BINARY SEARCH

Introduction

- Binary search is an algorithmic technique used to locate a specific value within a sorted collection, typically an array.
- Unlike linear search, which has a time complexity of O(n) in the worst case, binary search boasts an impressive O(log n) time complexity.
- This makes it incredibly efficient for large datasets and is the reason why it's a cornerstone of algorithm design.

How Binary Search Works

1. Begin with a sorted array.
2. Compare the middle element of the array with the target value.
3. If the middle element is equal to the target, we've found it!
4. If the target is less than the middle element, eliminate the right half of the array and repeat step 2 on the left half.
5. If the target is greater than the middle element, eliminate the left half of the array and repeat step 2 on the right half.
6. Continue this process until the target is found or the search space becomes empty.

Finding The Maximum x Such That f(x) = true

- Let's say we have a boolean function f(x) . Usually, in such problems, we want to find the maximum or minimum value of x such that f(x) is true. Similarly to how binary search on an array only works on a sorted array, binary search on the answer only works if the answer function is monotonic , meaning that it is always non-decreasing or always non-increasing.

- We want to construct a function lastTrue such that lastTrue(lo, hi, f)
- returns the last x in the range [lo,hi] such that f(x) = true. If no such
- x exists, then lastTrue should return lo-1.
- This can be done with binary search if f(x) satisfies both of the following
- conditions:
  - If f(x) = true, then f(y) = true for all y <= x.
  - If f(x) = false, then f(y) = false for all y >= x.

Implementation

```
int last_true(int lo, int hi, function<bool(int)> f) {
  // if none of the values in the range work, return lo - 1
  lo--;
  while (lo < hi) {
        // find the middle of the current range (rounding up)
        int mid = lo + (hi - lo + 1) / 2;
        if (f(mid)) {
              // if mid works, then all numbers smaller than mid also work
              lo = mid;
        } else {
              // if mid does not work, greater values would not work either
              hi = mid - 1;
        }
  }
  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

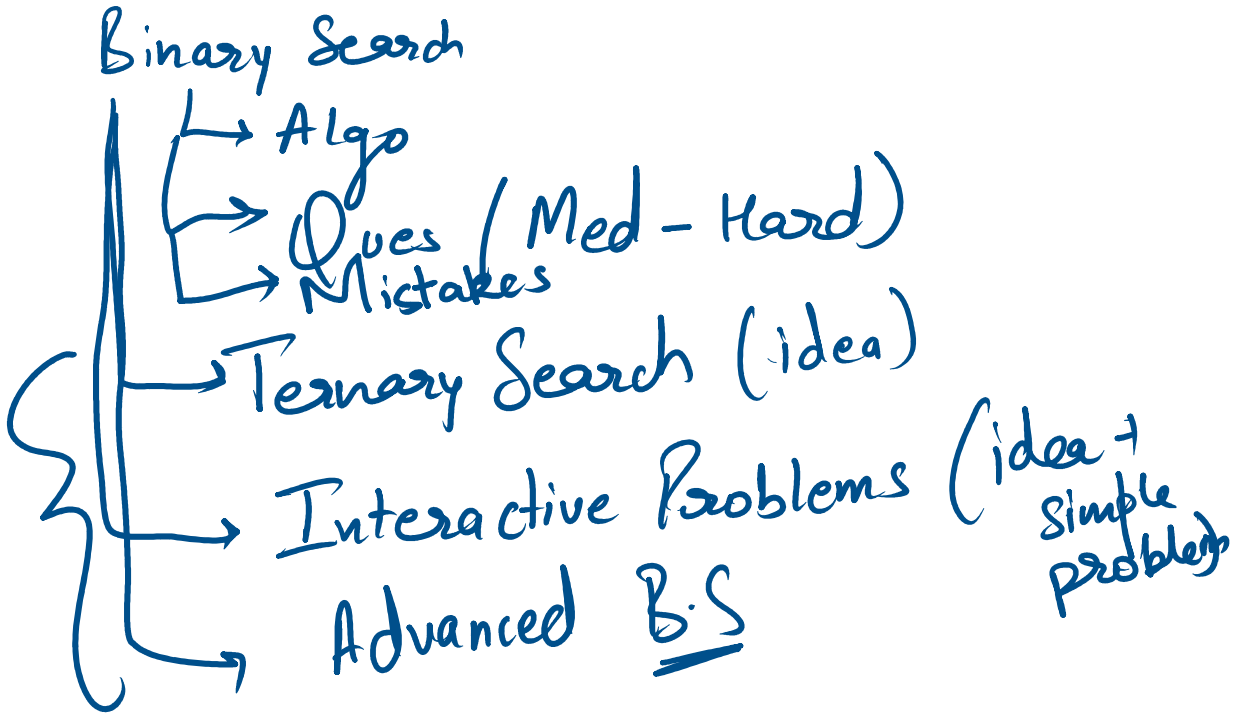
- **Advantages:**
- Efficient for large datasets.
- Optimal for situations where data doesn't change frequently.
- Ideal for situations with limited memory.
- **Disadvantages:**
- Requires a sorted dataset.
- Inefficient for unsorted data (preprocessing is needed).
- Insertions and deletions are more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

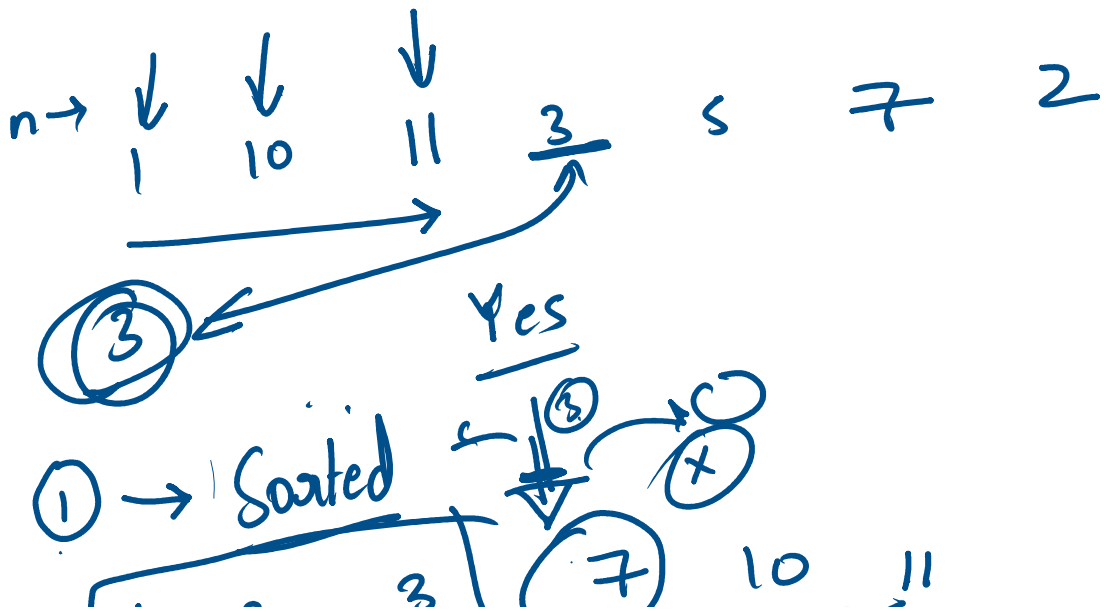
Applications

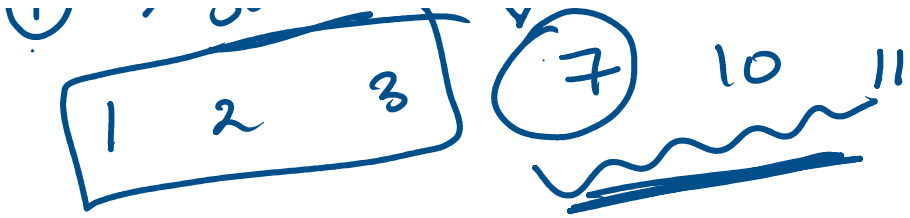
- Binary search is widely used in various applications:
  - Searching in databases and libraries.
  - Finding elements in games (e.g., guessing games).
  - Mathematical calculations (e.g., finding square roots).
  - Network routing algorithms.

Advanced Implementation

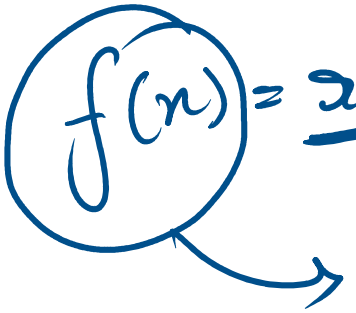
```
int last_true(int lo, int hi, function<bool(int)> f) {
  lo--;
  for (int dif = hi - lo; dif > AllowedError; dif /= 2) {
                }
  return lo;
}
```


Binary Search
Algo
Ques (Med - Hard)
Mistakes
Ternary Search (idea)
Interactive Problems (idea + simple problem)
Advanced B.S

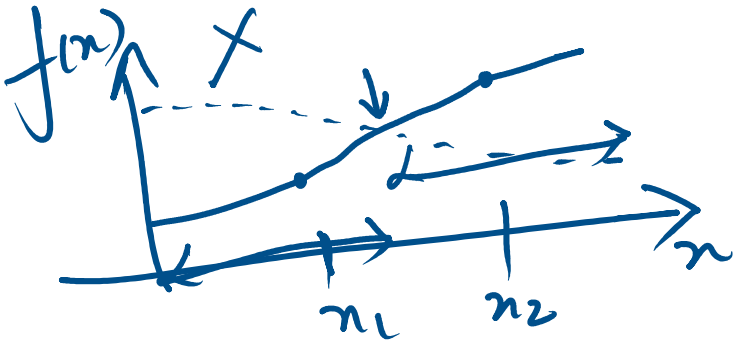

n
1
10
11
3
Yes
Sorted
7
10
11



Binary Search
↳ ans



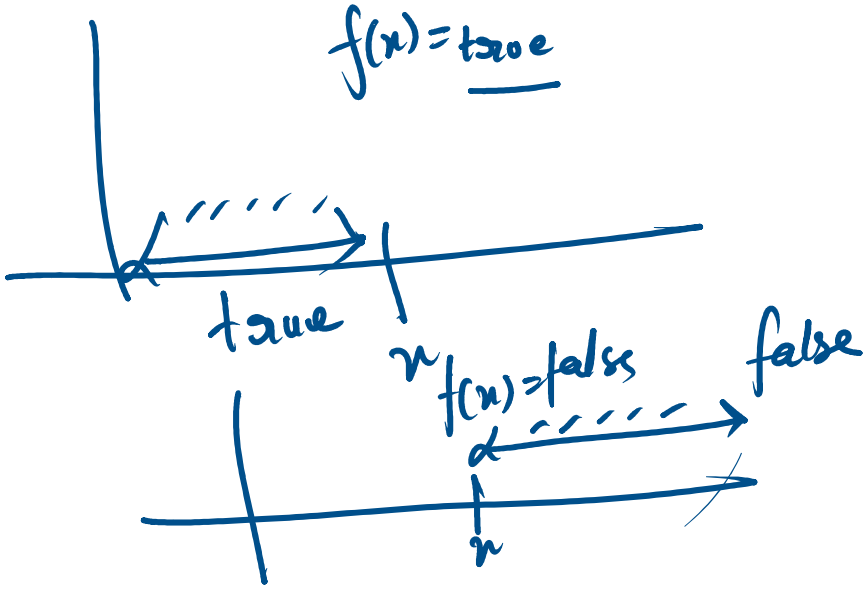
should be monotonus

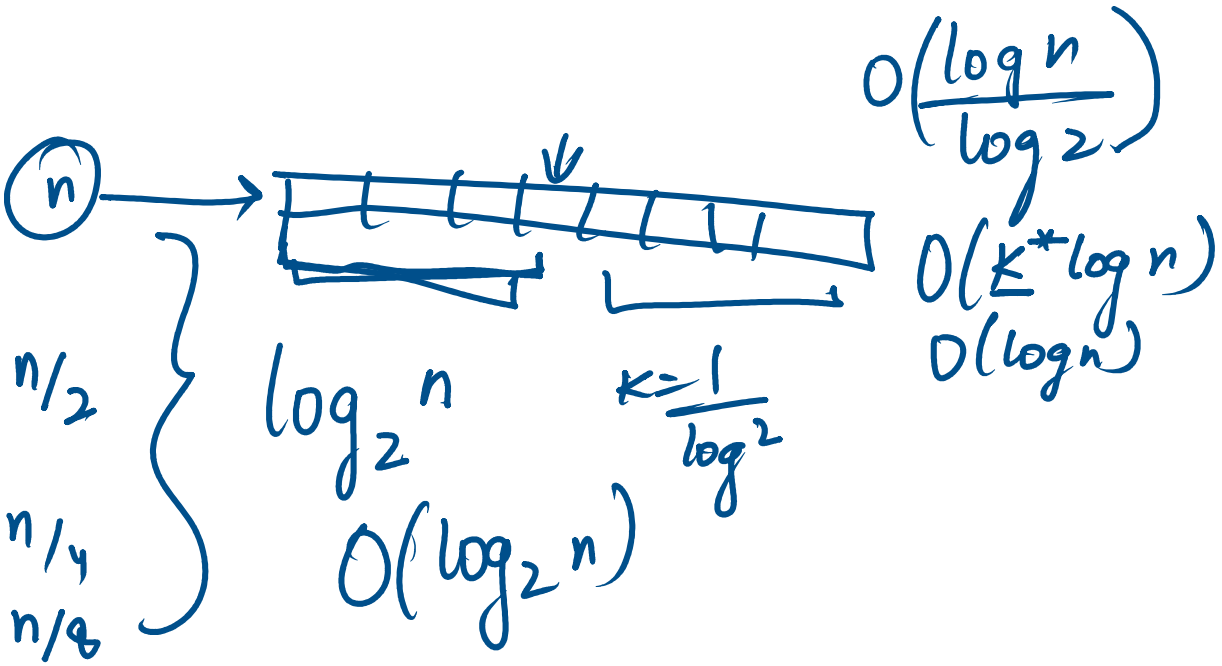


$n_1 < n_2$,
$f(n_1)$, $f(n_2)$

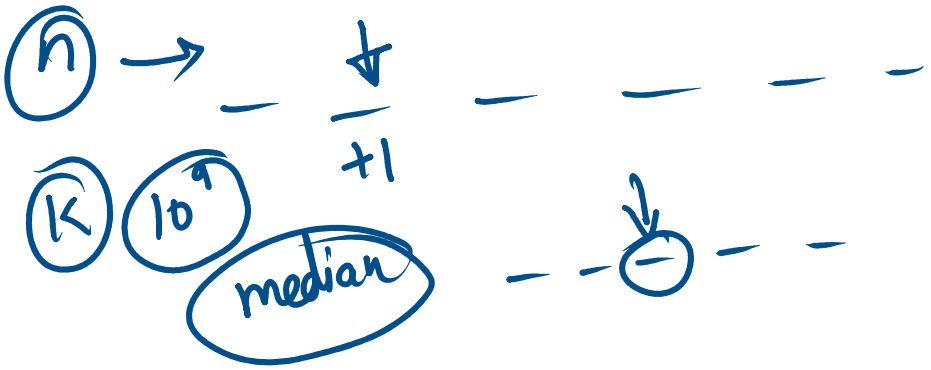
$x_1 < x_2$

$f(x_1) < f(x_2)$





n 2×10^5



maximize median

median

maximize

ans → median

low = high

mid

a[mid] == n

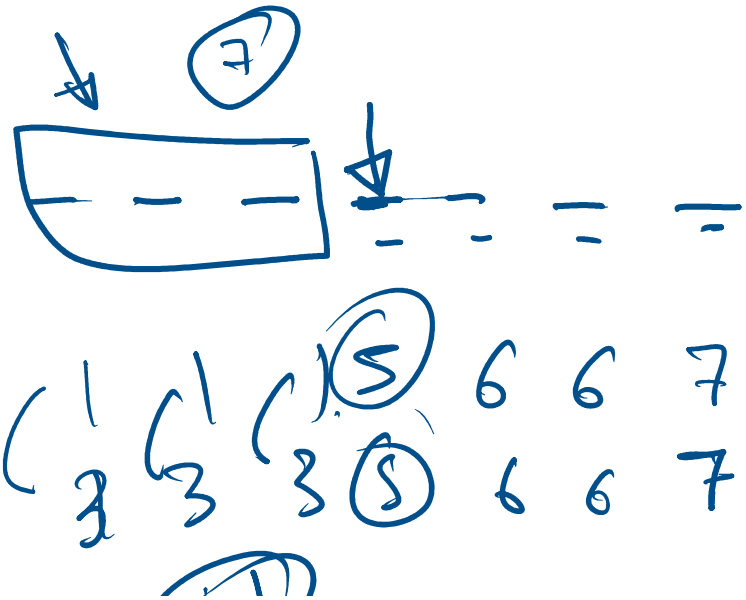
f(mid) → whether or not mid is possible as median

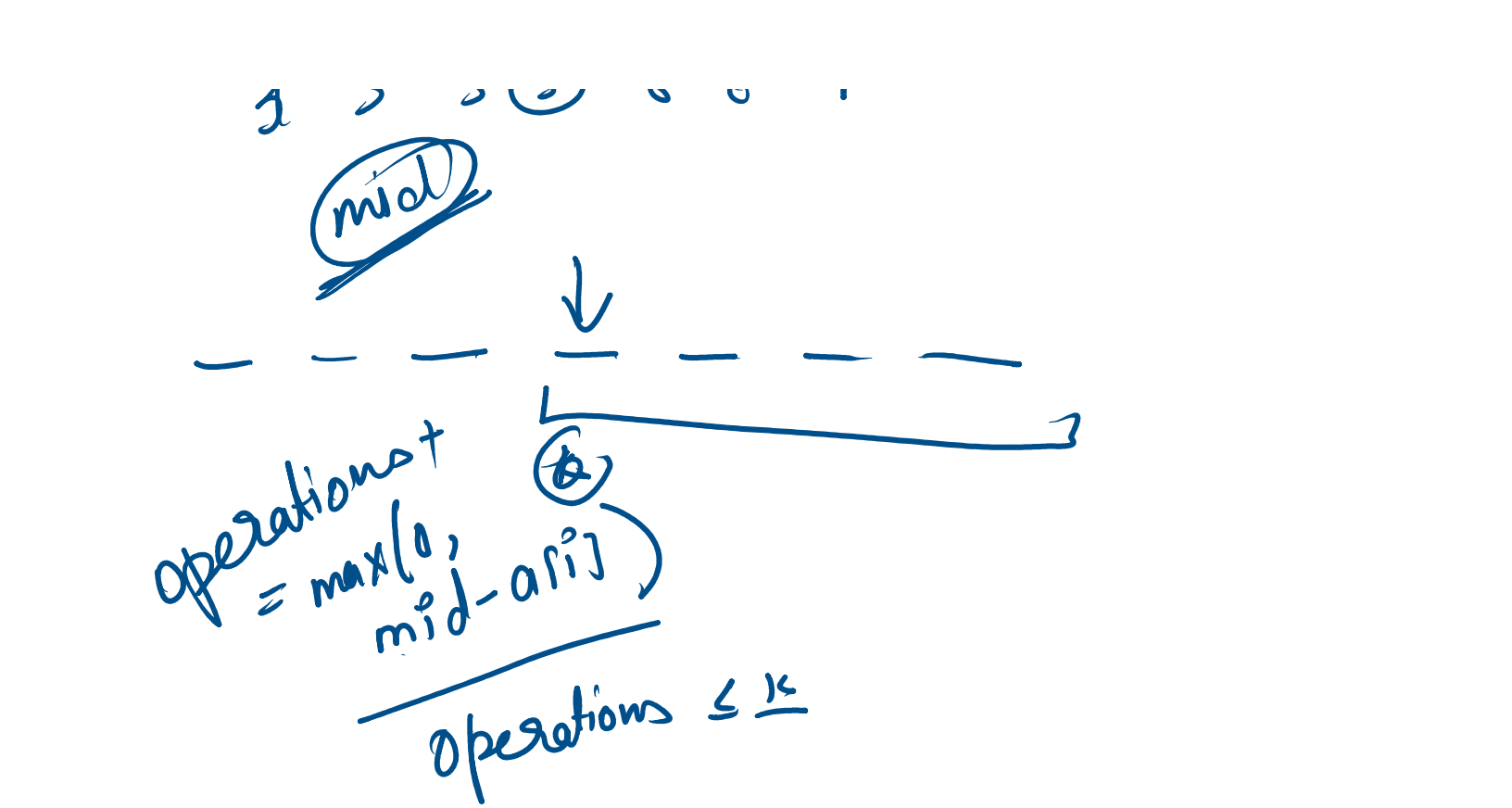
f(mid)

low = mid + 1

else

high = mid - 1





low

high

high = 1e9

1e10

h = 100

100

high = 1e9

low = 0

$\log_2(1e9)$ ≈ 32 < 18

nlogn

nlogn

high

$\log_2 = 32$

high = 1e18

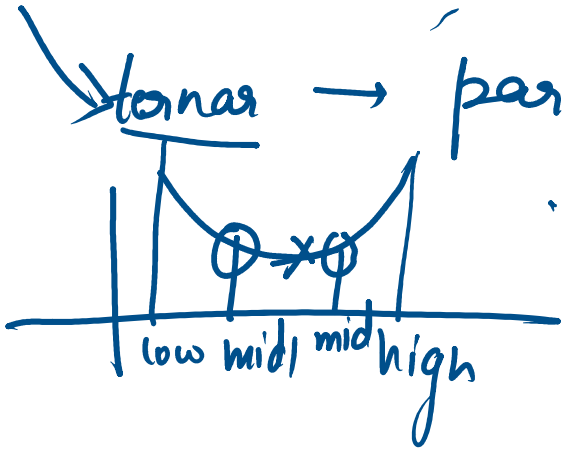
$\log_2 1e18 \approx 64$

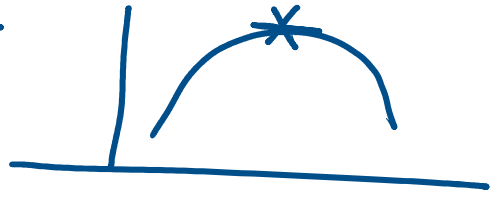
$2n \log n$

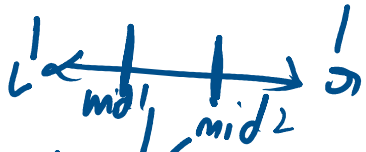
function →

monotonus

tonar → parabola → ternary





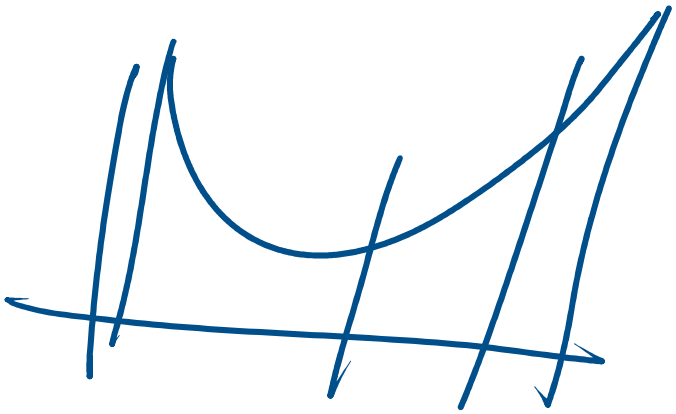


low high n ⇒ 1/3

low

$$\begin{cases} mid1 = low + \frac{high-low}{3} \\ mid2 = high - \left(\frac{high-low}{3}\right) \end{cases}$$

$\hookrightarrow \log_3(n)$

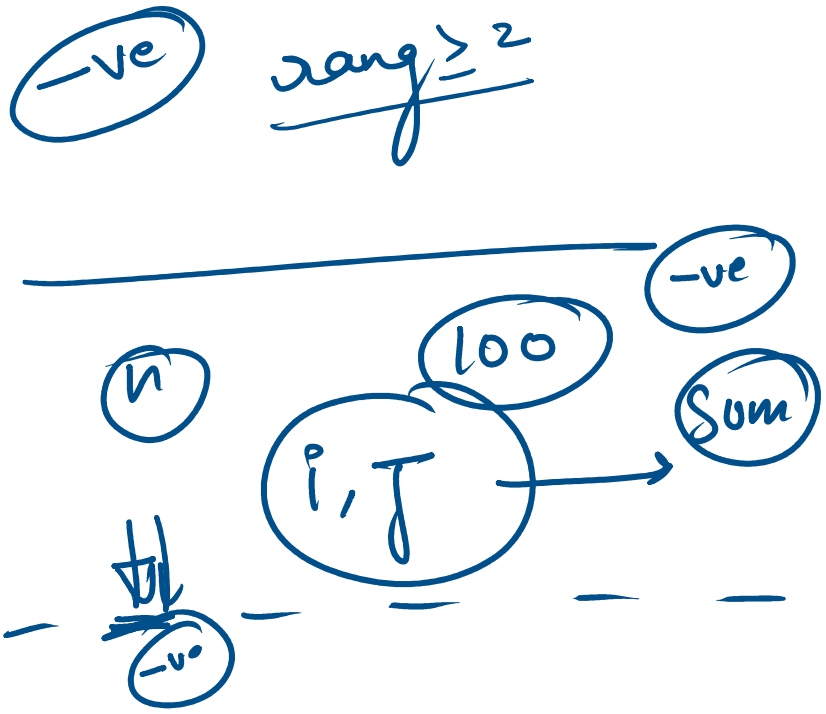


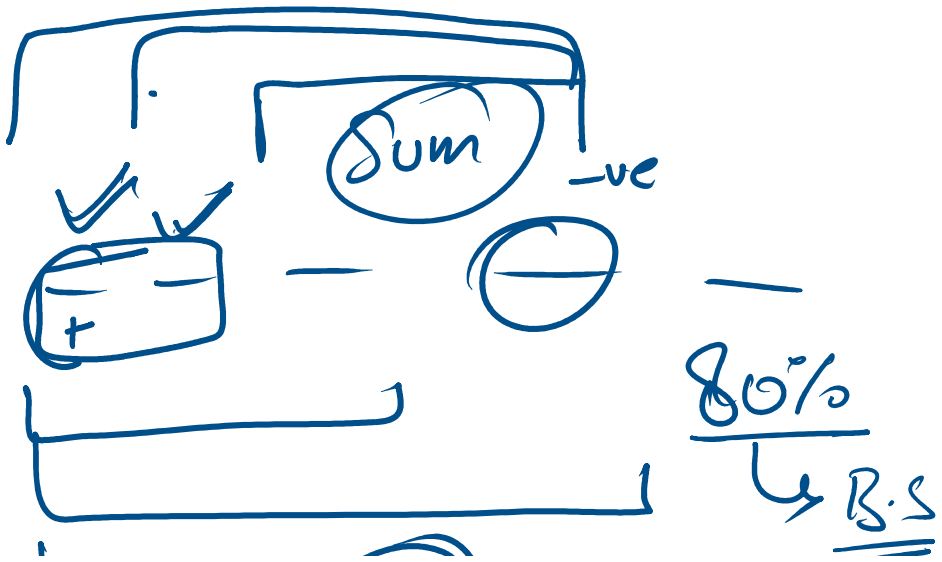
input

output

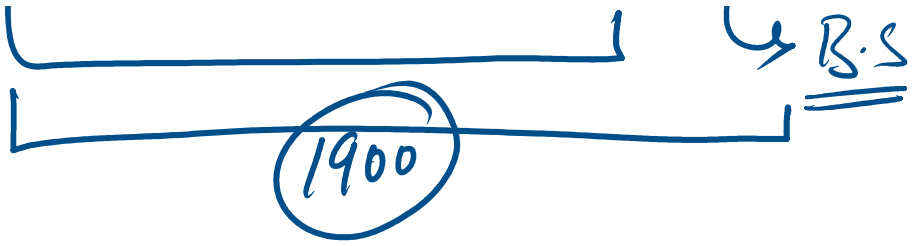
ask → get the final

get the final

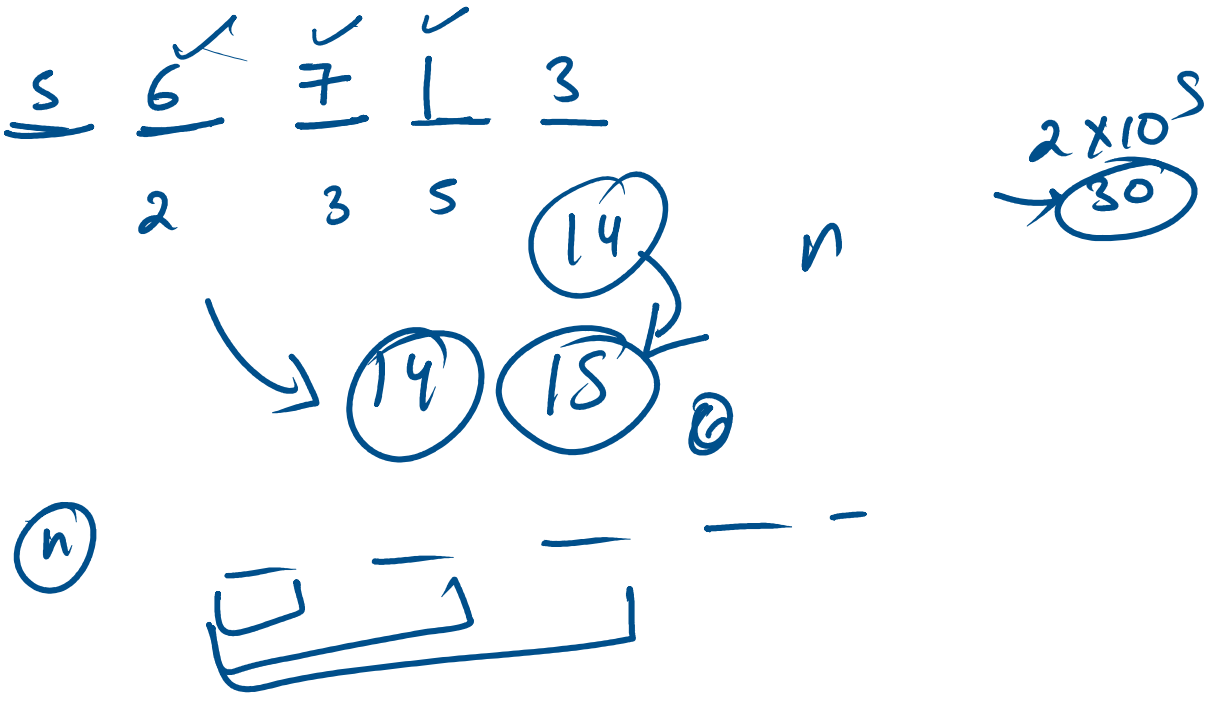




1500
1700
3000
Bs



$\log_2(2 \cdot 10^5) \leq 30$

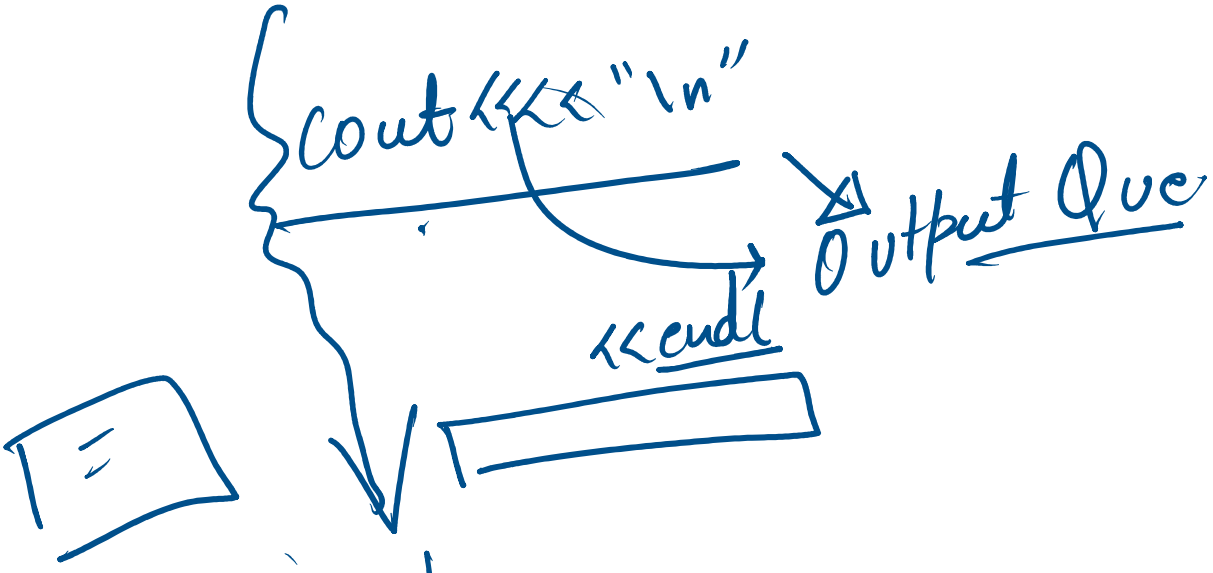


Weight = #stones

prefix sum

$\log_2(n)$ $< \log_2(2 \times 10^5) = 20$ (17)

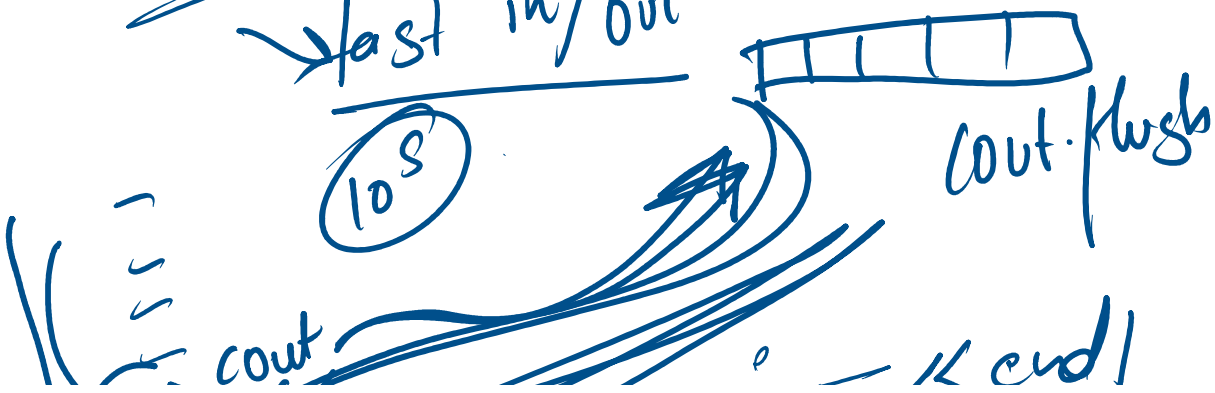
(18)

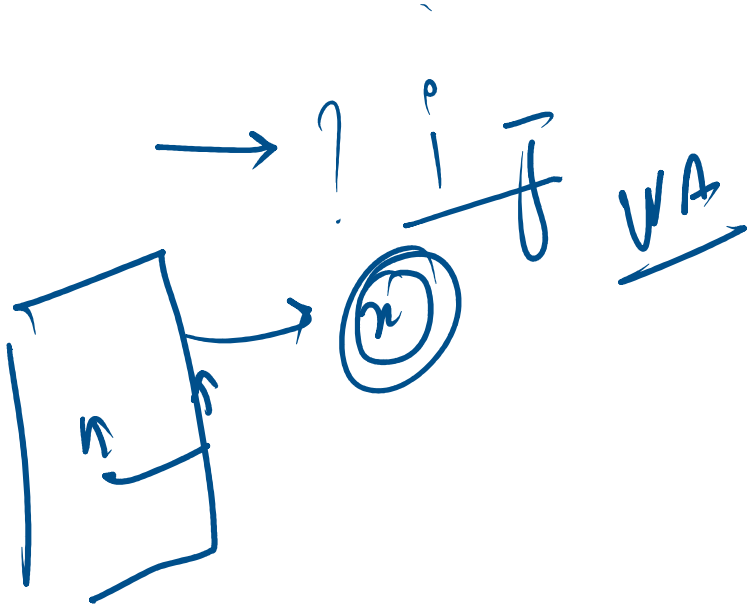


Interact ×

last in/out

10^5





70%

30%

Common pao

BS → (monotonic)

↳ f(x)

↳ how to app

Tes Scar → Basic

→ ∪ ∩

→ BSX

interactive → what

interactive → what B.S code

Advanced problems

mistakes

while (low < high)

while (low <= high)

int low, high, mid, ans

while (low ≤ high)

{ mid →

if (f(mid))

{ → ans = mid

high

}

else

low

else

3 high

cout << ans

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

low = 1 high = 2

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$

① low = mid

BS

Advanced

→ r - l > {error}

low = 0 10^8 try 0.0001

low

mid -ve low

int low = 0

l h 1e18

while

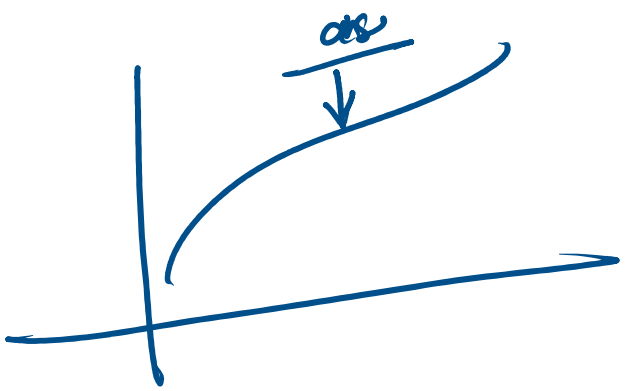
while

{ mid

}

array → a → first pos ≥ e

more { lower_bound
upper_bound }



integer

floating/decimal

low = 0 high

while (low ≤ high)

while (low ≤ high)
{

} while (low ≤ high

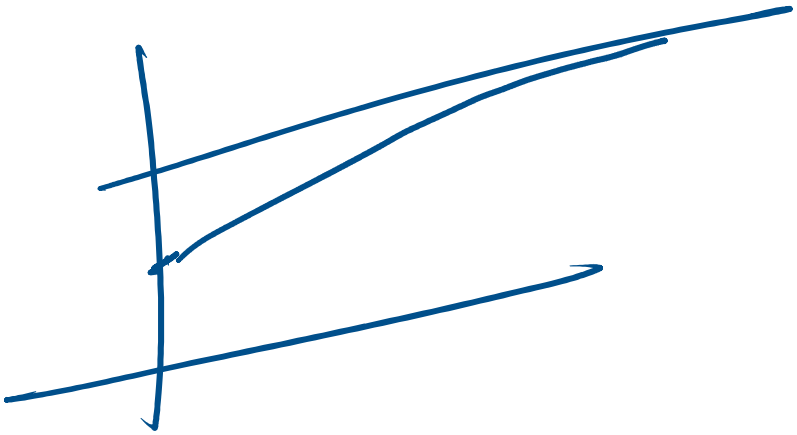
low = 0 high = 1

↓

0.5

low = 0.5 high = 1

0.75



while (high - low > error)

while (

10^{-6}

high = 1e9
low = 0

mid = (low + high) / 2

80

$\log_2 10^{15}$ ↑ 80

mid ↳ 10^9

mid * 10^{-6}] 10^{15}

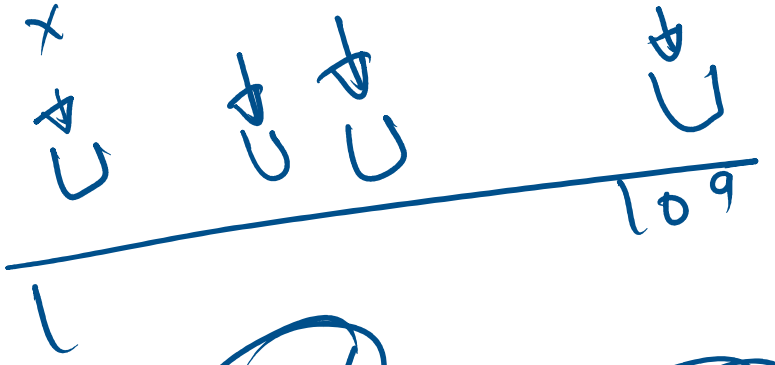
for (iter=1; iter<80; iter++)

while (high - low ≥ error)

Error 10^{-6}

while (low ≤ high)
{high-low ≥ eps}

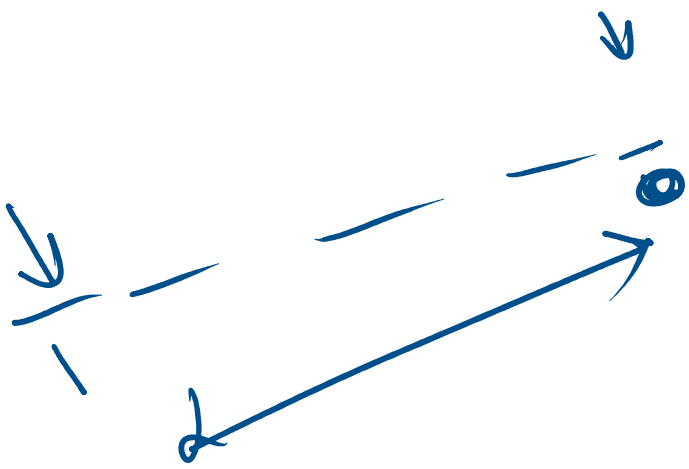
for (so)



max

c ≤ n
c is the no. of cows

a.
while( )
{ mid
if (func (mid)
{

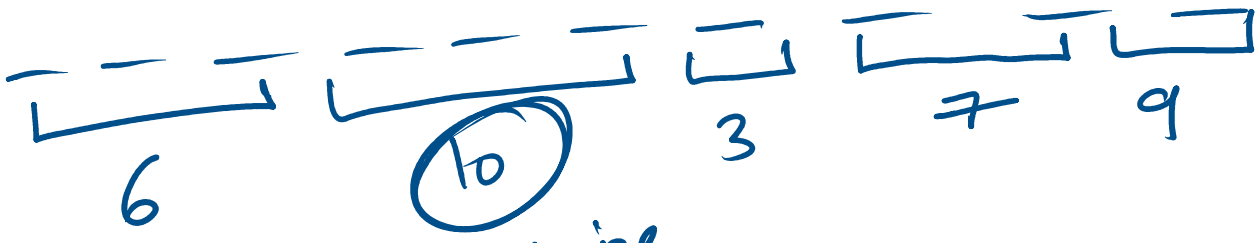


books → N

→ A[1], A[2] — — — A[N]

(m)

M S



max Sum

minimize

ans

low = 0, 10^9

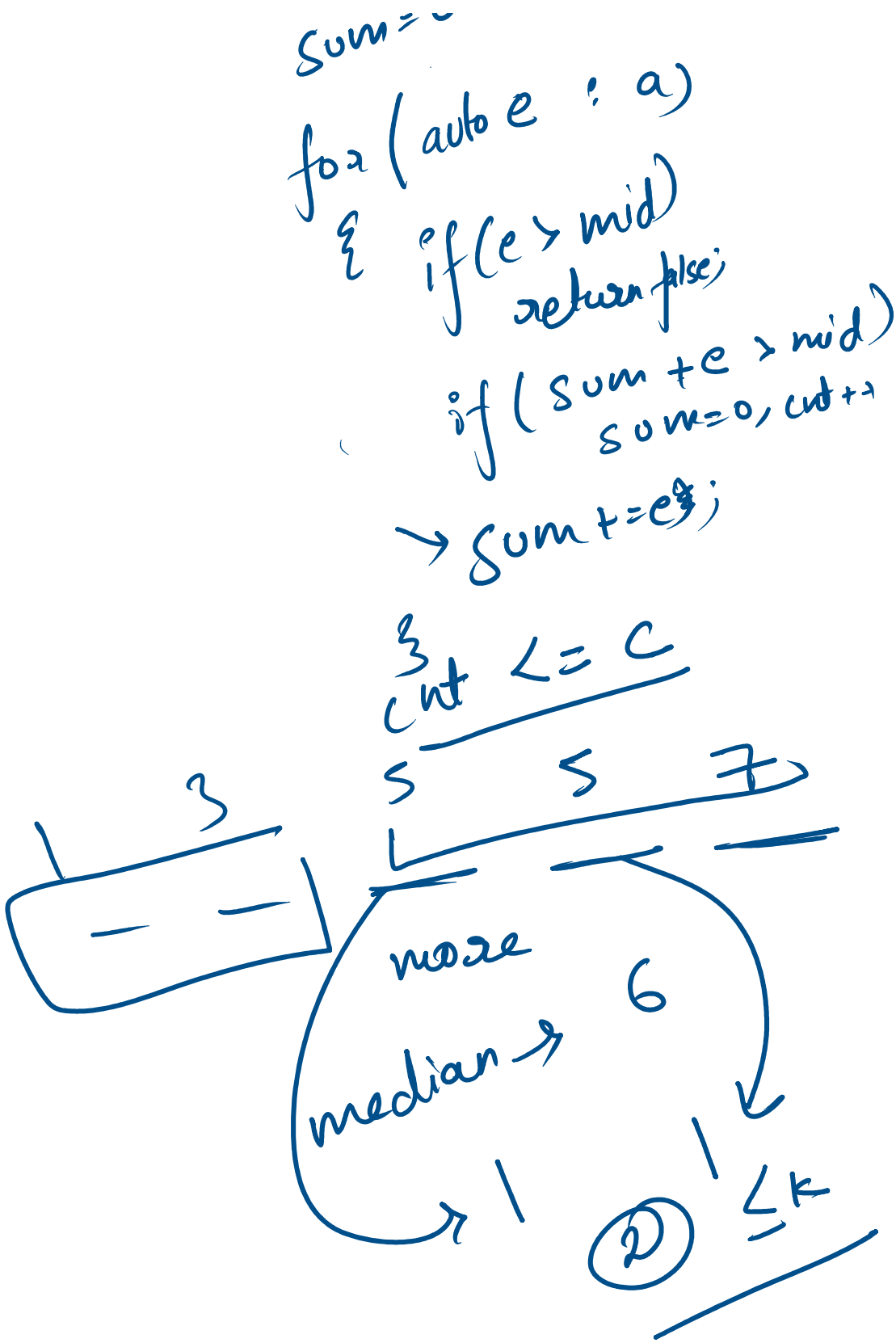
mid

func(mid)

{ cut = 0
Sum = 0

func

mi


sum=0
for (auto e : a)
{ if(e > mid)
return false;
if (sum + e > mid)
sum=0, cnt++
→ sum += e;
}
cnt <= c
5 5 7
3
more
median → 6
2
≤ k