peak

⇒ any one of the peak

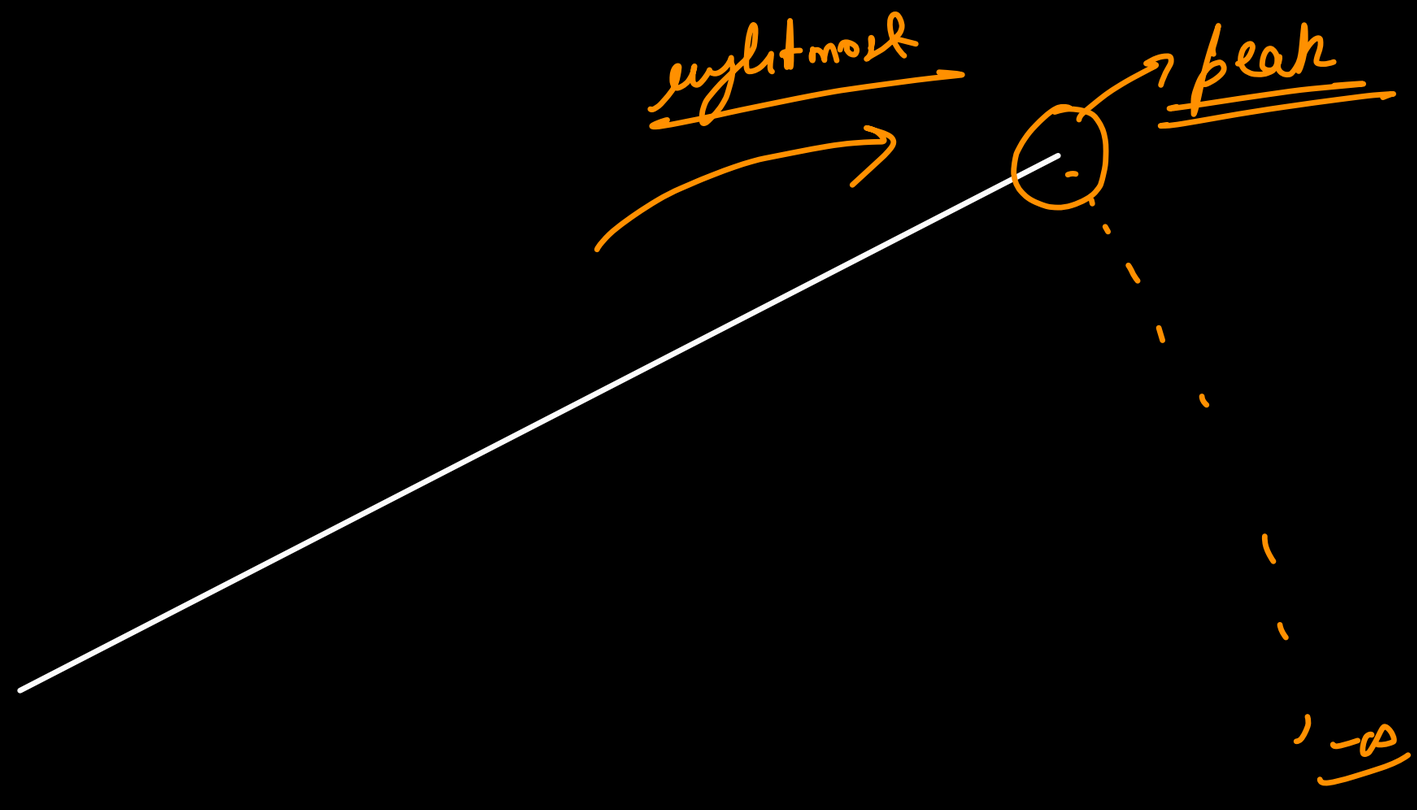
$a[i-1] < a[i] > a[i+1]$

peak

the element at i^{th} index will be the peak element if it is greater than the neighbors

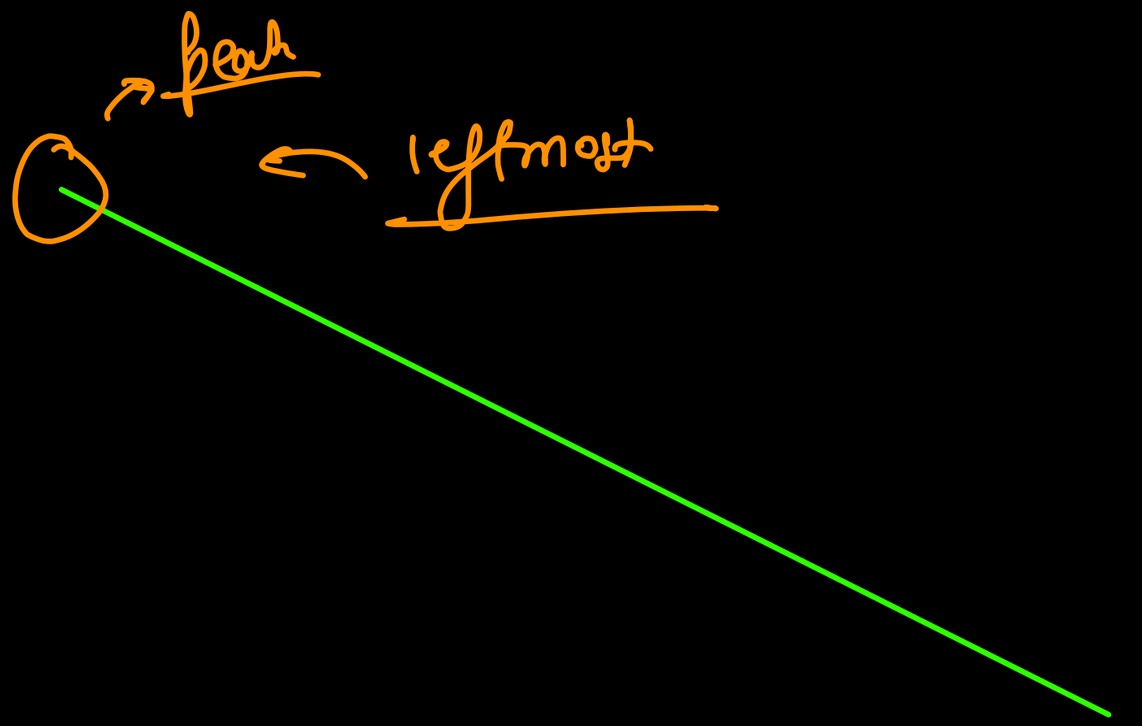
Case I →

whole array is sorted in inc order



Case II →

whole array is sorted in dec order



if we are on an inc curve, we will find ans on the right side

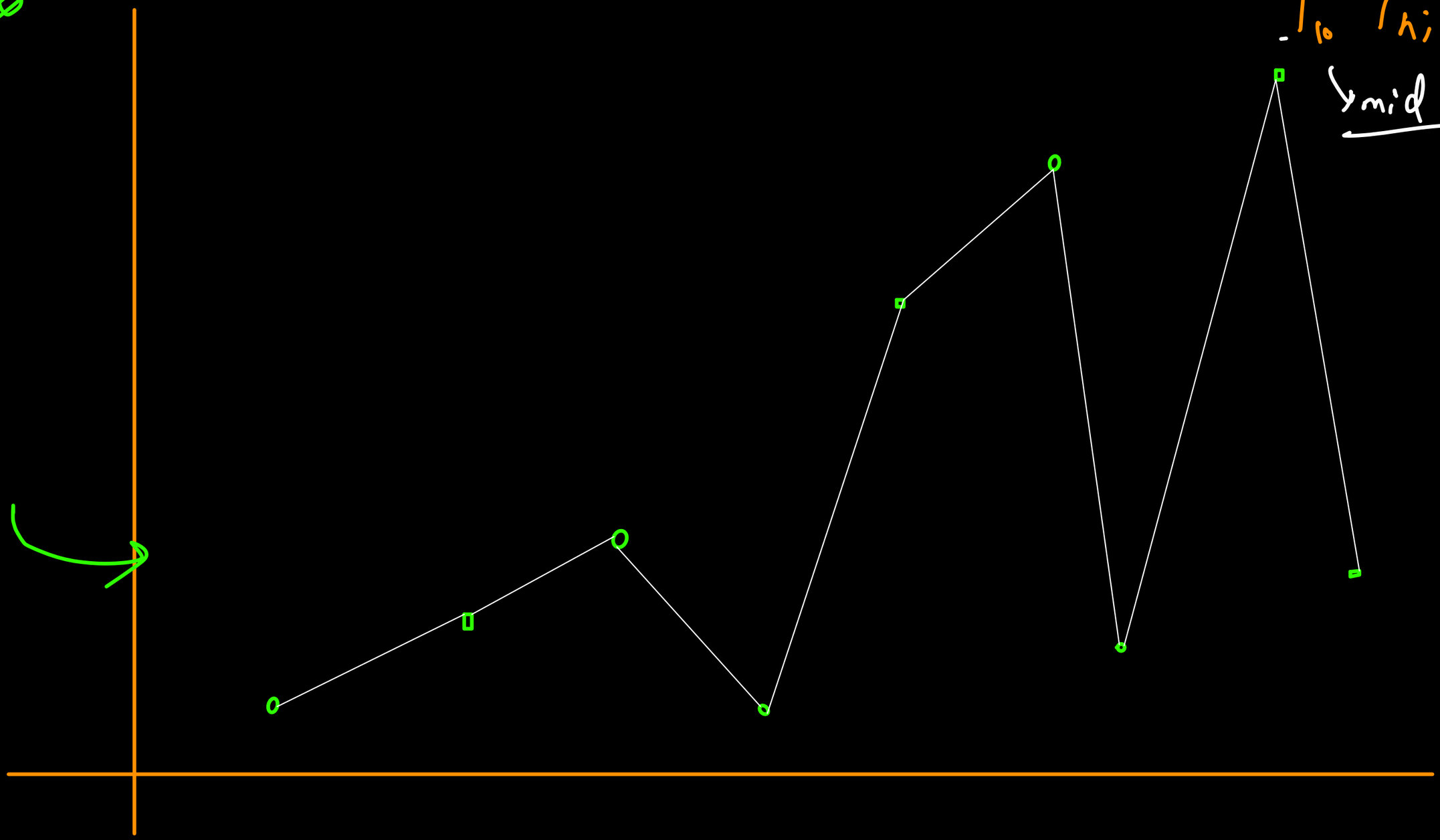
else if we are on a dec curve, we will find ans on the left side

0	1	2	3	4	5	6	7	8
1	2	3	1	6	8	2	9	3

$\log N$

lo hi

mid



```
while (lo <= hi) {
    if ( a[mid] > a[mid+1]  and  a[mid] > a[mid-1])
        return mid
    if ( a[mid] > a[mid+1]) {
        hi = mid - 1
    } else {
        lo = mid + 1
    }
}
```

Time → O(log n)

Space → O(1)

O(log n)

```
while (lo <= hi)
    mid = lo + (hi - lo)/2;
    if ( mid+1 >= n  and  a[mid] > a[mid-1])
        return mid;
    if (mid - 1 < 0  and  a[mid] > a[mid+1])
        return mid
    if ( a[mid] > a[mid+1]  and  a[mid] > a[mid-1])
        return mid
    if ( a[mid] > a[mid+1]) {
        hi = mid-1
    } else {
        lo = mid+1
    }
}
```

Q: Given a sorted (asc) array where every element is present twice only one element is present once. Find the single element in O(log N) time.

[1, 1, 3, 3, 8, 8, 10, 11, 11, 13, 13]

ans → 6 (index)

asc ← [1, 1, 2, 3, 3, 4, 7, 0, 8]
mid / high
0 1 2 3 4 5 6 7 8

first-occ → even
2nd occ → odd

Single elem

→ first occ → odd
2nd occ → even

How to distinguish if curr elem is 1st or 2nd index

i-1 ≥ 0 and a[i] == a[i-1]
→ 2nd occ

```
while ( lo < hi)
    mid = lo + (hi-lo)/2
    if (a[mid] != a[mid-1] and a[mid] != a[mid+1])
        return mid;
    if (a[mid] == a[mid-1]) → mid 2nd occ
        if( mid % 2 == 0)
            hi = mid-2;
        else
            lo = mid+1
    else { → 1st occ
        if (mid % 2 == 0)
            lo = mid+2
        else
            hi = mid-1
    }
```

O(log N)

O(1)

Binary Search On Ans

→ what is the quality we want to calc

Search Space

→ array of element

→ set

→ range of no.

2 ↓

[min - max] possible value of the quality

Search space

Square Root

$x \rightarrow 100$

$\sqrt{x}$

Square root of a number x, will always lie in the range $[1, x-1]$ (search space)

$x = 100$

$i = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10$

$ans = 1\ 1\ 1\ 2\ 2\ 2\ 2\ 2\ 3\ 3$

(1) Linear Search

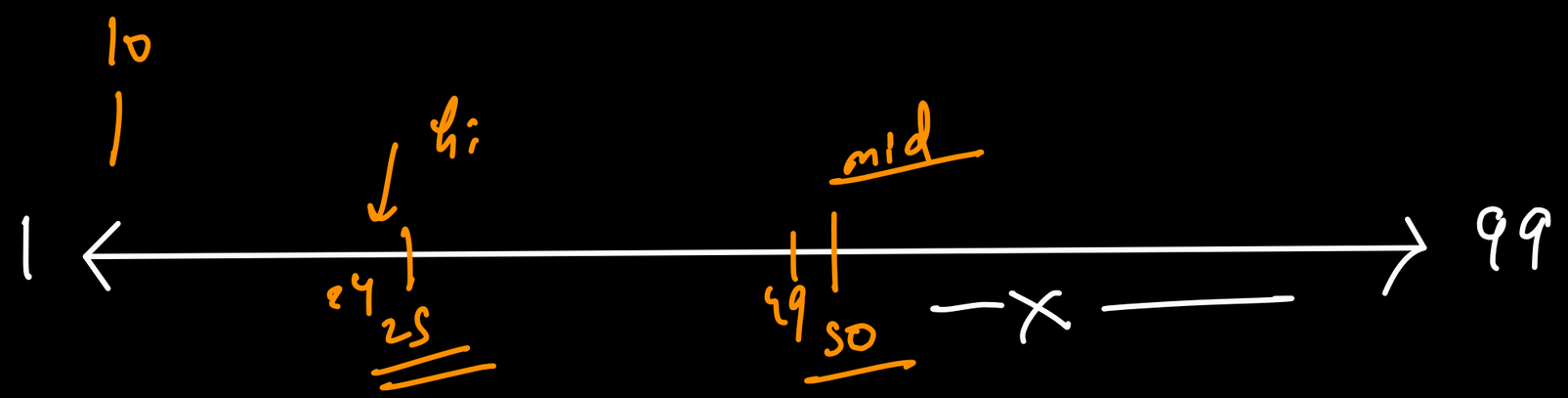
```
ans = -1
for (i=1; i<=x-1; i++)
    if (i*i <= x)
        ans = i;
    else
        break;
}
```

$O(\sqrt{x})$

first no. greater than root x we break the loop.

ans will be biggest value less than or equal to $\sqrt{x}$.

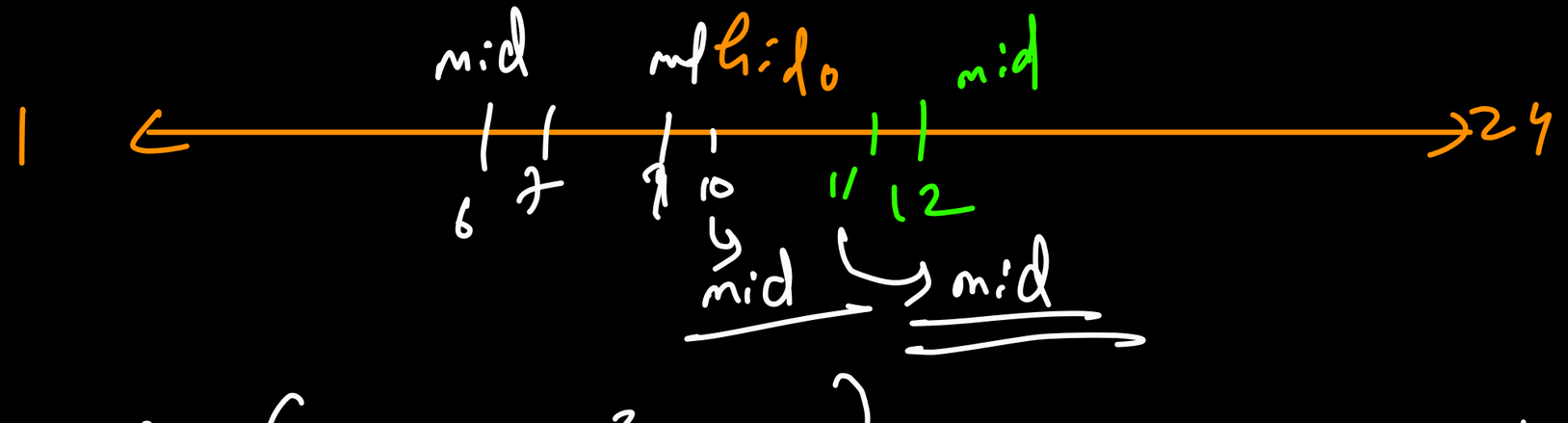
Binary Search



$mid^2 \leq x$

↓

mid can be a potential ans



if ($mid^2 > x$)
↳ discard right
hi = mid - 1

ans = ~~8~~ 9 10

else {
↳ discard left
lo = mid + 1
ans = mid
}

$O(\log x)$

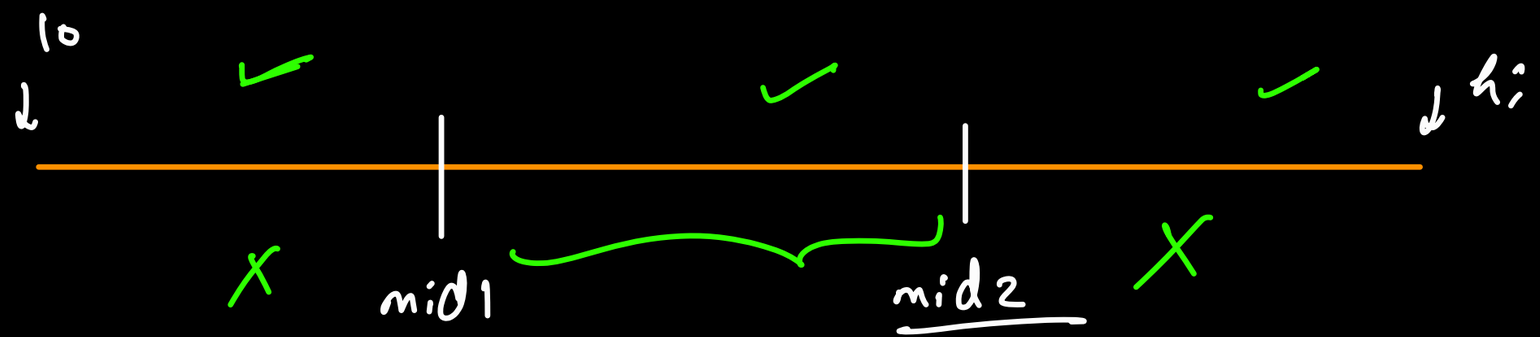
$O(1) \rightarrow$ space

$x \to 10^9$

$\sqrt{x} \to 10^5$

$\log x \to 30$

Ternary Search



$n \longrightarrow \frac{n}{3} \longrightarrow \log_3 n$

$mid1 = lo + \frac{(hi - lo)}{3}$

$mid2 = hi - \left(\frac{hi - lo}{3}\right)$

$n \longrightarrow \frac{n}{2} \hookrightarrow \log_2 n$

$\log_3 n < \log_2 n$


```
if( a[mid] > x) {

}
else {

}
```

```
if( x ≤ mid1)

else if (x2 ≥ mid2)

else {

}
```

$$n \longrightarrow \frac{n}{2} \longrightarrow \frac{n}{4} \longrightarrow \frac{n}{8} \cdots\cdots \frac{n}{2^k}$$

K elements

$$\frac{n}{2^k} = 1$$

$$k = \log_2 n$$

Total iterations of B.S

Total Comp $\rightarrow$ $\log_2 n$

$n \to \frac{n}{3} \to \frac{n}{9} \to \frac{n}{27} \cdots\cdots \frac{n}{3^m}$ — m iterations

$\frac{n}{3^m} = 1 \to n = 3^m$ Taking $\log_3$ both sides

$\log_3 n = m$

Total iterations

Total comp in Nervaj Seoul = $2 \log_3 n$

$\log_2 n$ $\qquad$ $2\log_3 n$

you can convert the bases

$2\log_3 n \rightarrow \dfrac{2\log_2 n}{\log_2 3}$ $\qquad$ $\log_2 3 \rightarrow 1.585$

$\log_2 n$ $\qquad$ $\dfrac{2}{\log_2 3} \times \log_2 n$

$\downarrow$ $\qquad$ $\angle$

$1 \times \log_2 n$ $\qquad$ $1.26 \times \log_2 n$

Hence, Binary Search is always better than ternary Search.

for big n.

x → 4321

ans → 4321.

ans = (ans * 10 + last digit)