Searching

Let's say we want to find an element in an array.

First thought would be to iterate over the elements in an array and check whether any of its elements matches with element we are searching for.

```
int linear_search(int a[], int n, int key){
        for(int i=0; i<n; i++){
                if(a[i]==key){
                        return i;
                }
        }
        return -1;
}
```

Time complexity of Linear Search : O(n)

```
int linear_search_sorted_array(int a[], int n, int key){
        for(int i=0; i<n; i++){
                if(a[i]==key){
                        return i;
                }else if(a[i] > key){
                        return -1;
                }
        }
        return -1;
}
```

Time complexity of Linear Search in Sorted Array: Still O(n)

Binary Search Algorithm

- Divide and Conquer Algorithm
- One of the unusual Divide and Conquer problem where we will have only one subproblem after division step!
- Won't search the entire array like Linear Search Algorithm
- Prerequisite : Array needs to be sorted

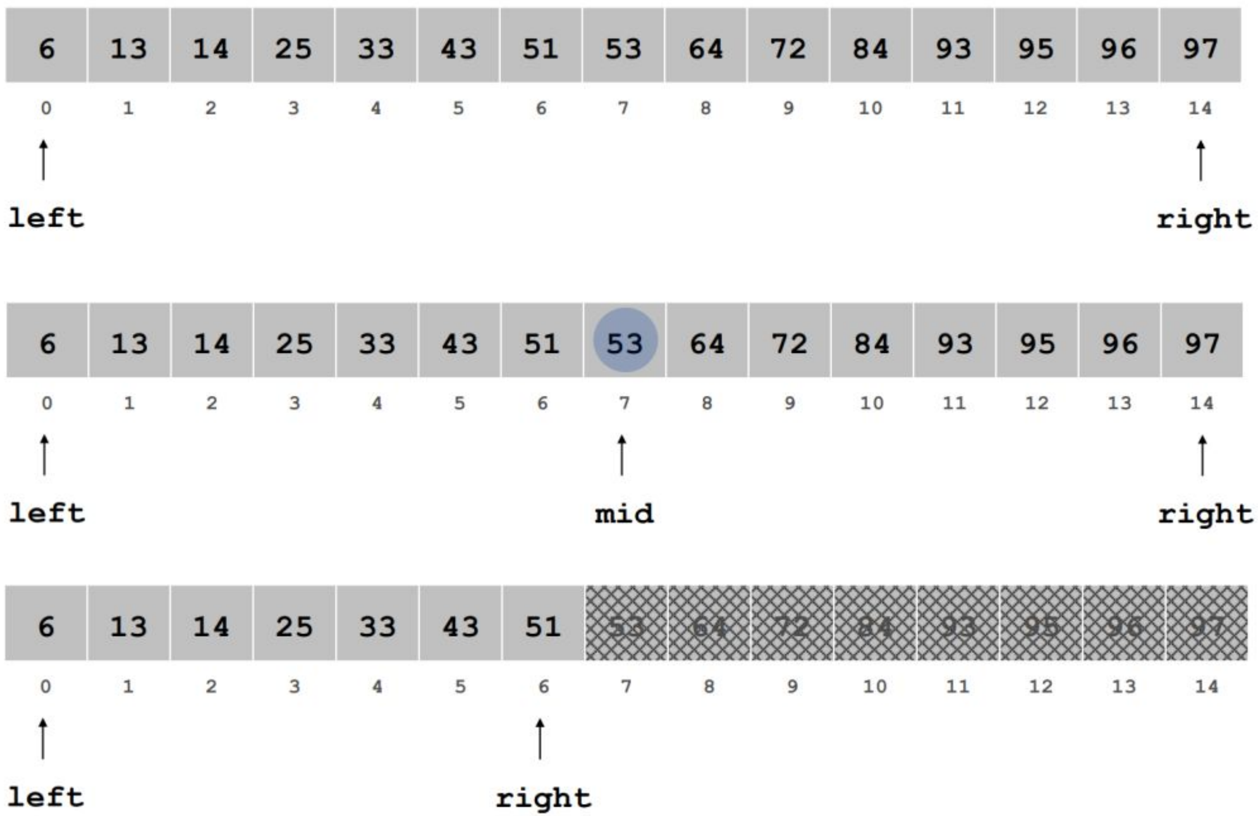
Binary Search Algorithm

Binary search : Given key and sorted array a[], find an index such that a[index] = key, or report that no such index exists.

Algorithm :

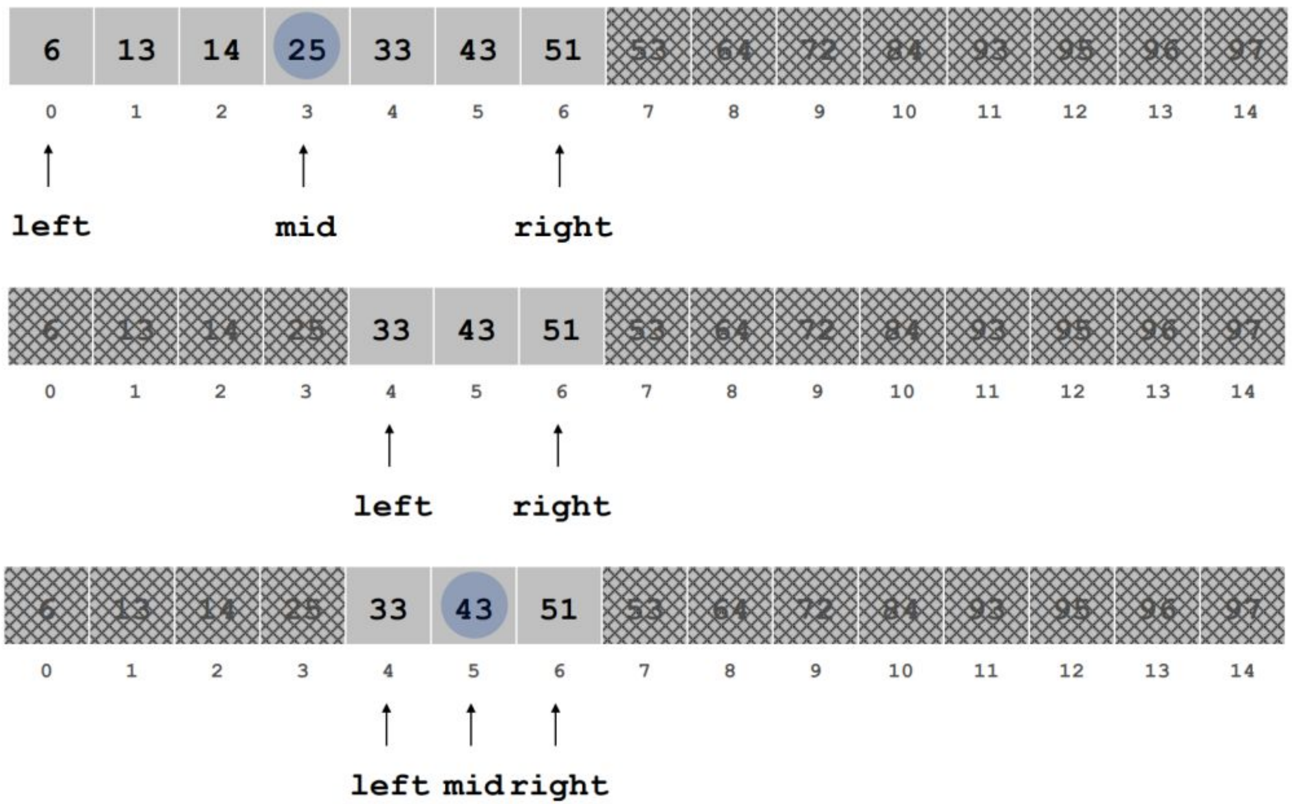
```
int binary_search(int a[], int n, int key){
        int left, right, mid;
        left = 0;
        right = n-1;
        while(left <= right){
                mid = left + (right - left) / 2;
                if(a[mid] == key) {
                        return mid;
                }else if(a[mid] < key) {
                        left = mid + 1;
                }else{
                        right = mid - 1;
                }
        }
        return -1;
}
```

Binary Search Algorithm



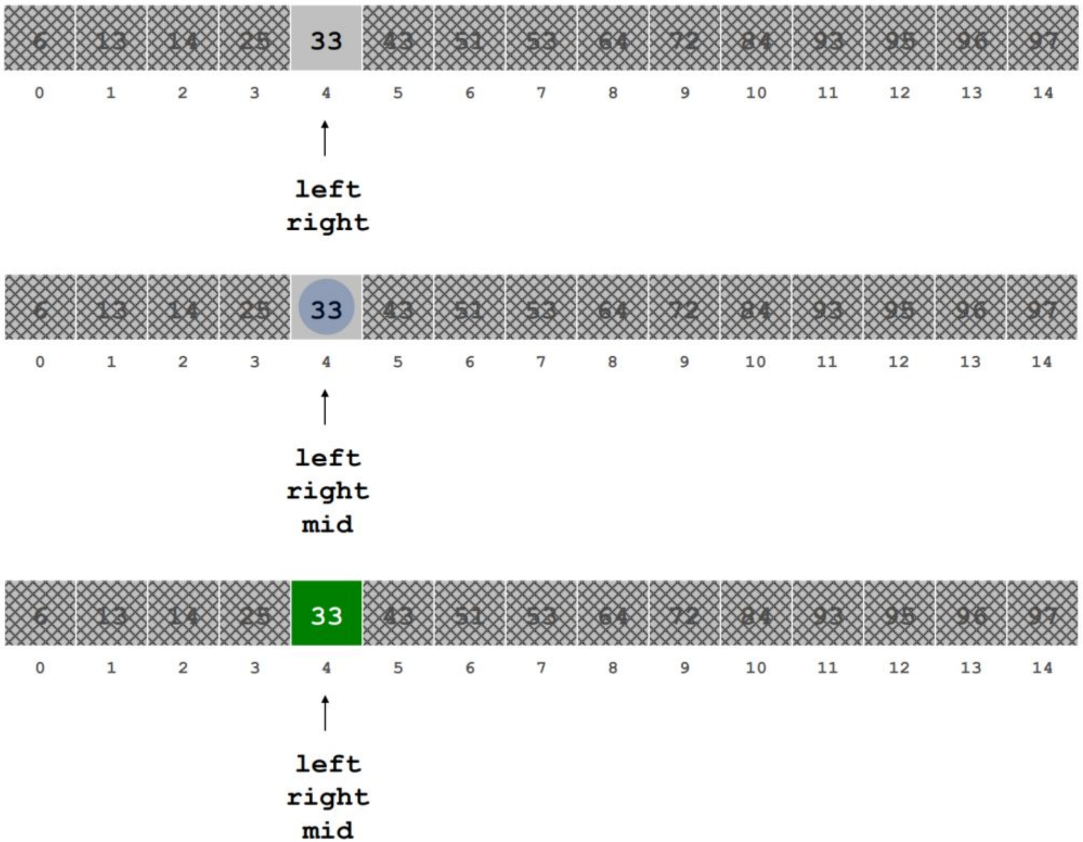

```
int binary_search(int a[], int n, int key){
        int left, right, mid;
        left = 0;
        right = n-1;
        while(left <= right){
                mid = left + (right - left) / 2;
                if(a[mid] == key) {
                        return mid;
                }else if(a[mid] < key) {
                        left = mid + 1;
                }else{
                        right = mid - 1;
                }
        }
        return -1;
}
```

Binary Search Algorithm




```
int binary_search(int a[], int n, int key){
        int left, right, mid;
        left = 0;
        right = n-1;
        while(left <= right){
                mid = left + (right - left) / 2;
                if(a[mid] == key) {
                        return mid;
                }else if(a[mid] < key) {
                        left = mid + 1;
                }else{
                        right = mid - 1;
                }
        }
        return -1;
}
```

Binary Search Algorithm




```
int binary_search(int a[], int n, int key){
        int left, right, mid;
        left = 0;
        right = n-1;
        while(left <= right){
                mid = left + (right - left) / 2;
                if(a[mid] == key) {
                        return mid;
                }else if(a[mid] < key) {
                        left = mid + 1;
                }else{
                        right = mid - 1;
                }
        }
        return -1;
}
```

Binary Search Algorithm

Time Complexity

After first iteration, length of array = n

After second iteration, length of array = n/2

After third iteration, length of array = (n/2)/2 = $n/2^2$

....

After k^{th} iteration, length of array = $n/2^k$

Length of array becomes 1 after k iterations.

$n/2^k = 1$

=> $n = 2^k$

=> $\log_2(n) = \log_2(2^k)$

=> $\log_2(n) = k\log_2{}^2$

=> $k = \log_2{}^n$

Time complexity of Binary Search = $O(\log_2{}^n)$