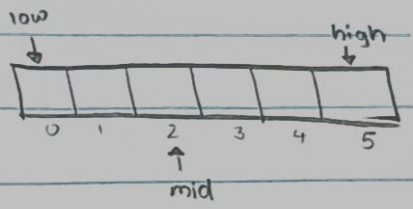
VII Binary Search

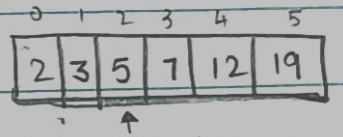


$\frac{0+5}{2} = 2$

- applicable only for unique sorted array
- Trimming down search space

iterative

```
func( arr, n., target){
    low = 0; high = n-1;
    while (low <= high){
        mid = (low + high)/2;
        if (arr[mid] == target) return mid;
        if (arr[mid] < target){
            low = mid+1; }
        else { high = mid - 1; }
    return -1;
}
```



low = 0 high = 5

↳ while (0 <= 5)

mid = $\frac{0+5}{2}$ = 2

arr[mid] = 5

5 == 12 ✗

5 < 12 ✓

low = 2+1 = 3

high = 5

again ↳ while (3 <= 5)

mid = $\frac{3+5}{2}$ = 4

arr[mid] = 12

arr == 12 give mid

mid → low → +1

high ← -1

answer will come as mid

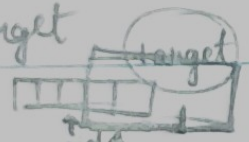
HiQ

Recursive

```
func (arr, low, high, target) {
    if (low > high) {
        return -1;
    } ;
    mid = (low + high)/2
    if (arr[mid] == target)
        { return mid; }
    else if (arr[mid] < target) {
        return func (arr, mid+1, high, target);
    }
    return func (arr, low, mid-1, target);
}
}
```

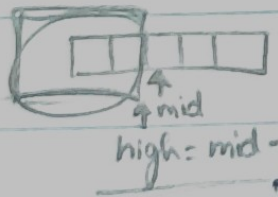
* mid = low + (high - low)/2

mid < target



low = mid+1

low +1



high = mid-1

high -1

The trick is

ans

The trick is (ans)

Lower Bound: smallest index such that `arr[index] >= x`

Upper Bound: smallest index such that `arr[index] > x`

```
bool found = binary_search(arr.begin(), arr.end(), target)
auto lb = lower_bound(arr.begin(), arr.end(), target);
auto ub = upper_bound(arr.begin(), arr.end(), target);
```

(lb)

```
if (arr(mid) >= x)
    ans = mid;
    high = mid-1;
else
    low = mid+1;
```

(ub)

```
if (arr[mid] > n)
    ans = mid;
    high = mid-1;
else
    low = mid+1;
```