



Binary Search is only applicable for 'monotonic search spaces'. For example, B.S. can be applied for searching an element in a SORTED ARRAY only.

Eg: arr[]: {1, 3, 3, 6, 7, 10} \rightarrow monotonically increasing search space

arr[] : {1, 3, 3, 6, 7, 10} search = 7

- arr[]: {1, 3, 3, 6, 7, 10}

- $$\text{mid} = (\text{low} + \text{high}) / 2; \quad \rightarrow (0 + 5) / 2 = 2$$

arr[]: {1, 3, 3, 6, 7, 10}

- ③ Check the following :

(i) If $\text{arr}[\text{mid}] == \text{search}$, return mid , you found it!

(ii) If $\text{arr}[\text{mid}] < \text{search}$, then all elements $\text{arr}[0 \dots \text{mid}]$ will definitely be smaller than search . Thus, update your search space : $\text{low} = \text{mid} + 1$.

(You have discarded $0 \dots \text{mid}$ indices)

(iii) If $\text{arr}[\text{mid}] > \text{search}$, then all elements $\text{arr}[\text{mid} \dots n-1]$ will definitely be greater than search . Thus, update your search space : $\text{high} = \text{mid} - 1$.

(You have discarded $\text{mid} \dots n-1$ indices)

0 1 2 3 4 5
 $\text{arr}[] : \{1, 3, 3, 6, 7, 10\}$
low mid high

$\text{arr}[\text{mid}] = 3$ and $3 < 7$
 $\text{low} = \text{mid} + 1 = 3$.



0 1 2 3 4 5
 $\text{arr}[] : \{1, 3, 3, 6, 7, 10\}$
mid low high

④ Repeat steps ② and ③ untill $\text{low} \leq \text{high}$.

0 1 2 3 4 5
 $\text{arr}[] : \{1, 3, 3, 6, 7, 10\}$
low mid high

$\text{arr}[\text{mid}] == 7$, return mid ;

return 4;

EXAMPLE : $\text{arr}[] : \{4, 8, 16, 32, 64\}$ $K = 4$

① $\{4, 8, 16, 32, 64\}$
low mid high

② $(arr[mid] = 16) > 4 \Rightarrow high = mid - 1.$

$\{4, 8, 16, 32, 64\}$
↑ ↑ ↑
low high mid

③ $(arr[mid] = 8) > 4 \Rightarrow high = mid - 1$

mid
↓
 $\{4, 8, 16, 32, 64\}$
↑ ↑
low high

④ $(arr[mid] = 4)$ We found it!

Note: If $low > high$ and we haven't found the element, then the element doesn't exist at all.

Code :

```
int binarySearch(int arr[], int size, int key) {  
  
    int start = 0;  
    int end = size-1;  
  
    int mid = (start + end)/2;  
  
    while(start <= end) {  
  
        if(arr[mid] == key) {  
            return mid;  
        }  
  
        //go to right wala part  
        if(key > arr[mid]) {  
            start = mid + 1;  
        }  
        else{  
            end = mid - 1;  
        }  
  
        mid = (start+end)/2;  
    }  
    return -1;    I
```

```
int even[6] = {2,4,6,8,12,18};
int odd[5] = {3, 8, 11, 14, 16};

int evenIndex = binarySearch(even, 6, 20);

cout << " Index of 18 is " << evenIndex << endl;

int oddIndex = binarySearch(odd, 5, 8);

cout << " Index of 8 is " << oddIndex << endl;
```

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ binarySearch.cpp -
o binarySearch && "/Users/lovebabbar/"binarySearch
Index of 18 is -1
Index of 8 is 1
```

CAUTION :

What if you had an array of size $2^{31} - 1$ and during some iteration :

$$\text{low} = 2^{31} - 2 \quad \text{and} \quad \text{high} = 2^{31} - 1,$$

$$\text{mid} = \frac{\text{low} + \text{high}}{2}$$

will throw integer overflow error because $2^{31} + 2^{31} - 1 = 2^{32} - 1$

which is greater than INT_MAX. Thus, we can either use long long or even better,

$$\text{mid} = \text{low} + \frac{\text{high} - \text{low}}{2}$$

New Code :

```
int binarySearch(int arr[], int size, int key) {
    int start = 0;
    int end = size-1;

    int mid = start + (end-start)/2;

    while(start <= end) {
        if(arr[mid] == key) {
            return mid;
        }

        //go to right wala part
        if(key > arr[mid]) {
            start = mid + 1;
        }
        else{ //key < arr[mid]
            end = mid - 1;
        }

        mid = start + (end-start)/2;
    }
    return -1;
}
```

Linear Search v/s Binary Search :

Linear Search's Time Complexity = $O(n)$.

In Binary Search, we are essentially halving our search space in every iteration.

$$\Rightarrow n \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \frac{n}{8} \rightarrow \dots \rightarrow 1.$$

Applying Geometric Progression :

$$a = n$$

$$r = \frac{1}{2}$$

we want to know the no. of elements passed by the time we reach 1.

Say 1 is the z^{th} element.

$$ar^{z-1} = 1$$

$$\rightarrow n \left(\frac{1}{2}\right)^{z-1} = 1$$

$$\rightarrow 2^{z-1} = n$$

$$\rightarrow 2^z = 2n$$

\rightarrow Taking \log_2 on both sides :

$$\log 2^z = \log 2n$$

$$\rightarrow z = \log(2n) \approx \boxed{O(\log n)}$$