C++
Binary Search

Lecture-24

Raghav Garg

What and Why?

Search space

arr = {1, 2, 4, 5, 9, 15, 18, 21}

7 ops

target = 18

```
bool flag = false; //target not present
for(int i=0; i<n; i++){
    if(arr[i] == target){
        flag = true;
        break;
    }
}
if(flag == true) cout << "Present";
else cout << "Not Present";
```

linear search

T.C. = O(n)

Binary search algorithm → Works only if array is sorted (ascending or descending)

	0	1	2	3	4	5	6	7	8
arr =	1	2	4	5	9	15	18	21	24

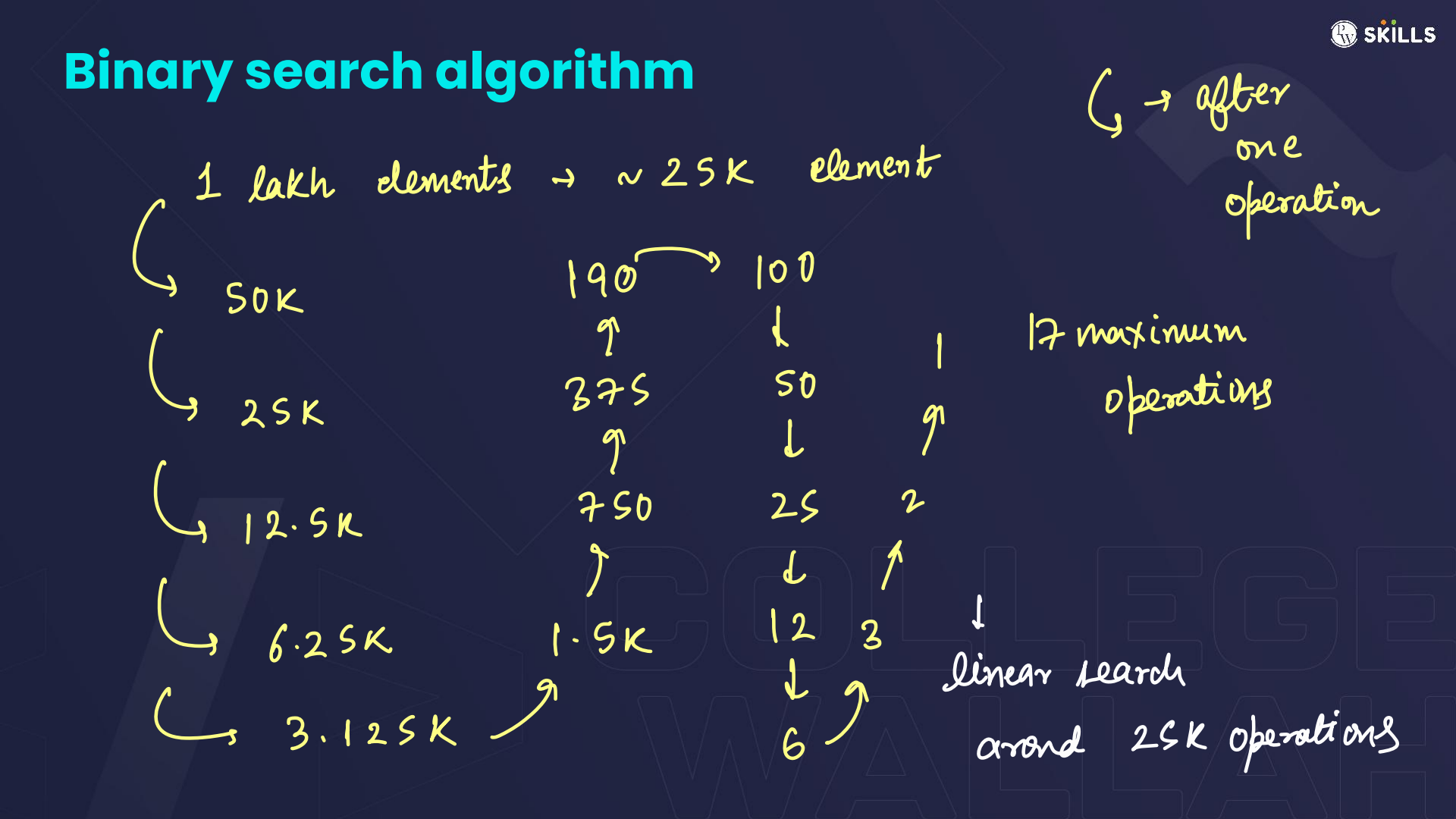
lo
hi
mid

target = 15

```
int lo = 0;
int hi = n-1;
```

```
int mid = (lo+hi)/2;
if(arr[mid] == target) return true;
if (arr[mid] < target) lo = mid+1;
if (arr[mid] > target) hi = mid-1;
```

Binary search algorithm



Ques : Binary Search [Leetcode 704]

```
int lo = 0;
int hi = n-1;
while (lo<=hi) {
    int mid = lo + (hi-lo)/2;
```

$$\frac{lo+hi}{2} = \frac{lo+lo+hi-lo}{2} = \frac{lo+lo}{2} + \frac{hi-lo}{2}$$

$$= lo + (hi-lo)/2$$

Time complexity analysis

Total no. of elements = n

Total no. of operations = x

$$n \to \frac{n}{2} \to \frac{n}{4} \to \frac{n}{8} \to \frac{n}{16} \to \frac{n}{32} \cdots\cdots 1$$

$\sim x$ terms

$$\Rightarrow \frac{n}{2^x} = 1$$

$$\Rightarrow 2^x = n$$

$$\Rightarrow x = \log_2 n$$

$$n \to \frac{n}{2^1} \to \frac{n}{2^2} \to \frac{n}{2^3} \to \cdots \frac{n}{2^x}$$

T.C. $\to O(x) = \boxed{O(\log n)}$

*Lower bound

Ques: Given a sorted integer array and an integer 'x', find the lower bound of x.

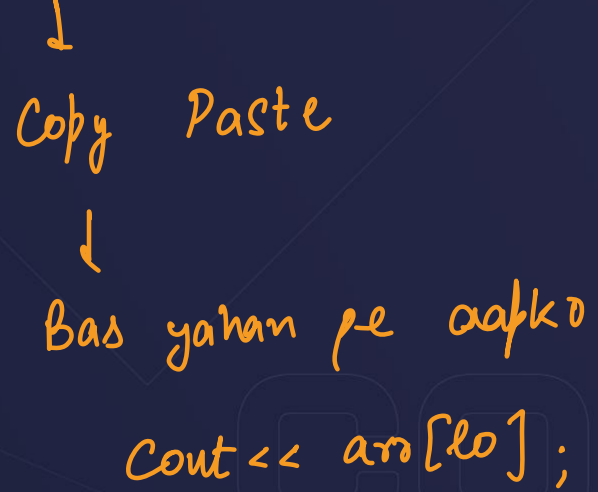

```
while (lo <= hi){
    mid = lo + (hi-lo)/2 ;
    if(arr[mid] == x)   cout << arr[mid-1];
    else if (arr[mid] < x)   lo = mid+1
    else if (arr[mid] > x)   hi = mid-1   cout << arr[hi];
}
```

lower_bound(arr.begin() , arr.end() , target)

vector

*Upper bound

Ques: Given a sorted integer array and an integer ‘x’, find the upper bound of x.



upper_bound(arr.begin() , arr.end() , target)

Ques :Given a sorted array of n elements and a target 'x'. Find the first occurrence of 'x' in the array. If 'x' does not exist return -1.

	0	1	2	3	4	5	6	7	8	9	10	11	12
arr = {	1,	2,	2,	3,	3,	3,	3,	3,	4,	4,	5,	8,	9 };

x = 3

lo
hi
mid

```
while (lo <= hi){
    int mid = lo + (hi-lo)/2;
    if (arr[mid] == x) → arr[mid-1] == 3 → hi = mid-1 ✓
                         ✗ → yes this is the first occurence
    if (arr[mid] > x)  hi = mid-1;
    if (arr[mid] < x)  lo = mid+1;
}
```

3

Ques : Given a sorted array of non-negative distinct integers, find the smallest missing non-negative element in it.

O(n) &
O(logn)

	0	1	2	3	4	5	6	7
arr =	0	1	2	3	4	8	9	12

ans = 5

```
for(int i = 0; i<n; i++){
    if(i != arr[i]) return i;
}
```

Ques : Given a sorted array of non-negative distinct integers, find the smallest missing non-negative element in it.

2nd approach : Binary Search $O(\log n)$

	0	1	2	3	4	5	6	7
arr =	0	1	2	3	4	8	9	12
					lo	mid		hi

```
mid = lo + (hi-lo)/2;
if(arr[i] == i ) lo = mid + 1;
if (arr[i] != i) cout << i , break
```


```
if (arr[mid] == mid) lo = mid + 1;
else {
    ans = mid;
    hi = mid - 1;
}
```

ans = 4 2

Ques : Sqrt(x)

[Leetcode 69]

$x = 36$

int lo = 0 ; 4 6

int hi = 36 ; 17 7

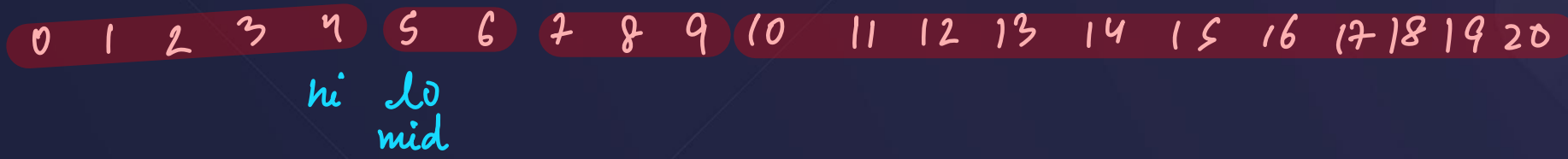
mid = 18 8 3 5 6

```
while (lo <= hi) {
    int mid =
    if( mid * mid == x ) return mid;
    if (mid*mid > x ) hi = mid - 1;
    if (mid*mid < x) lo = mid + 1;
```

Ques : Sqrt(x)

[Leetcode 69]

$x = 20 \quad \rightarrow ans = 4$



$mid^2 = x$ ✓

$mid^2 > x \rightarrow hi = mid - 1$

$mid^2 < x \rightarrow lo = mid + 1$