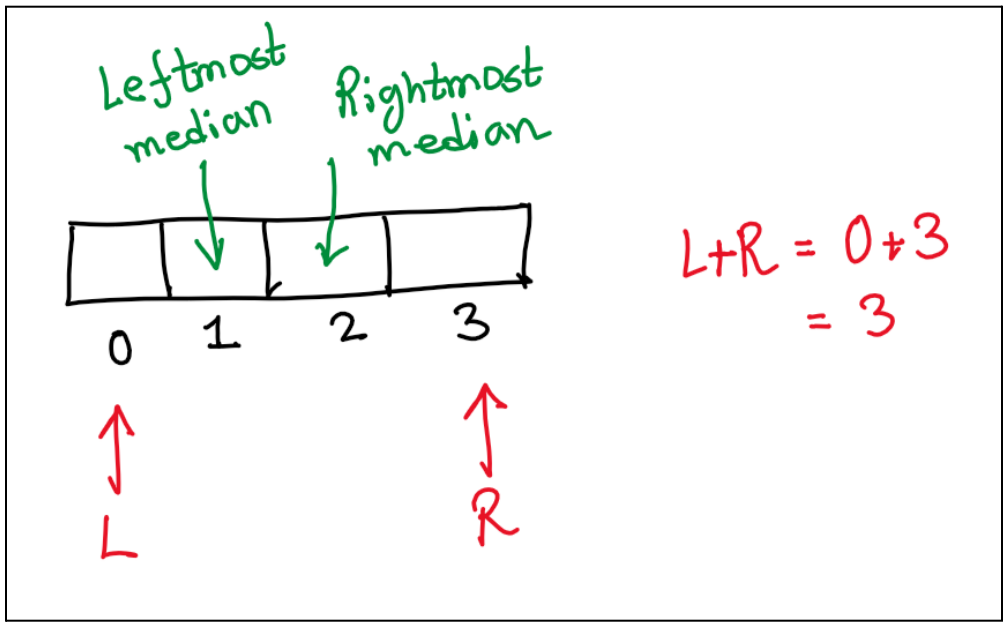
## Before jumping into the variation of binary search, let's clarify some basic concepts first:

1. **M = (L + R) / 2 is risky. It can create overflow issues.**

2. **M = L + (R - L) / 2 is safe. It is computationally the same as no 1, but it has no overflow issue.**

3. **If (L+R) is ODD, then there will be an even number of elements in total. So, there will be two median values. Let's look at the following figure for a better understanding:**



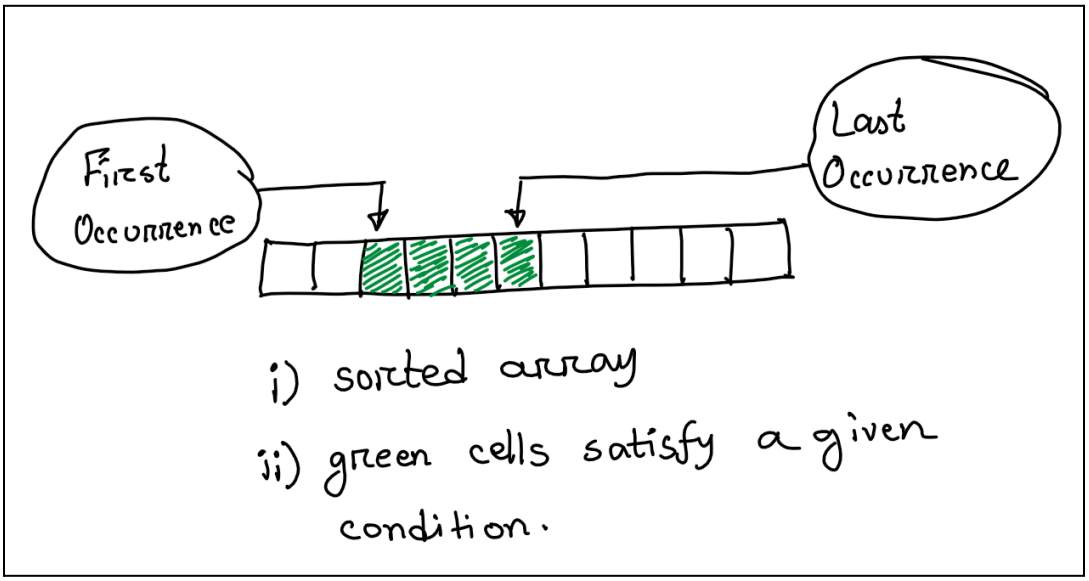

**If you divide (L+R) by 2, you will always get the leftmost median in such scenarios. But what if we want to get the rightmost median? :D**

**The answer is : M = (L+R+1)/2 :D**

## Types of Binary Search:

1. **Find the target element from an array.**
2. **Find the first occurrence of a number that satisfies a given condition.**
3. **Find the last occurrence of a number that satisfies a given condition.**

**To understand type 2 and 3, refer to the following figure:**




---

# Sample Code Template

1. **Find the target element from an array:**

```cpp
bool doesExist(vector<int>&vec, int target)
{
    int L = 0;
    int R = vec.size() - 1;

    while(L <= R)
    {
        int M = L + (R - L) / 2;
        if(vec[M] == target) return 1;
        if(vec[M] < target)
        {
            L = M + 1;
        }
        else
        {
            R = M - 1;
        }
    }
    return 0;
}
```

2. **Find the first occurrence of a number that satisfies a given condition (considering at least one number exists in the array that satisfies the given condition) :**

```
int first_occurence(vector<int>&vec, int target)
{
    int L = 0;
    int R = vec.size() - 1;

    while(L < R)
    {
        int M = L + (R - L) / 2;
        if(vec[M] < target)
        {
            L = M + 1;
        }
        else
        {
            R = M;
        }
    }
    return L;
}
```

3. **Find the last occurrence of a number that satisfies a given condition (considering at least one number exists in the array that satisfies the given condition) :**

```cpp
int last_occurence(vector<int>&vec, int target)
{
    int L = 0;
    int R = vec.size() - 1;

    while(L < R)
    {
        int M = L + (R - L + 1) / 2;
        /**
            this is equivalent to M = (L + R + 1) / 2
            we are using this because we want to go as
            right as we can. :D
        **/
        if(vec[M] > target)
        {
            R = M - 1;
        }
        else
        {
            L = M;
        }
    }
    return R;
}
```

**Note:**

1. **lower_bound = first occurrence of a number smaller or equal to a given number.**
2. **upper_bound = first occurrence of a number smaller or equal to a given (number + 1)**
3. **Hence, lower_bound(value + 1) == upper_bound(value)**
4. **type 2 and 3 do not entirely reflect the idea to find the lower and upper bound. But one can easily change the code of type 2, to find out lower_bound from an array. If you have the code for lower_bound, you don't need to write another function for upper_bound. :). Just change R = vec.size() in type 2 code and you're done.**