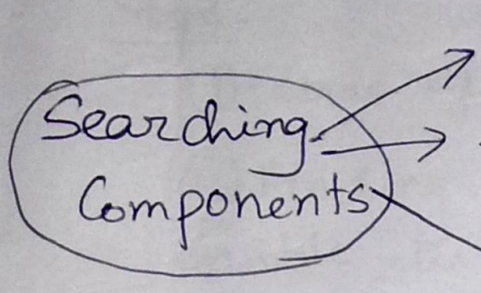
class-1

Predicate framework of binary search

(Abstract concept)



- Item
- Search Space.
- Algorithm for searching (Binary search is one such algorithm)

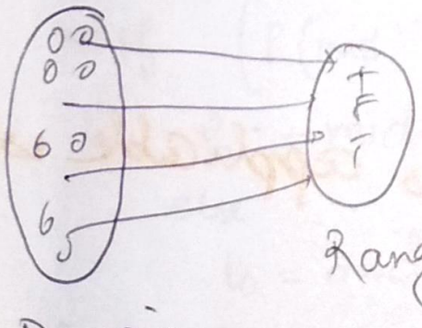
Binary search is an algorithm which can be applied on a search space having some certain properties.

① What are these properties of search space which enable you to do a BS?

* When?

→ i) what is a predicate?

→ A binary / boolean function that takes an arbitrary data as input and maps it to true or false.



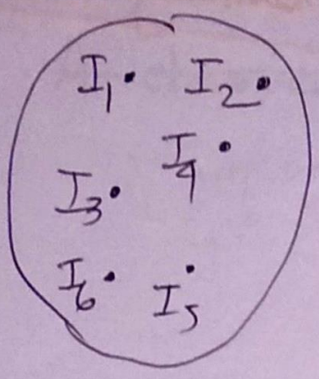
Domain

**

Binary search is applicable on a search space iff ∃ a P(x) such that if we define P(x) over the search space

then, $P(x) \Rightarrow P(y) \quad \forall \quad y > x$

① we have a S.S



② Define a function / predicate

$P(x)$.

$P(I_1) = T/F$
$P(I_2) = F/T$
$P(I_3) = F/T$
⋮
$P(I_9) = F/T$

(**)

$P(x) \Rightarrow P(y)$ $\forall\ y > x$: if $P(x)$ is true/false then $P(y)$ is T/F for all $y > x$

SS: I_1 I_2 I_3 ..... I_n

F F F .. T T T T

- **Conclusion:**

If Search space can be reduced to a form F*T* or T*F* then we can apply binary search over that search space.

② If Binary search is appliable what all we can find?

→ Using binary search we can find (last F - First T) or (last T - first F).

① F F .... F F T T ... T T

② T T ... T T F F .... F F

Framework of applying Binary Search:

Class-2

Steps-

① Come up with a predicate < F* T* / T* F* >

② Finding last F/T and first T/F should enable us to solve the problem.

Problems → explicit search question
Problems → optimization problems

Last F:

P(SS): F F F (F) T T T T
lo ↑ hi ↑

lo = 1, hi = n

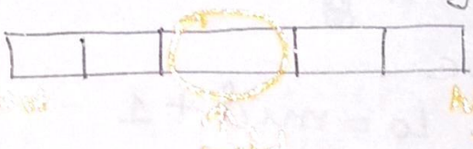
```
while (lo < hi)
{   mid = lo + (hi - lo + 1)/2
    if (P(mid))
        hi = mid - 1;
    else
        lo = mid;
}
if (!P(lo)) return Found.
else does not exist
```

We have to use upper mid here.
lower mid will lead to infinite loop

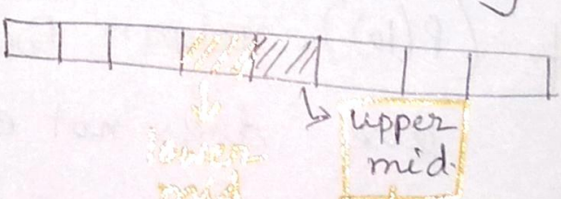
Possible mid →

two types of mid:

1> odd sized array.



2> even sized array.



input → last F

lower mid
F T
↑ ↑
lo hi

upper mid.
F T
↑↑
lo hi

lower mid: $\frac{lo+hi}{2}$

upper mid: $\frac{lo+hi+1}{2}$

$lo + \left(\frac{hi-lo}{2}\right)$

$lo + \frac{(hi-lo+1)}{2}$

handles int. overflow

Trick: ~~which update~~

Which variable is excluding the mid took that] mid (higher/lower)

F F F (T) T T

find

$lo = 1$, $hi = n$

while ($lo < hi$)

{ $mid = lo + \left(\frac{hi-lo}{2}\right)$

if (P(mid))

$hi = mid$

else

$lo = mid + 1$

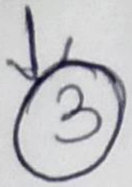
}

if (P(lo)) return found

else does not exist.

Problem 1 : (Leetcode) (34)

First occurance



2 2 3 3 4 4.

Last occurance

FF

Predicate:

$x >= target$

2 2 3 3 4 4.

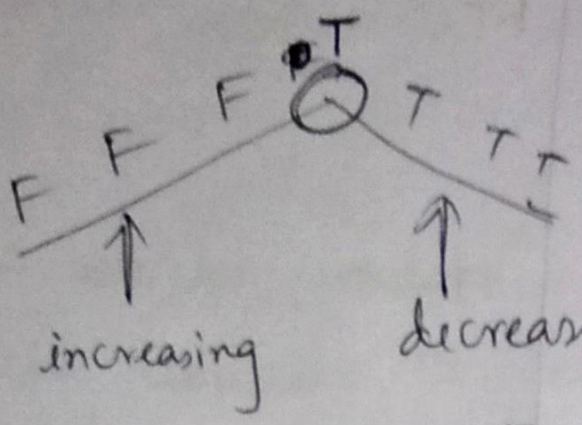
F F T T T T.

↑

find first T.

Problem-2 (Problem – 852)

Peak index in a mountain array



mathematically separate the two parts.

1. Predicate : $A[i+1] < A[i]$
   ↳ find first T
2. Predicate: $A[i-1] > A[i]$
   ↳ last F

---

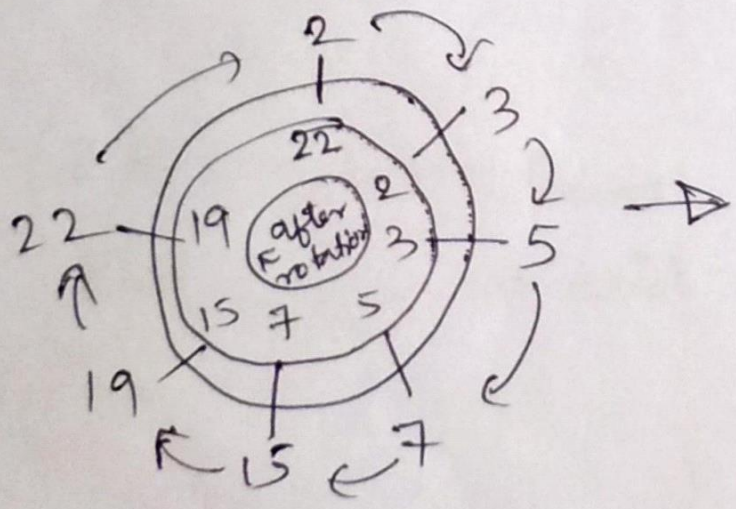
class-3

Problem leetcode-153

Rotated and sorted, find the min.

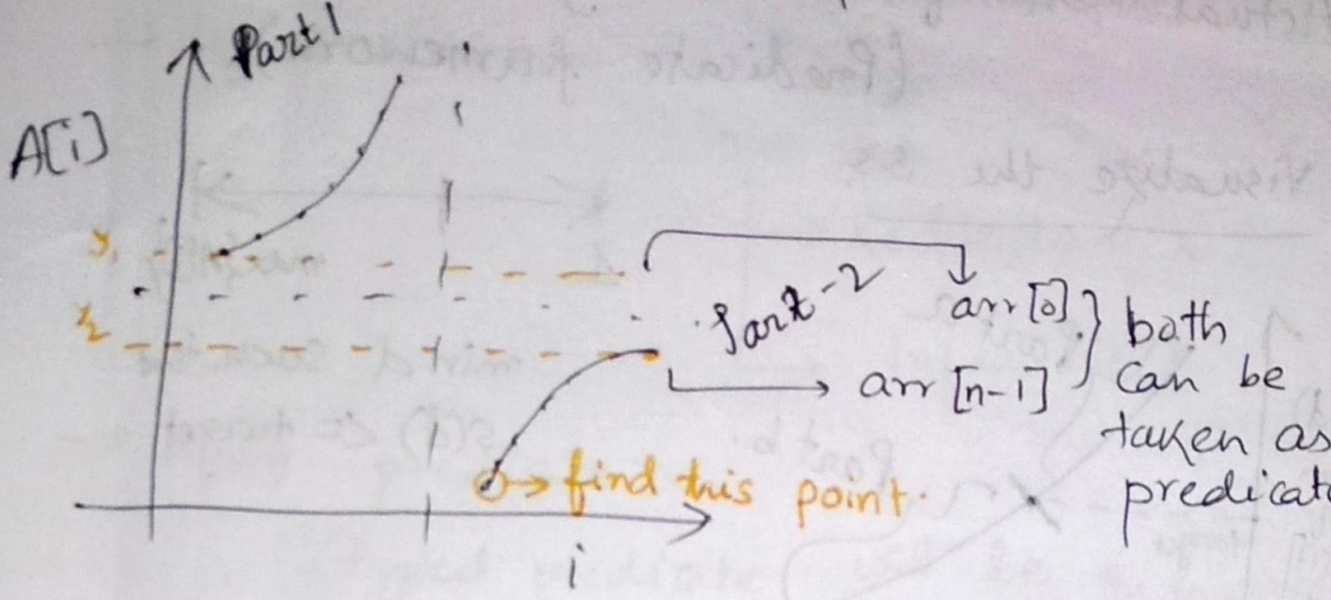
2	3	5	7	15	19	22

Rotate by 1 element (Clockwise) -



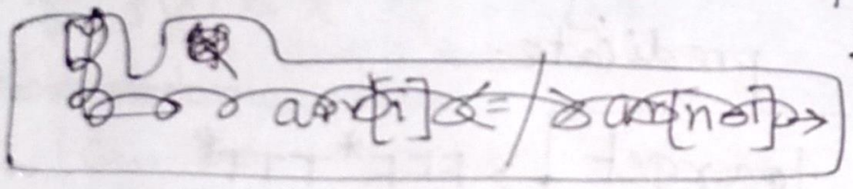
It's an explicit search question
↓
search space and item to search is given.

Visualization of the search space →



we have to come up with a predicate/function so that we can distinguish the two parts.

↳ arr[i] < arr[0] → ~~TTT*~~ FFF*TTT*
find first T.



---

* Optimization problem:

1283. Leetcode - Find the smallest divisor given a threshold.

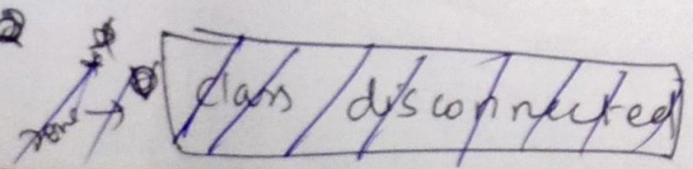
optimization,

- minimize/maximize $f(x)$ under certain conditions.
- here minimize d condition $S(d) \leq$ target

Point of view 1:

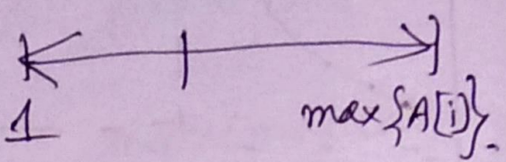
Search for minimum d out of all possible d.

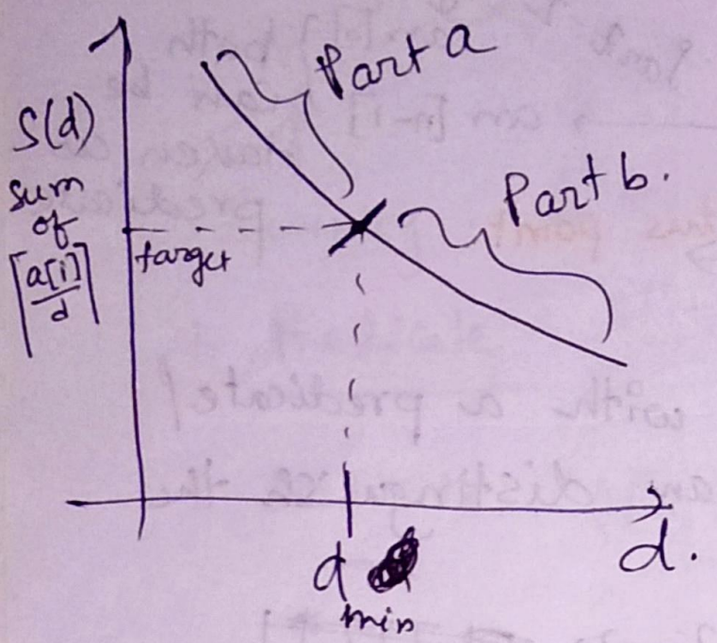
① Search space → 1 to max. element of array



② Actual task of searching.
(Predicate framework).

Visualize the S.S.





min d such that
S(d) <= target.

We have to ~~d~~ its distinguish part a and part b using predicate.

P(): S(d) <= target → FFF*TTT*

0:12 Disconnection

Optimization Problem Pattern

① we will be given a quantity (x) to minimize or maximize with a constraint ⇒

$f(x) \leq T$
$\geq T$

This is the general pattern for using binary search. (~~Not to~~ Can not be used always but useful most of the time)

Solution:

→ define a S.S → Naive search space.

→ Trim S.S
- → Redundant X
- → Infeasible X.

→ Apply prediate.

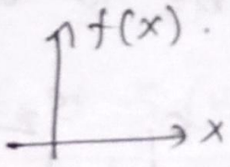
typical predicate will be the constraint

↓

$f(x) \leq T$

$\geq T$.

To get the intution we ~~can~~ have to see $f(x)$ vs x graph.



⊙ if $f(x)$ vs x is monotonic and it holds all the above conditions then ~~monotonic then~~ B.S can be applied.

Q. 1011. Capacity to ship packages within D days.

here, x = weight capacity → what we want to optimize

$f(x)$ = no. of days it takes with ~~max~~ X capacity.

~~min X S.~~

minimize x such that $f(x) <=$ given days

Search space → $\max\{A[i]\}$ ——— $\sum A[i]$

Tips – 50% medium, 40% hard, 10% easy → online test.

→ practice.

Class-4

Agenda: optimization problems of B.S
B.S on matrices.

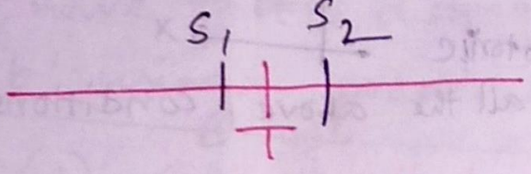
Q.1

1300. Sum of mutated array closest to target. (leetcode)

Optimization:

~~We have to optimize~~

Sum should be as close as possible to T.



largest $S_1 \leq T$ & smallest $S_2 \geq T$

~~if we can find largest $S_1 \leq T$~~

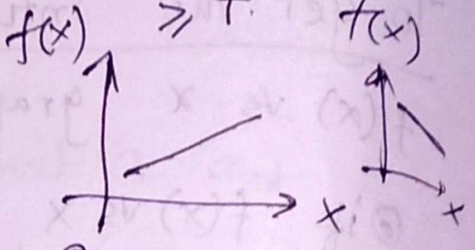
~~$S_2 = S_1 + 1$~~

Class of opt..

minimize/maximize X such that-

$f(X) \leq T$

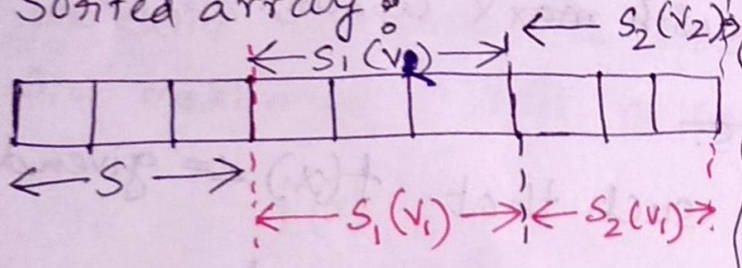
$f(X) \geq T$

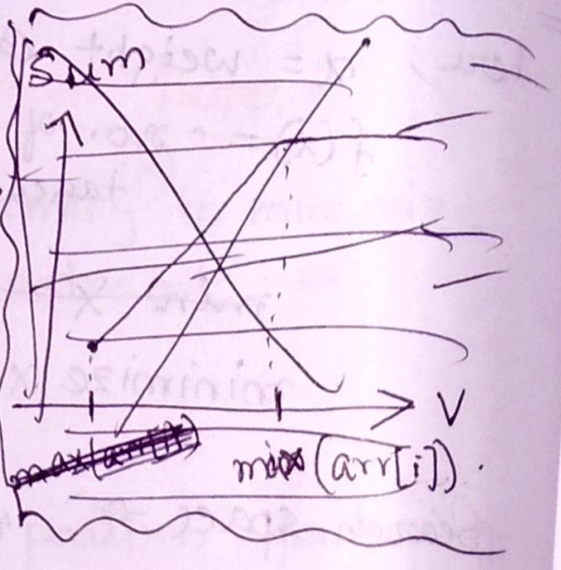


① f(x) should be monotonic.

Find S_1:

Sorted array:





if $V_1 < V_2$

$S(V_1) \leq S(V_2)$

if we use V_1

Sum of array = $S + S_1(V_1) + S_2(V_1)$

if ~~Q.~~ we take V_2,

Sum of array = $S + S_1(V_2) + S_2(V_1)$

Now,

$S_1(v_1) < S_1(v_2)$ | ∵ that part contained. $> v_1$ integers which was replaced by v_1.

$S_2(v_1) < S_2(v_2)$ | ∵ $v_1 < v_2$
∴ $v_1 * x < v_2 * x$

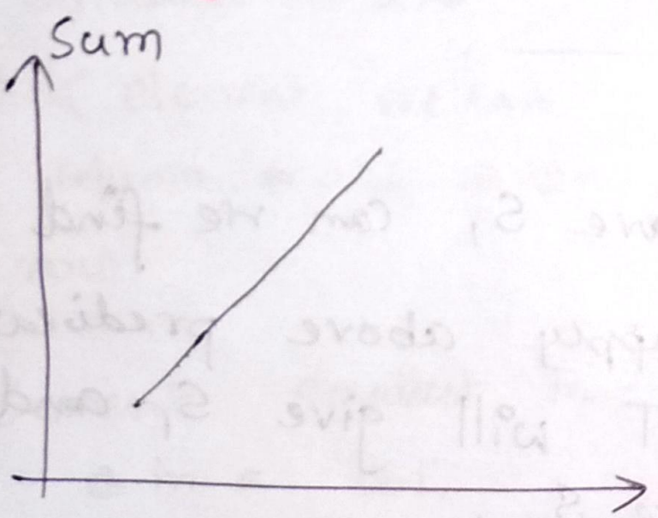
[x is the length of the 3rd/last part of array]

∴ $S_1(v_1) + S_2(v_1) < S_1(v_2) + S_2(v_2)$

$\Rightarrow S + S_1(v_1) + S_2(v_1) < S + S_1(v_2) + S_2(v_2)$

$\Rightarrow$ ~~$S(v$~~ $\boxed{Sum(v_1) < Sum(v_2)}$

∴ as value increases sum of modified array will also increase.



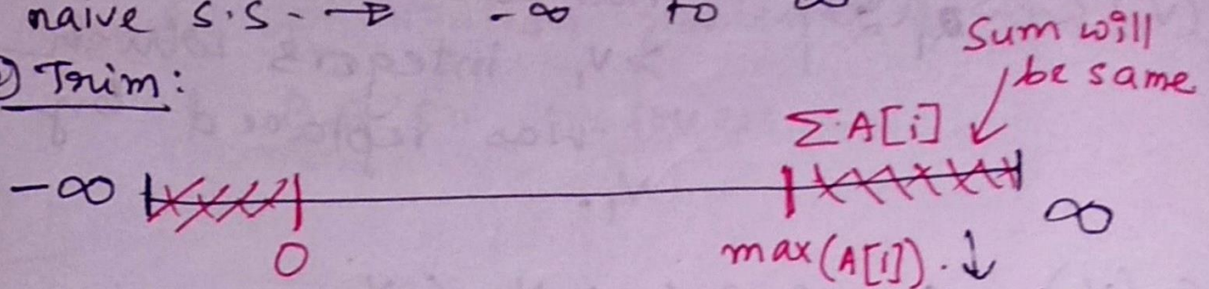
∴ finding largest S_1 became B.S problem.

$x =$ ~~Sum~~ value. $\quad \frac{f(x)}{Sum} \leq$ Threshold.

① Find out the search space.

naive S.S. → $-\infty$ to ∞

② Trim:



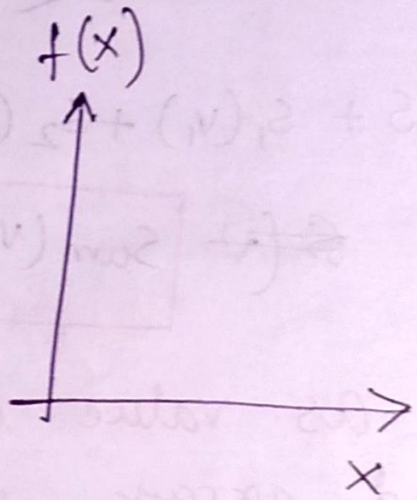
for same sum we have to find the minimum element. That's why we can eliminate this part. ~~the min el~~

② Apply the P.F.

P: $s(v) \leq T$

T T T T F F F F

we have to find.



S_1, S_2

Q. if we have S_1 can we find S_2.

if we apply above predicate then last T will give S_1 and first F will give S_2.

$$S_1 = f(V_{opt})$$

$$S_2 = f(V_{opt}+1)$$

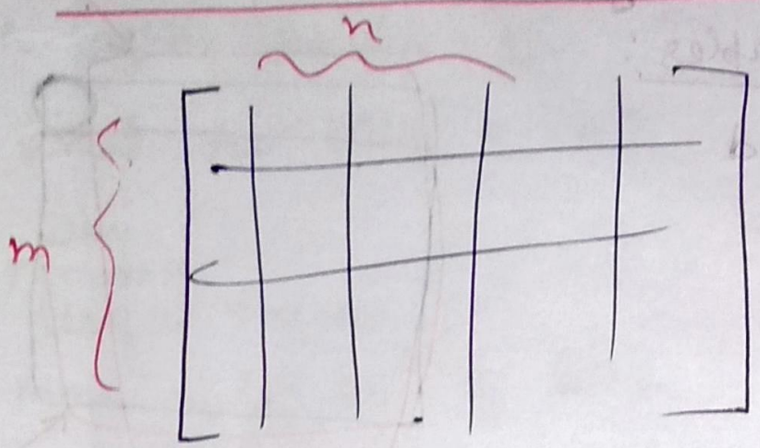
if (S_1 is equal/closer than S_2) return V_{opt};

else return $V_{opt}+1$;

8.2. B.S on matrix

74. Search a 2D Matrix

240. Search a 2D Matrix II:



$O(m+n)$ - optimal algo.

Reduce S.S by a row or by a column.

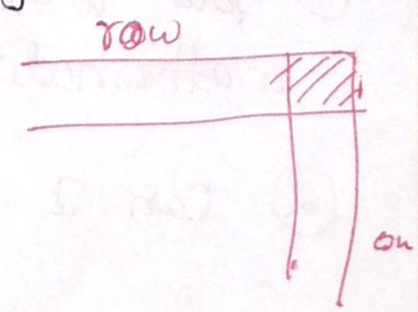
$O(m)$ → Reduce the S.S. by one row

$O(n)$ → " " " " " Column.

∴ $O(m+n)$ Reduce S.S by one row or one Col.

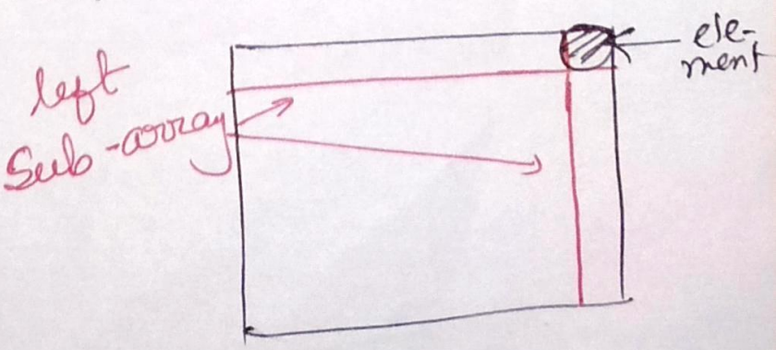
Q. which element I should compare it with?

→ an element that is largest in a row and smallest in Column.



if target < element, we can drop the column else we can drop the row.

→ an element smallest in a row and largest in a col.

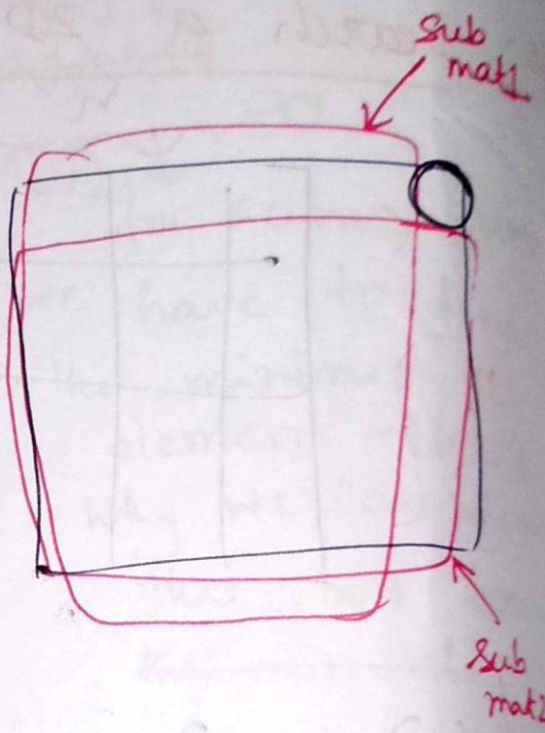


Search ← maintain S.S.

How to represent/maintain the search space mathematically.

Keep two variables:

row start and
col end.



Given a Matrix.
all positive integers.
find a subarray in the matrix
sum of the subarray is as large as possible and $S \leq t$

① How to represent the Search space mathematically?

② Can I fit this in the template?