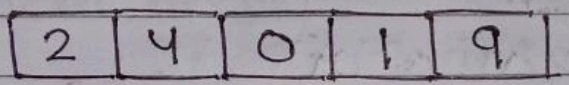
# Linear search

17-02-2023
Date.....

Linear search is a sequential search algorithm where we start from one end & check every element of the list until the desired element is found.

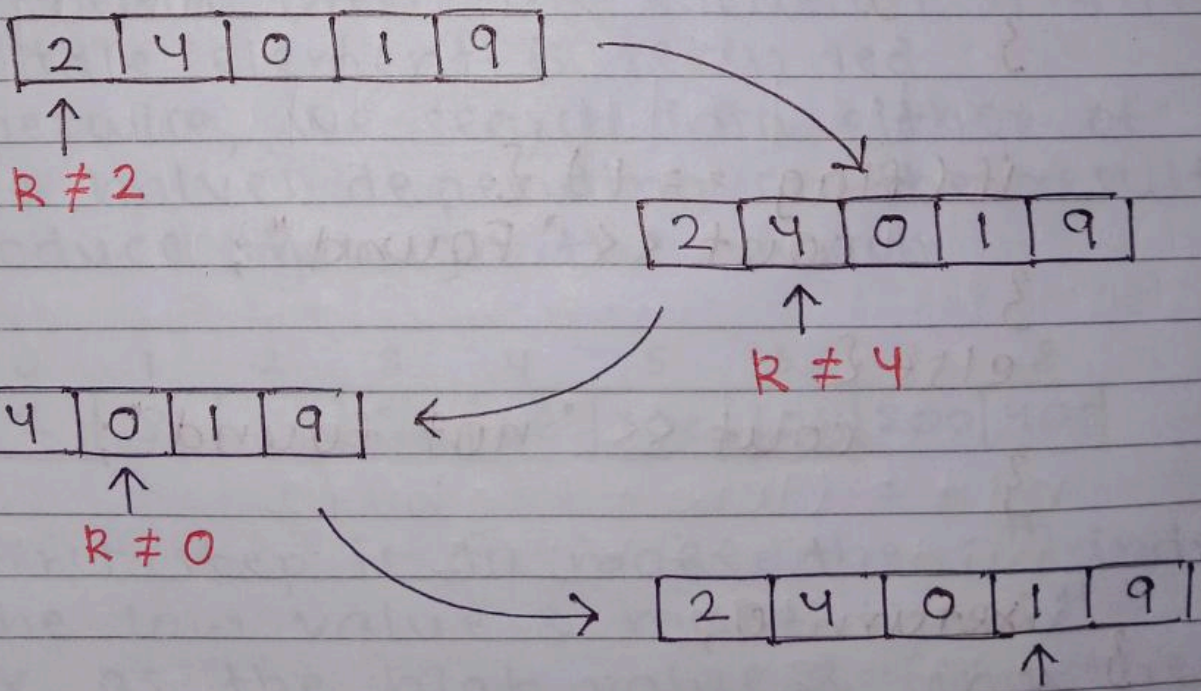
* How Linear search work ?
Suppose, we have an array :-




} this is the array to be searched for.

K = 1 → search item.

step 1, start from the first element, compare 'K' with each element of the array.




when "x == K", return the index, else print "not found".

Date.....

code :-

```
#include<iostream>
using namespace std;
int main()
{
    int arr[] = {2, 4, 0, 1, 9};
    int k = 1;
    int flag = 0;

    for(int i = 0; i<5; i++){
        if(arr[i] == k){
            flag = 1;
            break;
        }
        else{
            flag = 0;
        }
    }

    if(flag == 1){
        cout << "Found";
    }
    else{
        cout << "not found";
    }

    return 0;
}
```

# Binary Search

Date.....

Binary search is an optimized search algorithm for finding an element in an array.

If the array is not sorted, we need to sort them first.

there is only one condition of Binary search, i.e., the array must be sorted, either in ascending or desce -nding order.

* Binary search follows the divide & conquer approach in which the list is divided into two halves & the item is compared with the middle element of the list. If the match is found then, the location of the middle element is returned. Otherwise, we search into either of the halves depending on the result produce through the match.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 2 | 8 | 14 | 32 | 66 | 100 | 104 | 200 | 400 |

The first step is to make the $0^{th}$ index as the low value & make the $8^{th}$ index as the high value & find the mid value.

Date.....

Mid value is finded by,

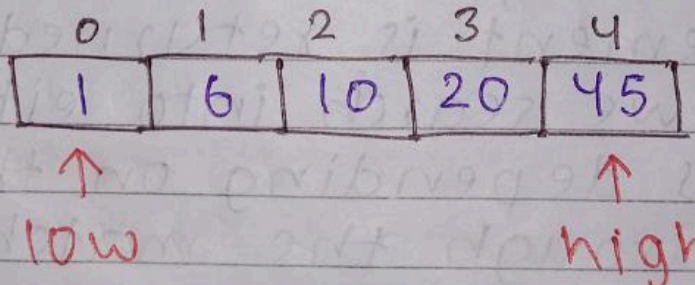
$$(low + high) / 2$$

by using this formula, there may be chance to get Integer Overflow Error, if low & high both are having INT_MAX. So, alternatevely, we can use one more formula,

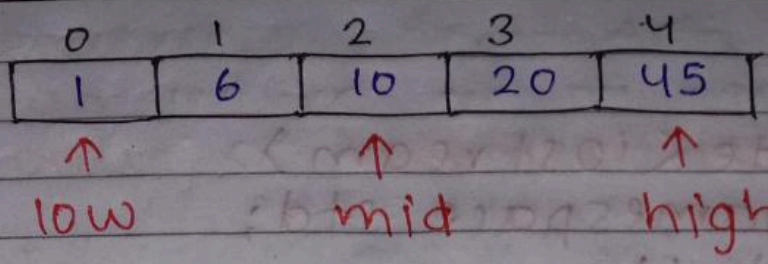
$$low + (high - low) / 2$$

For Example,

R = 1



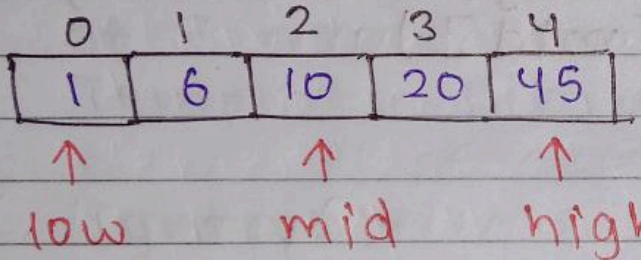

mid = (low + high) / 2
mid = (0 + 4) / 2
mid = 4 / 2
mid = 2




Now, check "arr[mid] = R", if it is equal then return mid.

If "arr[mid] > R", then we shift our end to mid - 1, & again search.

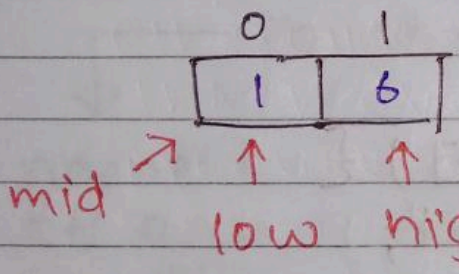
If "arr[mid] < R", then we shift our start to mid +1 & again search.




→ arr[mid] ≠ R

↓

R < arr[mid], so we search at left part & shift high to mid - 1.




←

↓

arr[mid] = R → so, we return the mid value, because that is the element of the array which we want to search.



Teacher's Sign .....

Date.....

code :-

```
#include<iostream>
using namespace std;
int main()
{
   int arr[] = {10, 19, 23, 40, 49};
   int n = 5;
   int R = 19;
   int low = 0, high = n-1;
   int mid, flag = 0;

   while(low <= high){
      mid = low + (high - low)/2;
      if(R == arr[mid])
      {
         flag = 1;
         break;
      }
      else
      {
         if(R < arr[mid]){
            high = mid - 1;
         }
         else{
            low = mid + 1;
         }
      }
   }
```

Date.....

```
if(flag == 1){
    cout << "element found";
}
else{
    cout << "not found";
}

return 0;
}
```

* Another approach to search an element using Binary search is using a predefined function of Binary Search in STL (Standard Template Library).

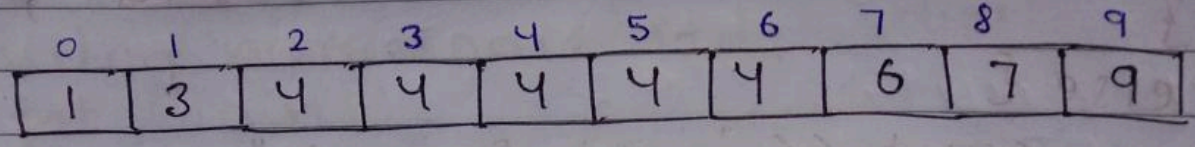
Syntax:-

```
binary_search(v.begin(), v.end(), 3);
```

- `binary_search` → name of a function.
- `v.begin()` → starting address of vector/array.
- `v.end()` → ending address of vector/array.
- `3` → search item.

Date.....

Ques. Find the first occurence of an integer in an array.




target = 4.

First occurence of 4 is at 2nd index.

we can iterate over the array linearly & compare arr[i] with our target, when we find an element just return the index of the element.

code :- (using Linear Search Approach)

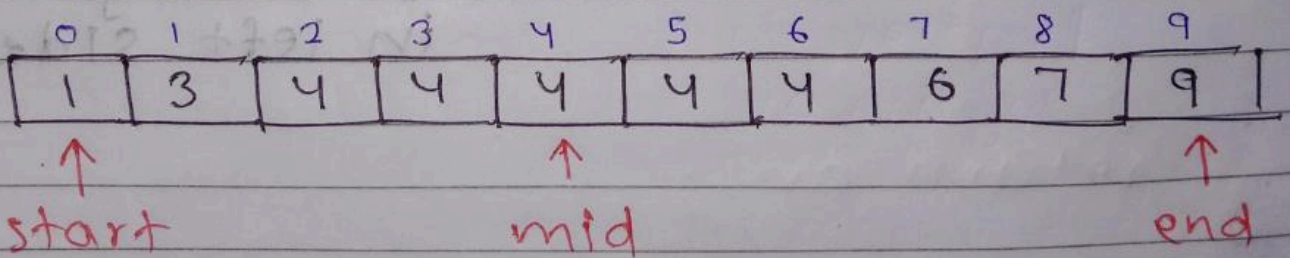
```
#include<iostream>
using namespace std;

int firstOcc(int arr[], int n, int t)
{
    for(int i=0; i<n; i++)
    {
        if(arr[i] == target)
        {
            return i;
        }
    }
}
```

Teacher's Sign

```
int main()
{
    int arr[] = {1, 3, 4, 4, 4, 4, 4, 6, 7, 9};
    int n = 10;
    int target = 4;

    int index = firstOcc(arr, n,
                         target);

    cout << "\nFirst occurrence of " <<
     target << " is at " << index <<
     " index " << endl;

    return 0;
}
```

can we solve the above problem using Binary?
The given array is sorted in ascending order means we can apply Binary search on this, that means we have to find out the formula only.

First we find the mid element,




```
mid = (start + end) / 2
mid = (0 + 9) / 2
mid = 4
```

Spiral

Teacher's Sign .....

Date.....

Now we compare arr[mid] with target.

if(arr[mid] == target)

↳ this returns true.

↓

but if we return the index of this item the our output is wrong because we have to return the index of first occurrence.

when we find our element, means when arr[mid] == target then we have to do two things :-

↓ ①

① store that index

↓ ②

② search in left side of array, because first occurrence is always found in left side.

Date.....

If we find that our arr[mid] == target then also we have to check the left side again because there is a possibility that our element could also be placed at left side also

so, our conditions are :-

(1)
```
if(target == arr[mid])
{
    ans = mid;
    end = mid - 1;
}
```

(2)
```
if(target > arr[mid])
{
    start = mid + 1;
}
```

(3)
```
if(target < arr[mid])
{
    end = mid - 1;
}
```

Date.....

code :- (using Binary Search Approach)

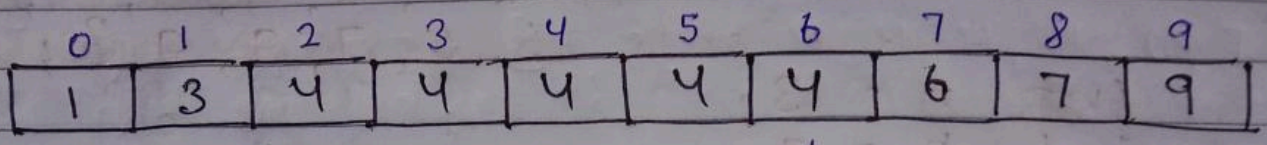
```
#include<iostream>
using namespace std;

int firstOccurrence(int arr[], int n,
                    int R)
{
    int start = 0, end = n-1;
    int mid, ans = -1;
    while(start <= end)
    {
        mid = (start + end)/2;
        if(R == arr[mid])
        {
            ans = mid;
            end = mid -1;
        }
        else{
            if(R > arr[mid]){
                start = mid + 1;
            }
            else{
                end = mid - 1;
            }
        }
    }
    return ans;
}
```

Date 1.9.12

```
int main()
{
    int arr[] = {1, 3, 4, 4, 4, 4, 4, 6,
                 6, 6, 7, 9, 9, 3};
    int n = 13;
    int k = 9;

    int ans = firstOccurrence(arr,
                              n, k);

    if(ans == -1){
        cout << "element not found";
    }
    else{
        cout << "First occurrence of"
             << k << " is at " << ans <<
        "index" << endl;
    }

    return 0;
}
```

Date.....

Ques Find the last occurrence of an integer in an array.




target = 4

last occurrence of 4 is at 6th index.

using the same previous logic of Binary search, we can also find out the last occurrence of an element. But there is a minor change in a condition, i.e.,
when our arr[mid] == target then instead of searching in left side, now we search in the right side,

```
if(target == arr[mid])
{
    ans = mid;
    start = mid + 1;
}
```

Date.....

code:- (using Binary search Approach)

```
#include<iostream>
using namespace std;

int lastOccurrence(int arr[], int
                   n, int K)
{
    int start = 0, e = n-1;
    int mid, ans = -1;
    while(start <= e)
    {
        mid = (start + end) /2;
        if(K == arr[mid])
        {
            ans = mid;
            start = mid +1;
        }
        else
        {
            if(K > arr[mid]){
                start = mid +1;
            }
            else{
                e = mid -1;
            }
        }
    }
    return ans;
}
```

Date.....

```
int main()
{
    int arr[] = {1, 3, 4, 4, 4, 4, 4, 6, 7};
    int n = 9;
    int k = 4;
    int ans = lastOccurrence(arr, n,
                             k);

    if(ans == -1)
    {
        cout << "not found";
    }
    else
    {
        cout << "last occurrence of " <<
            k << " is at " << ans <<
            "index" << endl;
    }

    return 0;
}
```

Date.....

Now, there is one more approach to find the first & last occurrence of an element.
There is a predefind functions in STL to find the first & last occurrence of an element in an array.

- lower_bound → to find the first occurrence of an element
- upper_bound → to find the last occurrence of an element.

Syntax :-

```
lower_bound(v.begin(), v.end(), 3);
```

- lower_bound → name of function
- v.begin() → start address
- v.end() → end address
- 3 → search item.

Example :-

```
int ans = lower_bound(v.begin(), v.end(), 5);
cout << "Lower Bound is " << ans - v.begin();
```

Date.....

Ques. Find the total occurrence of an integer in an array.

[Total occurrence = last occurrence - first occurrence + 1]

code:-

```
int main()
{
    int arr[] = {1, 3, 4, 4, 4, 4, 4, 6, 6, 6,
                 7, 9, 9};
    int n = 13;
    int k = 9;

    auto firstAns = firstOccurrence(arr,
                    n, k);
    auto lastAns = lastOccurrence(arr,
                   n, k);
    int totalAns = lastAns - firstAns + 1;

    cout << "\n Total occurrence of " <<
        k << " in arrays is : " <<
                    totalAns;

    return 0;
}
```

Date.....

Ques. Smallest missing number.
or
Find missing number.

In this problem, we've given a sequen -ce of numbers & one number is missing from that sequence. we have to find out that missing number.
For example :-

{1, 2, 3, 5, 6, 7, 8}

↓

Here, all the numbers are present from 1 to 8, but 4 is not present. It means, 4 is missing.

If we have given a number, lets say n = 8, then the formula to find total sum of n numbers is,

$$\boxed{n * (n+1) / 2}$$

Date.....

Now, if we know in our case, we have 7 numbers but there should be 8 numbers, means we have a missing number, then we can calculate the missing number by applying the formula.

$$\frac{(n+1) * (n+2)}{2}$$

code :-

```
#include<iostream>
using namespace std;

int missingNum(int arr[], int n)
{
    int total = (n+1)*(n+2)/2;
    int sum = 0;

    for(int i=0; i<n; i++)
    {
        sum += arr[i];
    }

    int miss = total - sum;

    return miss;
}
```

Date.....

```
int main()
{
    int arr[] = {1, 2, 3, 5, 6, 7, 8};
    int n = 7;

    int miss = missingNum(arr, n);

    cout << "missing number in
        array is: " << miss;

    return 0;
}
```

Date.....

Ques Find Peak Element.

Peak element is the element which is greater than its left as well as its right side element.

code:- (simple approach)

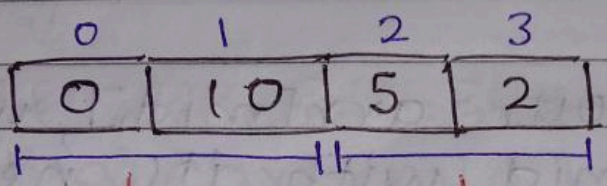
```
if(arr[0] > arr[1])
{
    cout << arr[0];
}

if(arr[n-1] > arr[n-2])
{
    cout << arr[n-1];
}

for(int i = 0; i < n; i++)
{
    if(arr[i] > arr[i-1] &&
            arr[i] > arr[i+1])
    {
        cout << arr[i];
    }
}
```

Above are the 3 conditions on which we can find out the peak element simply.

Date.....

* How to approach peak element problem using Binary Search?

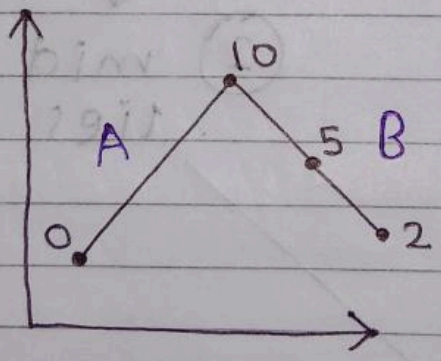
Suppose, we have an array :-




this part of the array is sorted in increasing order.

this part of the array is sorted in descend -ing order.

If we draw the graph of the above array then it is like :-




If we observe Line A, in this line every arr[i] < arr[i+1]

If we observe Line B, in this line every arr[i] > arr[i+1]

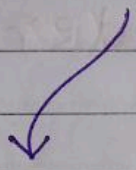
Date.....

For Peak element, the condition is,

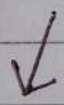
```
arr[i-1] < arr[i] > arr[i+1]
```

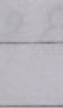
After finding our arr[mid], we can compare our mid with its next entering element,

```
if(arr[mid] > arr[mid+1])
```



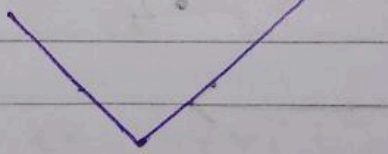
after comparing the mid, we have 2 conditions,





① mid element may be a peak element.

② mid element lies in Line B



Because, both conditions lies in our if condition.

Date.....

If we write the reverse condition, i.e.,

```
if(arr[mid] < arr[mid + 1])
```

↙

Here we can confidently say that arr[mid] is not the Peak Element, because peak element is not less than any element.

So, in this situation we have to check to the right side, means

[start = mid + 1]

Date.....

code:-

```
#include <iostream>
using namespace std;
int main()
{
    int arr[] = {15, 185, 44, 136, 199,
                                88, 69};
    int n = 7;
    int start = 0, end = n-1, mid;

    while(start < end)
    {
        mid = (start + end) / 2;
        if(arr[mid] < arr[mid + 1])
        {
            start = mid + 1;
        }
        else
        {
            end = mid;
        }
    }

    cout << "Peak element is " <<
                        arr[start];

    return 0;
}
```