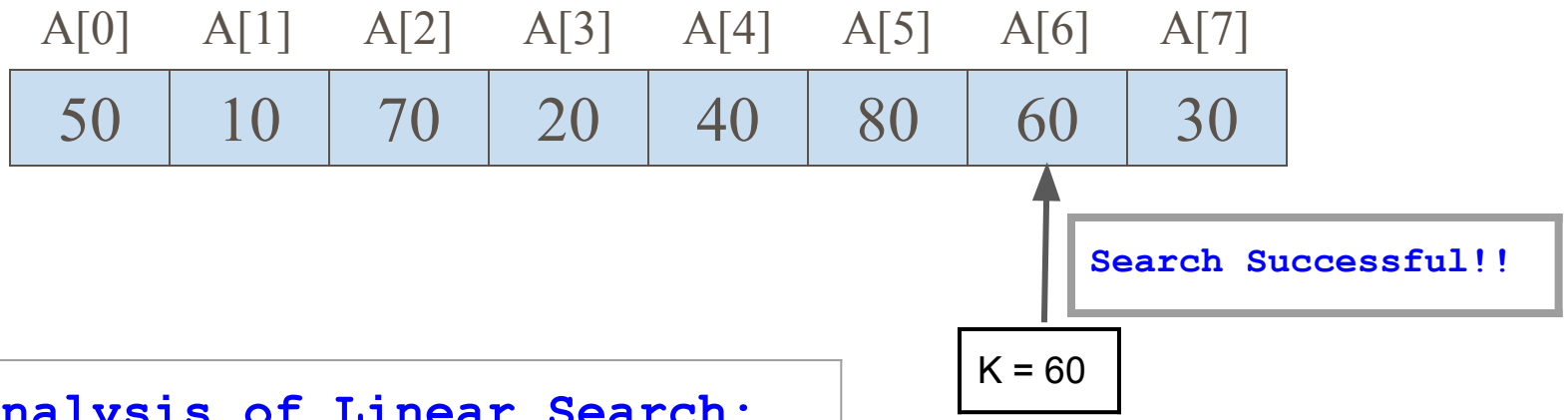
# Linear Search




**Searching Key ( K ) = 60 : Found**

**Time Complexity = O ( n )**

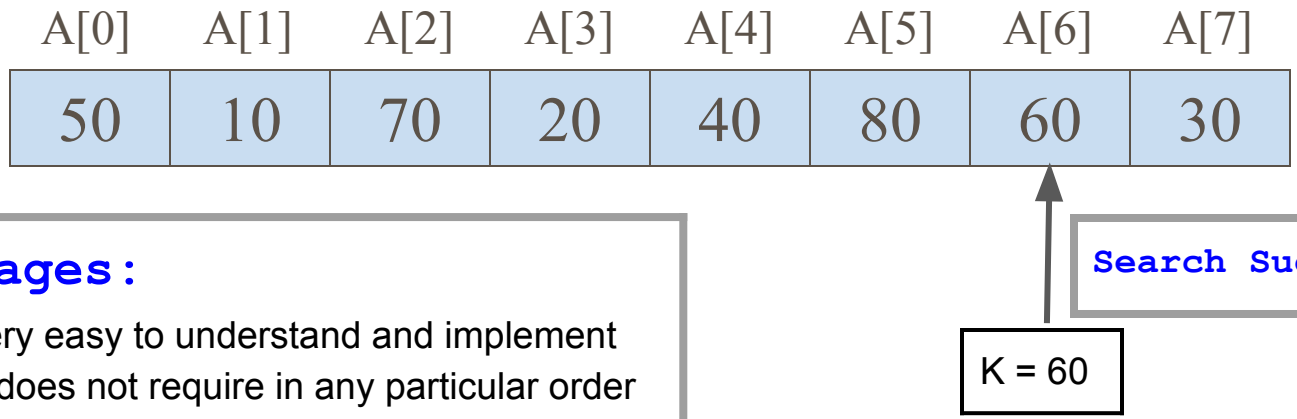
# Linear Search




**Analysis of Linear Search:**

- Best Case
- Worst Case
- Average Case

# Linear Search




**Advantages:**

- It's very easy to understand and implement
- Data does not require in any particular order

**Disadvantages:**

- It has very poor efficiency
- Performance of the algorithm scales linearly with the size of the input
- It is slower than other searching Algorithms

# Linear Search

```
void ls(int array[10],int n,int search)
{
    int i;
    for (i = 0; i < n; i++)
    {
        if (array[i] == search)    /* If required element is found */
        {
            printf("%d is present at location %d.\n", search, i+1);
            break;
        }
     }
     if (i == n)
         printf("%d isn't present in the array.\n", search);
}
```

# Binary Search

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] | A[7] | A[8] |
|---|---|---|---|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |

**Searching Key ( K ) = 70 ?**

# Binary Search

| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] | A[7] | A[8] |
|---|---|---|---|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |

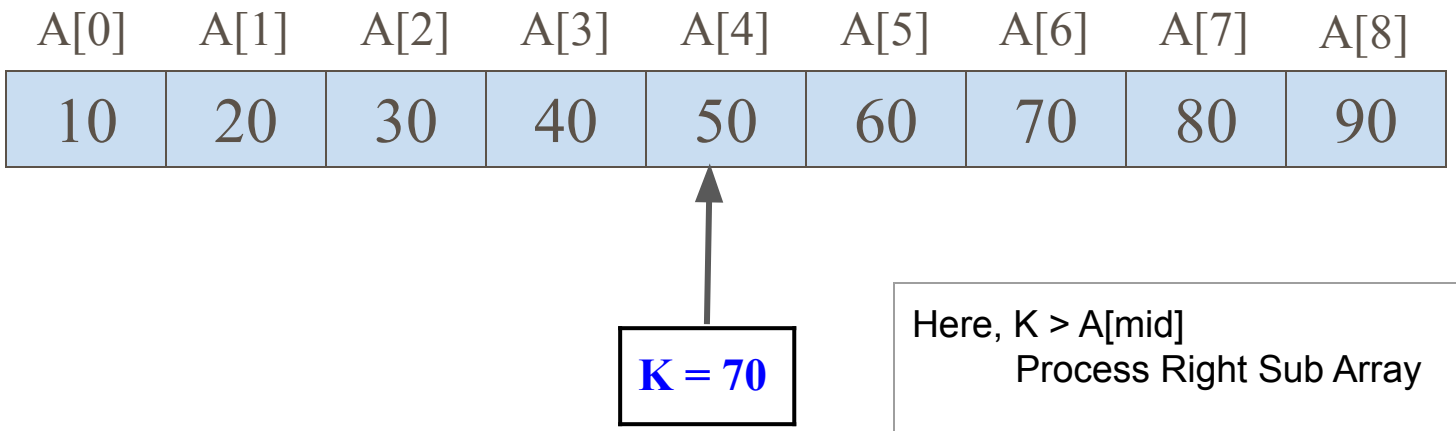
1. Find mid ?
2. Compare mid with key(K)
3. If K = = A [mid]
   successful
4. If K > A [mid]
   search right sub list
   else
   search Left sub list.
5. Stop

First find mid
mid= Floor[(low+high)/2]

Here,
Low = 0, high = 8
mid = Floor[(0+8)/2]
Mid = 4
i.e. A[mid] = A[4] = 50

Then, follow given steps

# Binary Search

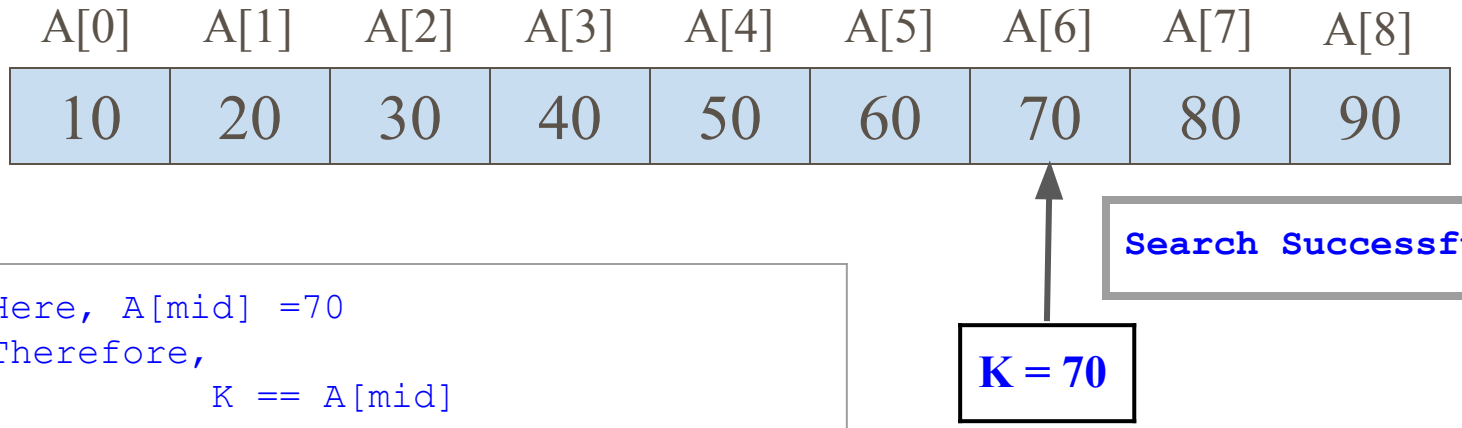



Here, K > A[mid]
Process Right Sub Array

So, here,
Low = 5, high = 8
mid = floor [(5+8) / 2]
mid = 6

Thus
A[mid] = 70

# Binary Search




```
Here, A[mid] =70
Therefore,
          K == A[mid]
             Return Search
Successful
```

**Time Complexity = O ( log(n) )**

# Binary Search

```
void bs(int array[10],int low,int high,int search)
{   int mid;
    mid = (low+high)/2;
    while (low <= high)
    { if (array[mid] < search)
            low = mid + 1;
        else if (array[mid] == search)
        {   printf("%d found at location %d.\n", search, mid+1);
            break;
        }
        else
            high = mid - 1;

        mid = (low + high)/2;
    }
    if (low > high)
        printf("Not found! %d isn't present in the list.\n", search);
}
```