BINARY SEARCH 2

Maximum element in a sorted and rotated array

- Given a sorted array **arr[]** of distinct elements which is rotated at some unknown point, the task is to find the maximum element in it.

- Examples:
- Input: arr[] = {3, 4, 5, 1, 2}
- Output: 5

- Input: arr[] = {1, 2, 3}
- Output: 3

Approach

- A simple solution is to traverse the complete array and find maximum. This solution requires O(n) time.
  We can do it in O(Logn) using Binary Search. If we take a closer look at above examples, we can easily figure out the following pattern:

- The maximum element is the only element whose next is smaller than it. If there is no next smaller element, then there is no rotation (last element is the maximum). We check this condition for middle element by comparing it with elements at **mid – 1** and **mid + 1**.
- If maximum element is not at middle (neither mid nor mid + 1), then maximum element lies in either left half or right half.
  - If middle element is greater than the last element, then the maximum element lies in the left half.
  - Else maximum element lies in the right half.

Advanced Implementation

```
int last_true(int lo, int hi, function<bool(int)> f) {
  lo--;
  for (int dif = hi - lo; dif > AllowedError; dif /= 2) {
                }
  return lo;
}
```

Complexity Analysis

- Binary search divides the problem size by half in each step, leading to a logarithmic growth rate.
- Time Complexity: O(log n)
- Space Complexity: O(1) (constant space, as we're not using any additional data structures)

Advantages and Disadvantages

- **Advantages:**
- Efficient for large datasets.
- Optimal for situations where data doesn't change frequently.
- Ideal for situations with limited memory.
- **Disadvantages:**
- Requires a sorted dataset.
- Inefficient for unsorted data (preprocessing is needed).
- Insertions and deletions are more complex and costly than in linear data structures.

Applications

- Binary search is widely used in various applications:
  - Searching in databases and libraries.
  - Finding elements in games (e.g., guessing games).
  - Mathematical calculations (e.g., finding square roots).
  - Network routing algorithms.

BS

TS

Interactive

ABS

BS

70%

20%

80%

BS

result → monotonic

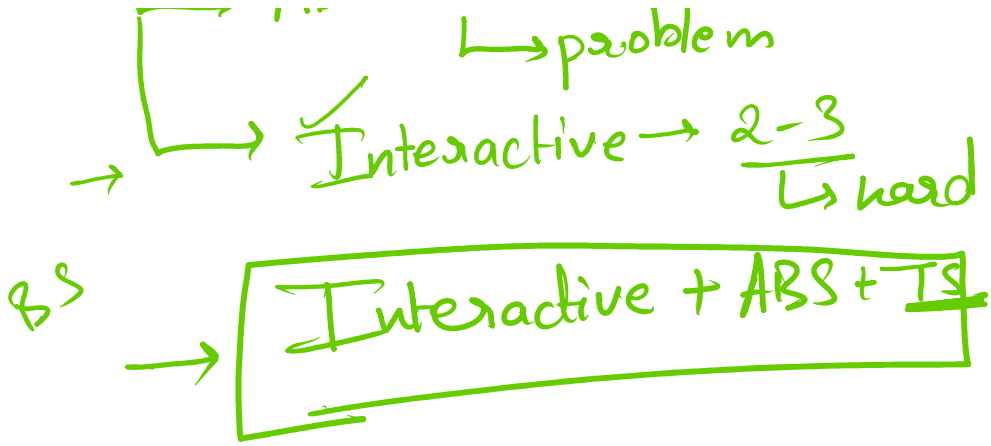
BS

monotic →

ABS → square root → if(mid*mid≤n)

(cube root) → if(mid*mid*mid ≤ n)

ABS + TS → heavy → problem



BS

smallest number x
for $\lfloor x \rfloor$, x!
is having n
zeroes

$n \leq 10^4$

```
while (low ≤ high)
{
    mid = (low + high)/2
    if (func(mid))
        ans=mid, high=mid-1
    else
        low = mid+1
}
```

}

func {

5

(3) 10 → 2

n

1 · 2 · 3 · 4 · (5) · 6 · 7 · 8

(5) × 5

zeros = n / (5)

25

n/25

n/25

625

n >

zeros >

$n! = 1 \times 2 \times 3 \cdots n$

2 (5)

(10)

zeros >

zeros = n/5

(5)(5)

n/25

$5^3 \times$

5^4 n/(25)

5^3 x n/125 5^4 n/625

func → monotone

Yes

n n

(n+1)

n! x (n+1)

n

5n

5n → n

5n

5n

25

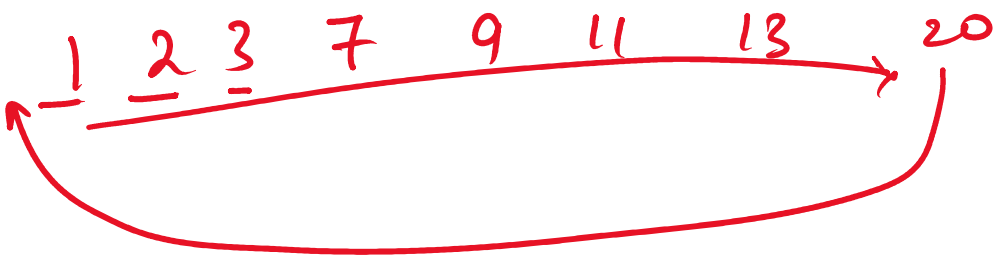
125

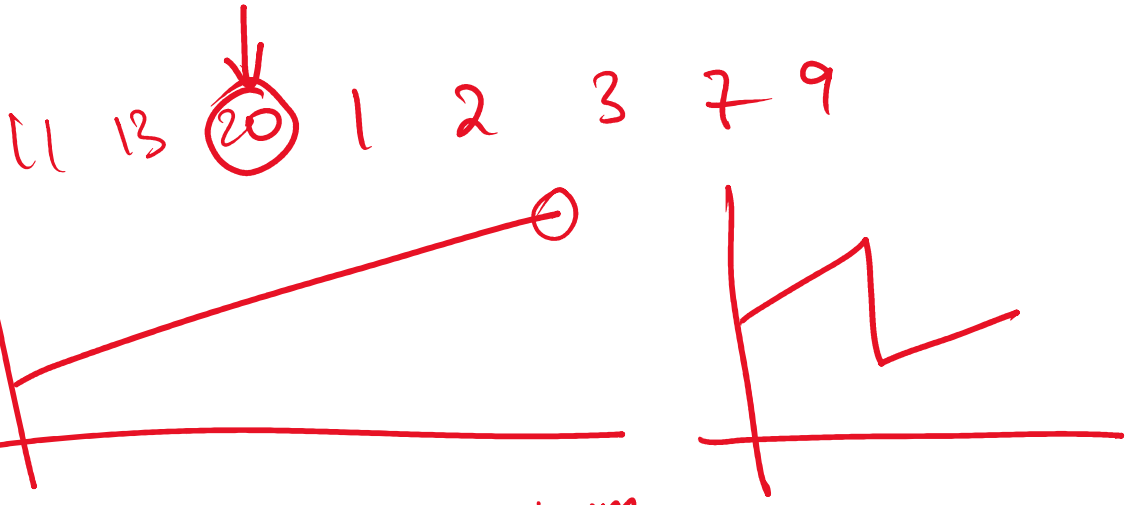
625

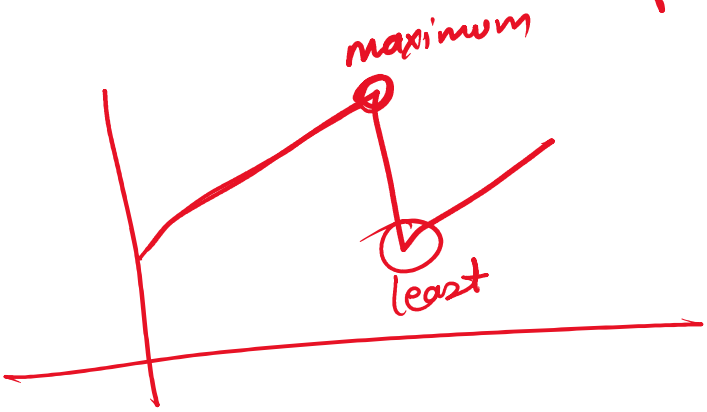
Rotated Sorted Array

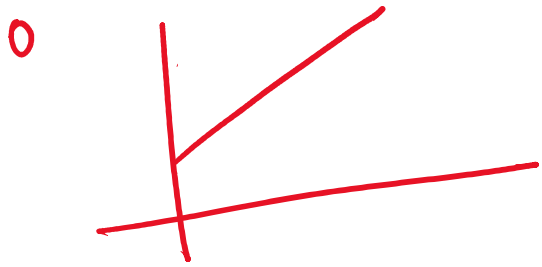
{

→ position of maximum element


1 2 3 7 9 11 13 20

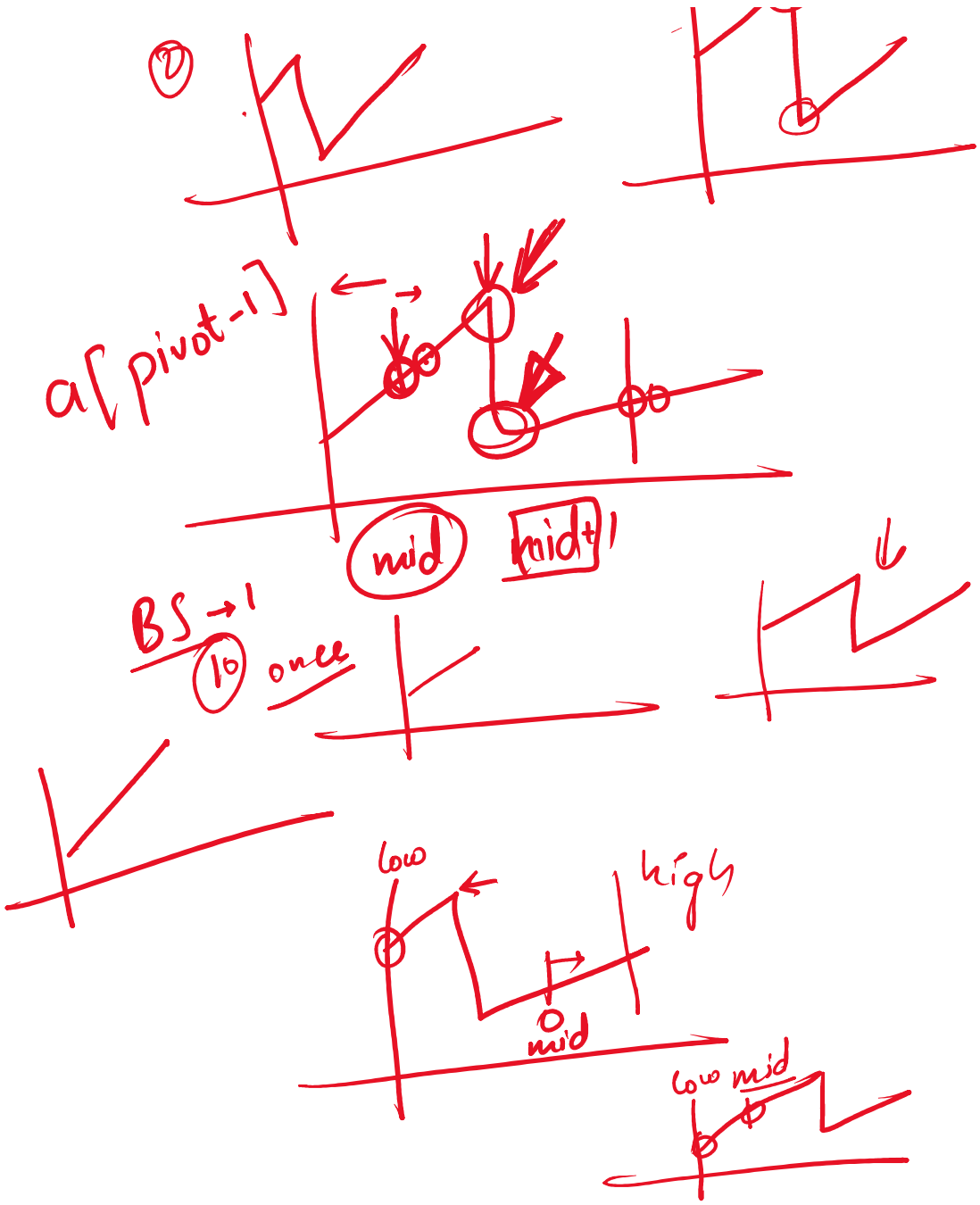

11 13 20 1 2 3 7 9

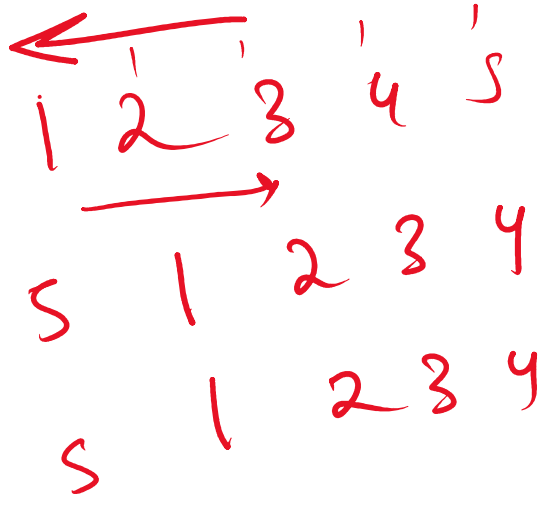

maximum
least


0

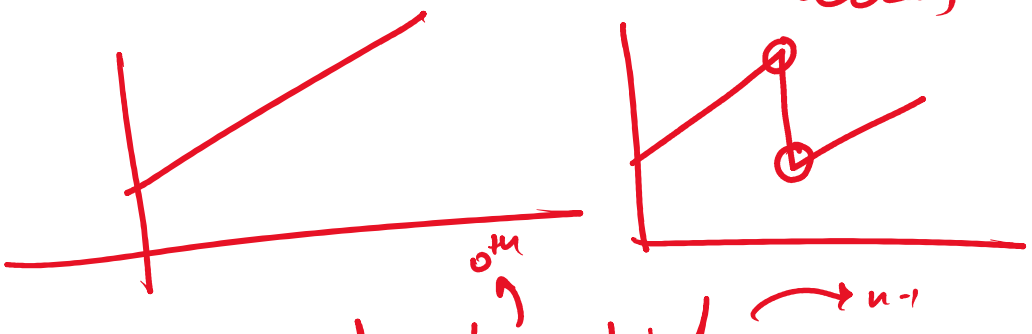

0


11
19

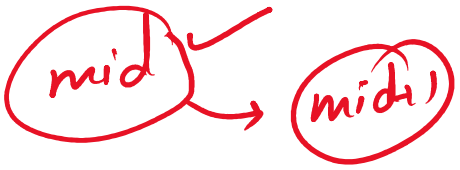

a[pivot-1]
mid
mid+1
BS → 1
low
high
mid
low mid


1 2 3 4 5
5 1 2 3 4
5 1 2 3 4

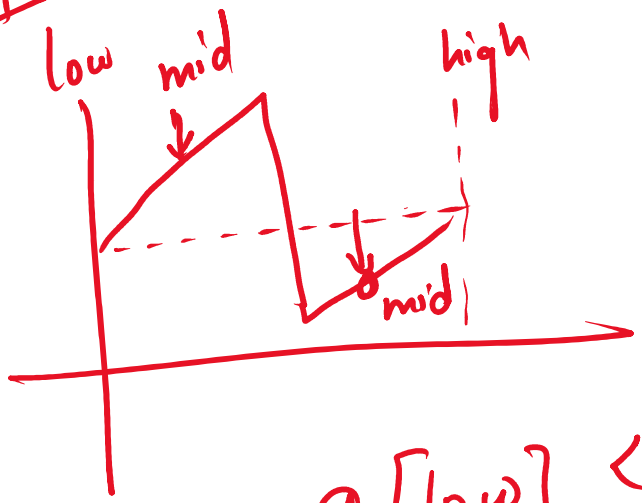
Given → Roted sorted array



$$mid = \frac{low + high}{2}$$



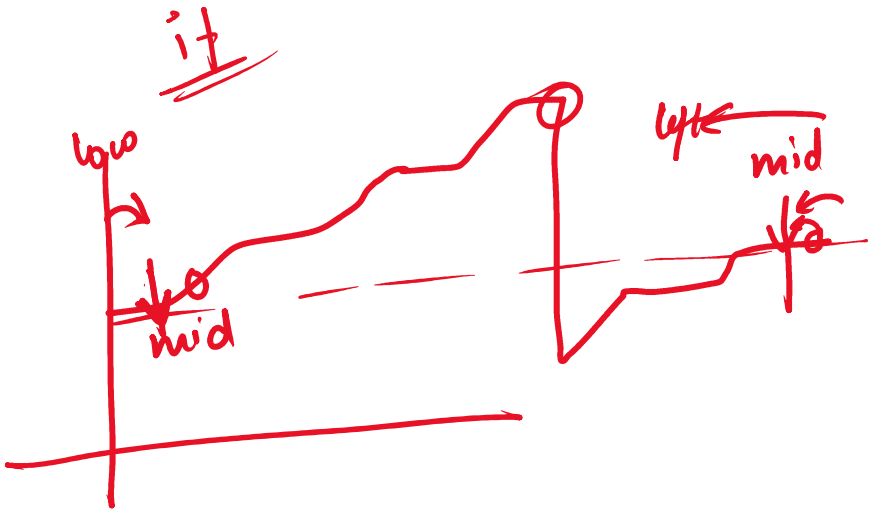
array



a[low] < a[mid] right range

a[low] > a[mid] left range

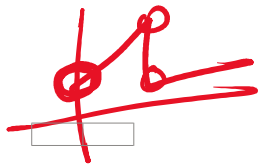
have monotone



idea

↳ monotone

↳ check for some special condition → result



Advanced BS

decimals

error

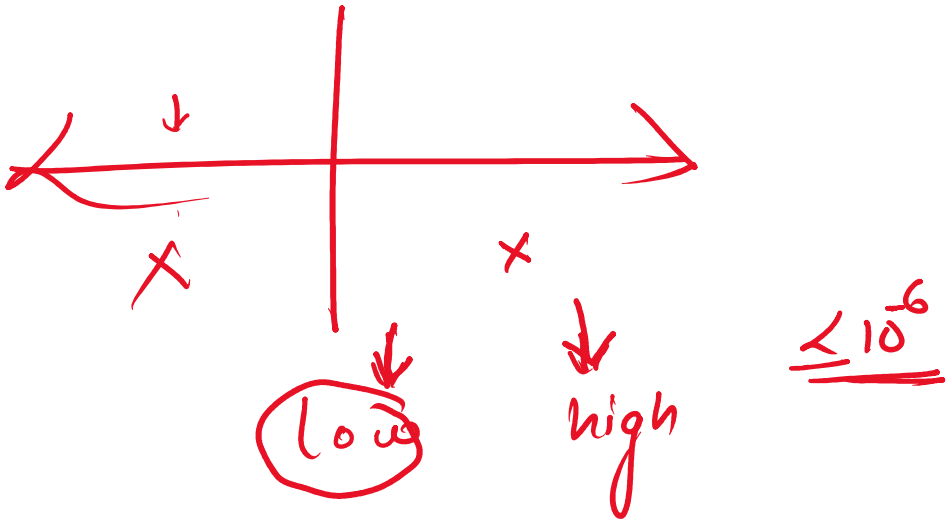
integers

8 → 4 → 2 → 1 → 0

8 → 4 → 2 → 1 → 0.5 → 0.25 → 0.125

low high

error



$\leq 10^{-6}$

Square root

BS

Sqrt

10^{-6}

2 - 3

→ SquareRoot ( )

→ Sqrt → 10^9

Sqrtl 10^{18}

Square root

(n) → decimal

high = n

decimal low = 1 high = n

while (high - low ≥ 10^{-6})

{

mid

mid x mid ≤ n

ans = mid , low = mid + 10^{-6}

este high = mid - 10^{-6}

low = high

low ↔ high

3

low

mid

10^{-6}

10^{-6}

10^{-6}

0.5 >>> 10^{-6}

1

mid

ans

$1+10^{-6}$

1.5

low = 2

10^{-6}

h

1 x 1 < 2.25

1.5 ≤ 2.25

2.02 > 2.5

ABS

→ Decimals

→ 10^{-9}

low = mid + 10^{-9}

high - low ≥ 10^{-9}

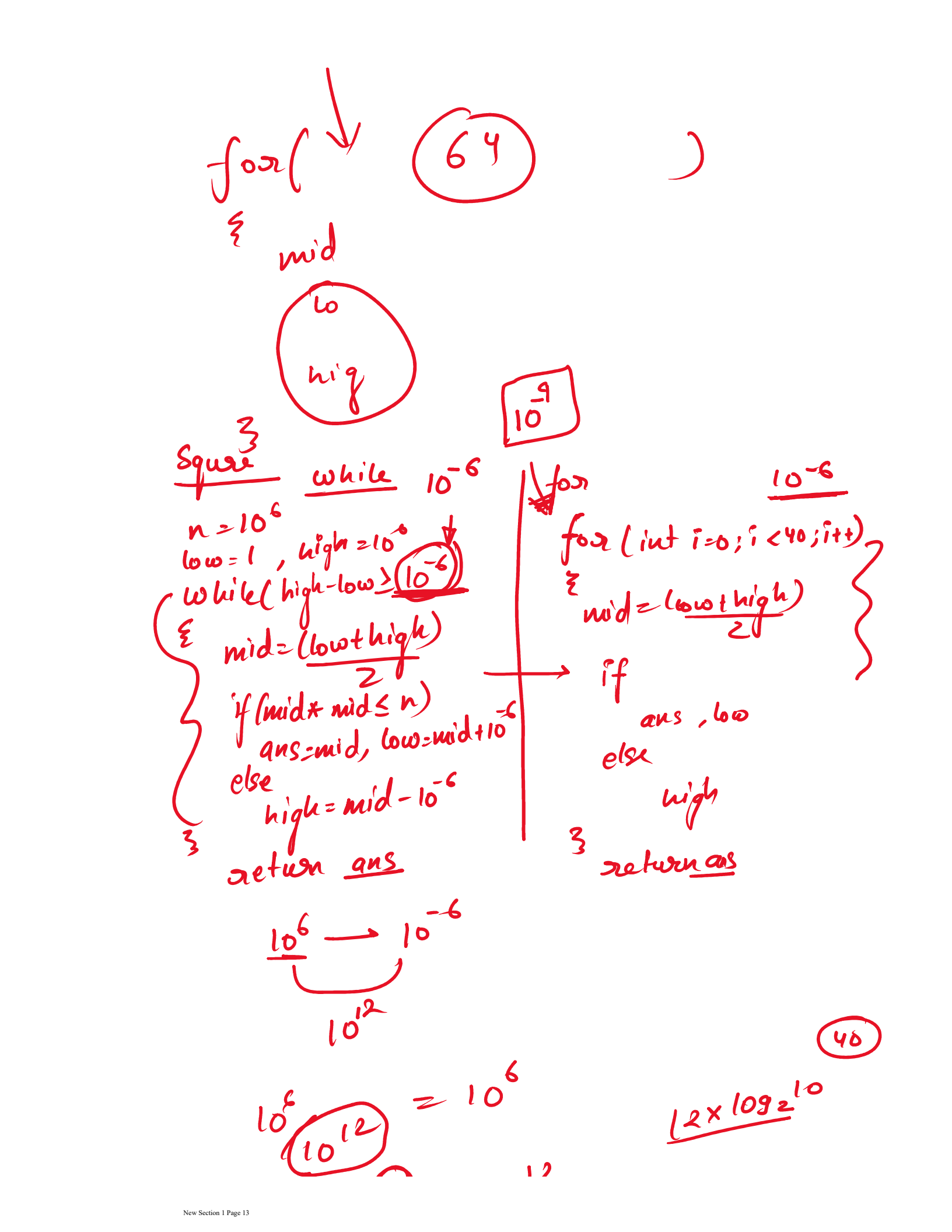
1.00000001

while (high - low ≥ error → 10^{-9})

10^9 → 10^{-9}

10^{18} ← ②

64 ← 2^{64}


for(
64
)
{
mid
lo
hig
}
Squre
while
10^{-6}
10^{-9}
for
10^{-6}
n = 10^6
low = 1 , high = 10^6
while(high-low ≥ 10^{-6})
{
mid = (low+high)/2
if(mid*mid ≤ n)
ans = mid, low = mid + 10^{-6}
else
high = mid - 10^{-6}
}
return ans
for(int i=0; i<40; i++)
{
mid = (low+high)/2
if
ans , low
else
high
}
return ans
10^6 → 10^{-6}
10^{12}
40
10^6 = 10^6
10^{12}
$12 \times \log_2 10$

10^{12}

$2^x = 10^{12}$

$x \log 2 = 12$

$x =$ $\log_2 10^{12}$

---

1 1 0 1 0 0 1 0 1 1

L R

② 3rd

? L, R

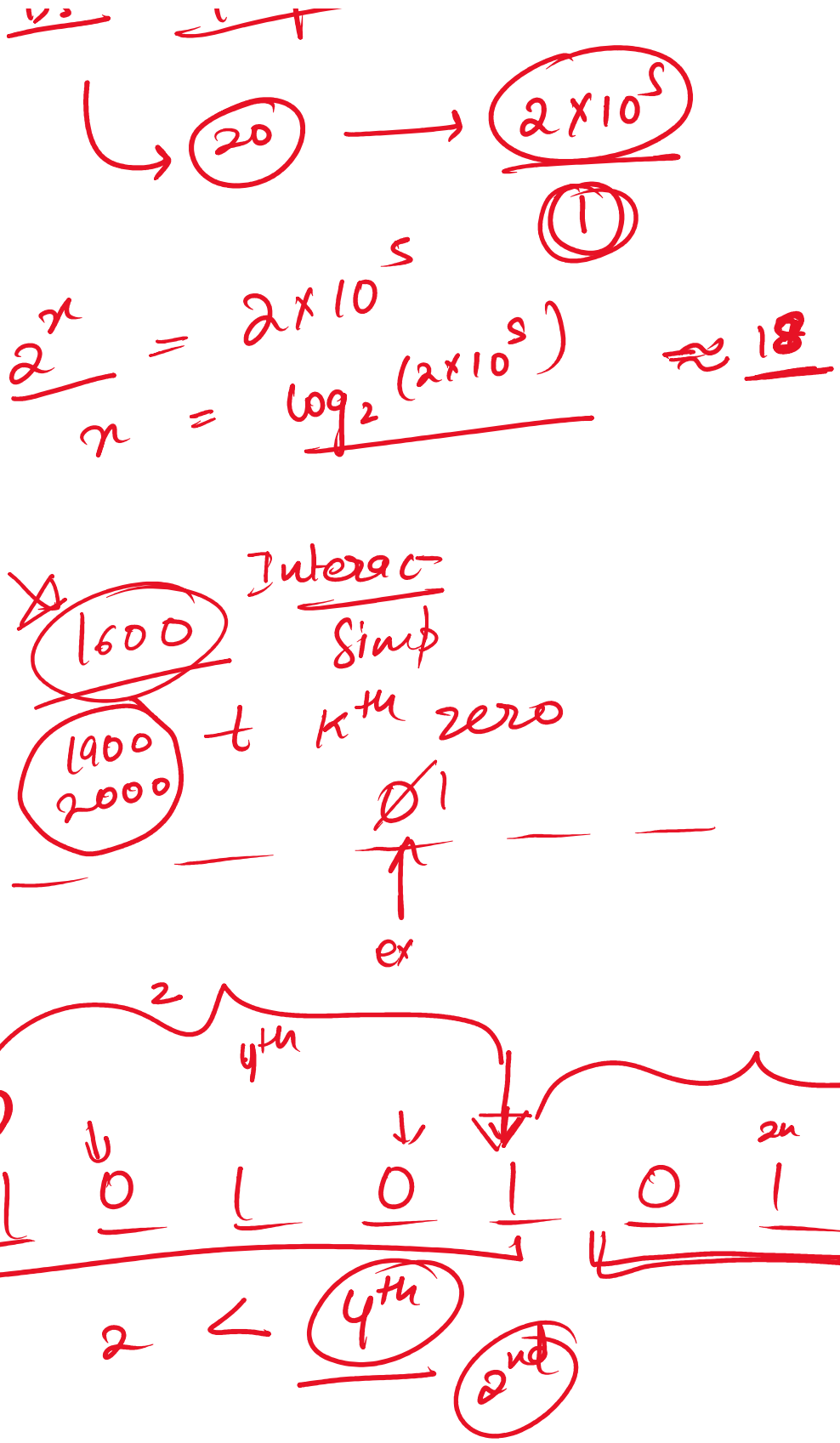
$n < 2 \times 10^5$

20

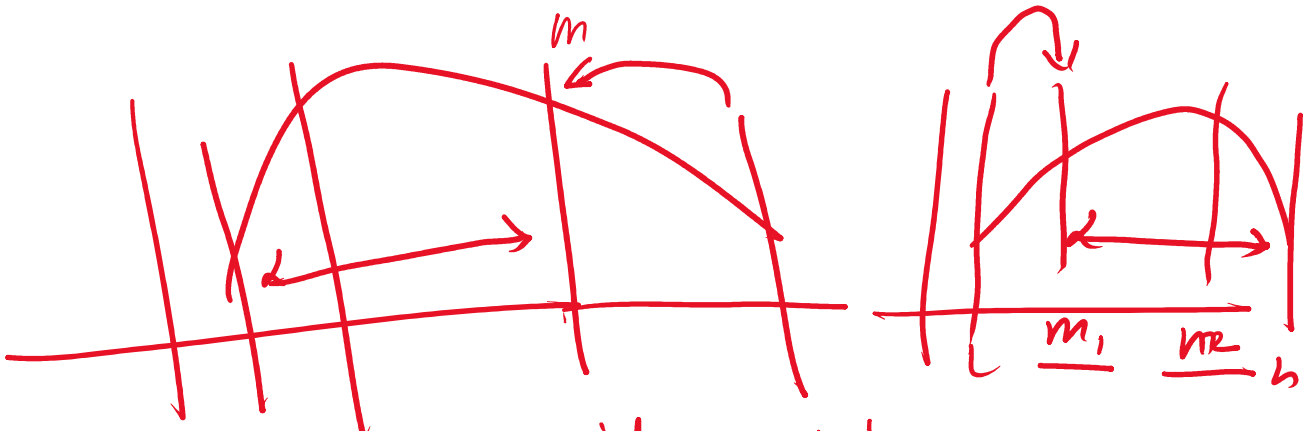
Prefix sum

L – Sum (0-1) = # zeros (0-L)

bs → prefix sum

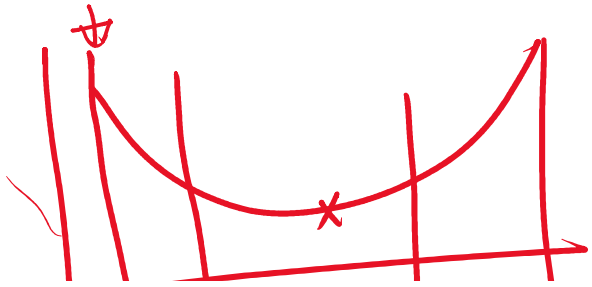

20
2×10^5
1
$2^x = 2 \times 10^5$
$x = \log_2(2 \times 10^5) \approx 18$
1600
1900
2000
Interac
Simp
+ K^{th} zero
Ø 1
ex
log
2
4^{th}
1 0 1 0 1 0 1 0 0
2m
+
$2 < 4^{th}$
2^{nd}

T·S

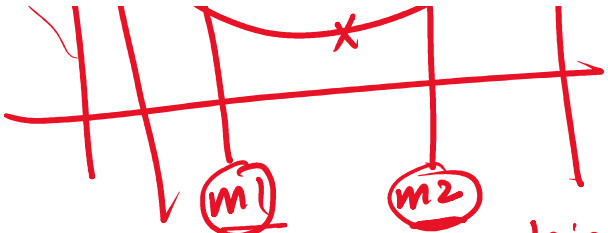


low mid1 mid2 high

if(a[mid] ≥ a[mid2])
high = mid2
else
low mid1



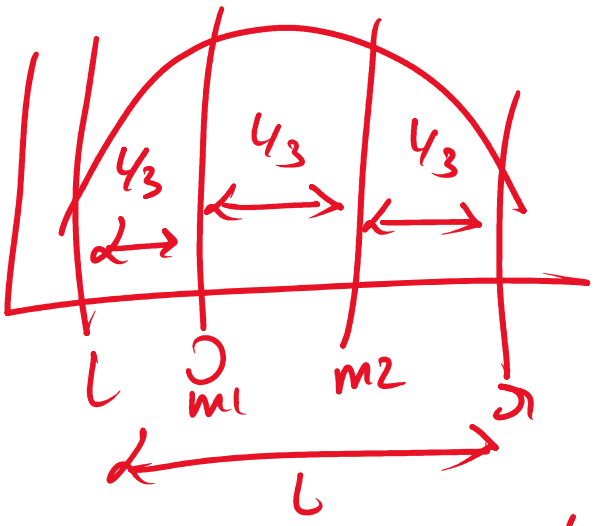
min → shift boundary to mose



min → swap to more value

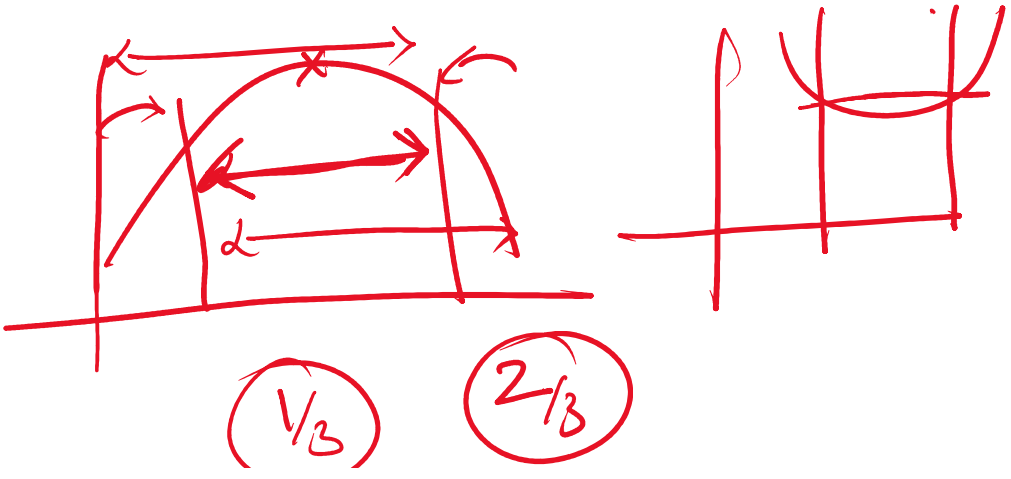
max → shift boundary to lower value

```
if(a[m1] >= a[m2])
    low = m1;
else
    high = m2;
```



eg

$$\begin{cases} m1 = l + \frac{(r-l)}{3} \\ m2 = r - \frac{(r-l)}{3} \end{cases}$$



(1/3) (2/3)

→ a_1 a_2 a_3 - - - a_n

↳ x = ?

min → max value → absolute value of the sum of seq

weakness → $a_1 - x$, $a_2 - x$ - - - - - $a_n - x$

0 ←——→ ∞

minimum weakness

max value → poorness

poorness → absolute value of the sum

8 (n) (n × n)

1 2 3 4

$x = 2.5$ | -1.5 -0.5 0.5 1.5 | max sum

-1.5 -0.5 → -2.0

0.5 1.5 → 2.0

max abs

max value of sum

-2.0 → 2.0

2.0 → 2.0

minimize → |sum|

max of

n -n -n -n -n -n

n^2 → Sub array → max Sum

2nd Qn to S → ME

$$\rightarrow Sum(i,j) = \sum_{i}^{n} a_i - x$$

→ sum(i,j) ←

$\max \sum_{i=0}^{n} \sum_{j=i}^{n} (|sum(i,j)|)$

minimized

$|a| = -a$ (if $a < 0$)

(n)

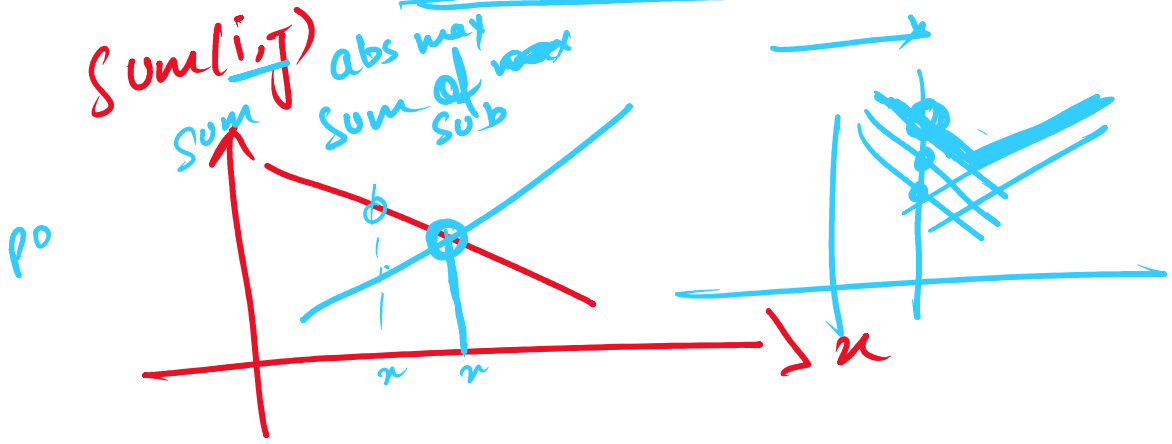
$\max \sum_{i=0}^{n} \sum_{j=i}^{n} \max(sum(i,j), -sum(i,j))$

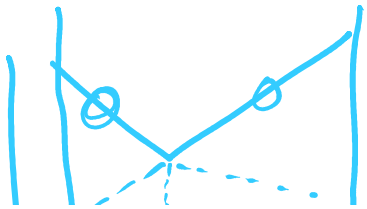
$sum(i,j) \rightarrow (A[i] + A[i+1] - - A[j]) - (j - i + 1) \times \Delta$

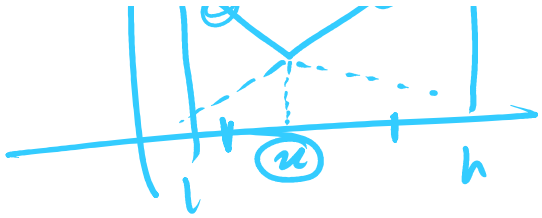
(c) − kn

$-sum(i,j)\,(j - i + 1)\,n - (A[i] + \{A[i+1] - - A[j])$

kn − (c)





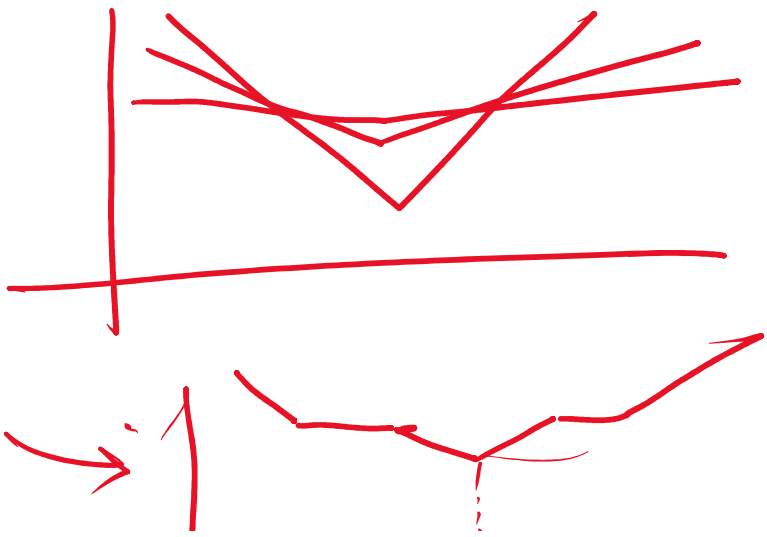


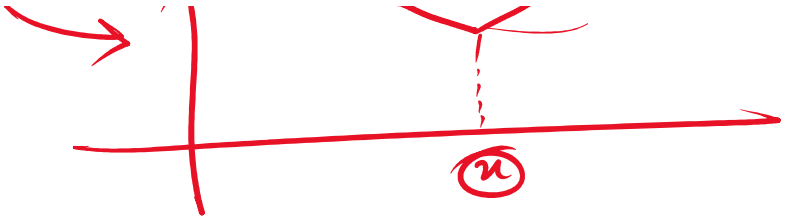
if(value[m1] ≤ value[m2])

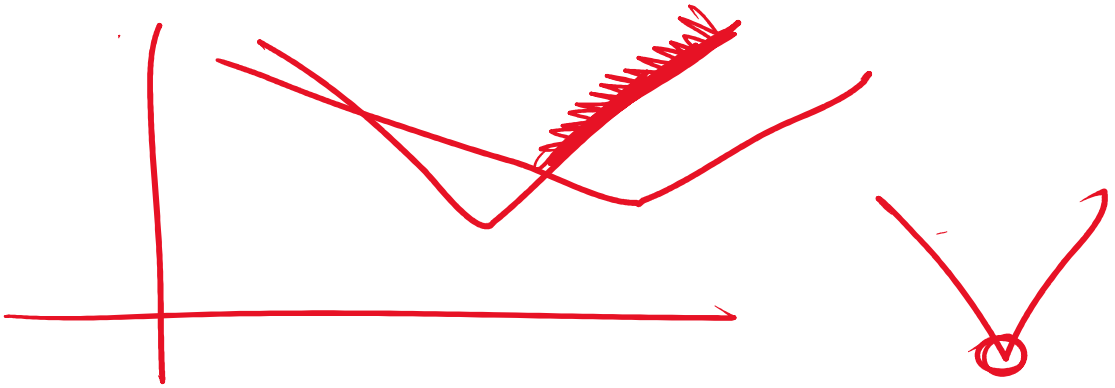
high = m2

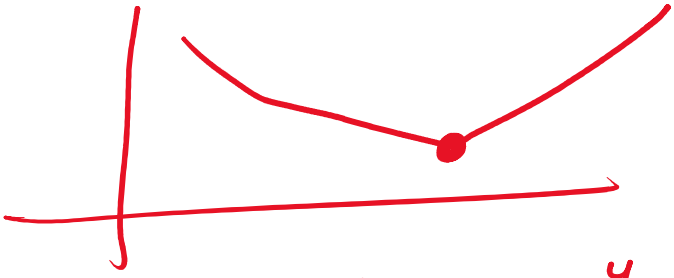
else

low = m1









range -10^4 – 10^4 WA

precision

→ while (high − low ≥ 10^{-6})

error → for → while for

→ always →

decimal → while

↳ WA

100

2^{100}

10^{-6}

low high

for (100)

{

m1

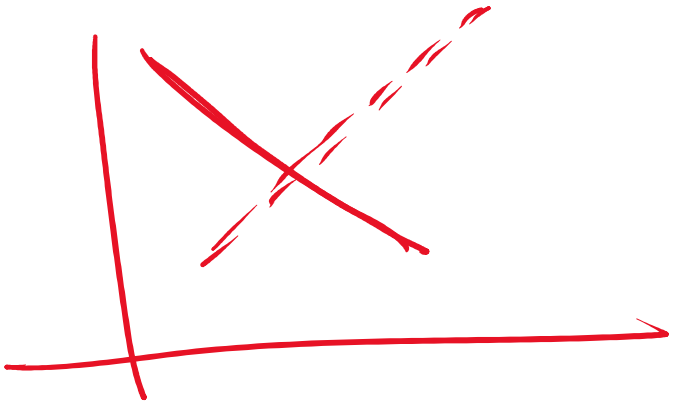
m2

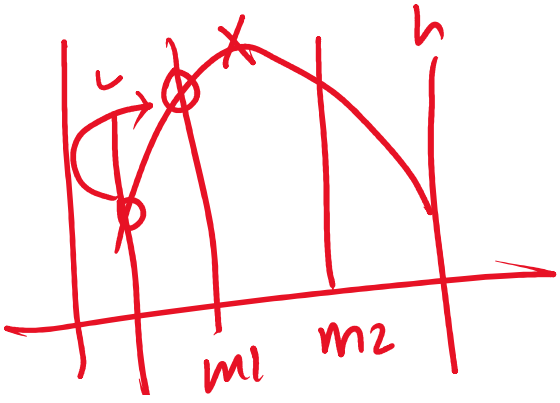
m1 > m2

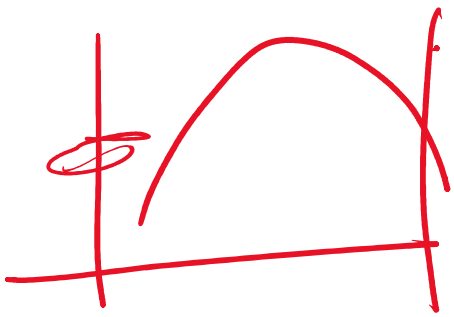
l − m1

else h − m2

}





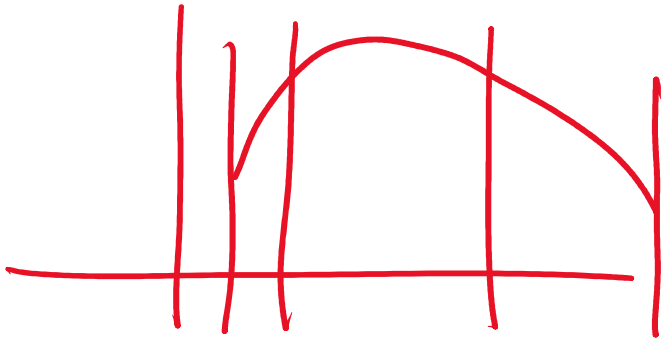


m1 m2

m1 < m2

l = m1

h = m2



value[m2] < value[m1]

high > m2

~~v[low] < v[m1]~~

max → parbola

val m1 < m2

min - para

m1 < m2