Recap : Sorting ✓ → $O(n^2)$ → $O(n \log n)$ → in-built sort f? | Compare f?

QS MS

partition

SEARCHING. | as fast as possible.

Problem : arr = [3, 1, 5, 2, 4, 8, 7] → 7 elements.

0 1 2 3 4 5 6

give you a 'key' ⇒ at what position it exists.

search (arr, 2) ⇒ 3

search (arr, 6) ⇒ -1

search (arr, 10) → -1

Idea 1 : Brute force. / Linear search.

mutated

arr = [....] n elements

```
for int i = 0, n-1
    if arr[i] == key:
        return i
return -1
```

T.C of search = $O(n)$ BAD.

Avg time complexity to add 1 number to the array = $O(1)$

push credit cards. ← { Banks do perform analysis on volumes of transactions.

10^8 elements = 1 sec.

not huge when it comes to searching

50 cr

1 cr = 10^7

10 cr = 10^8

50 cr = 5×10^8

Idea 2 : Use Binary search.

Data ⇒ 1 time sorting | one time effort.

n elements → $O(n \log n)$ cost. | negligible ∵ 1 time.

arr = [1 2 3 4 5 7 8]

0 = low, mid, high, mid, low, 3, 7 = high

search = 3.

mid = $\frac{l+h}{2} = \frac{0+7}{2}$

$= 3 //$.

```
while low <= high:
    mid = (low + high) / 2
    if arr[mid] == key:    return mid
    if arr[mid] < key:     low = mid+1
    else:                  high = mid-1
return -1
```

low <= high

mid = $\frac{0+2}{2}$

$= 1$

mid = $\frac{2+2}{2} = 2 //$

T.C. of search = $O(\log_2 n)$

Best sorting algo ever !!

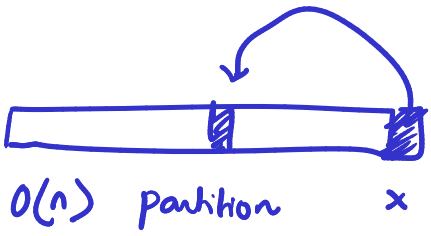
NOPE !!

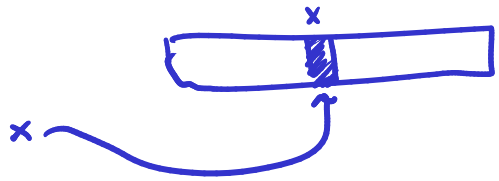
$8 \to 4 \to 2 \to 1$ (3)

$2^3 = 8$ $\quad 2^x = n$

look ups = $\log_2 n$

Avg time to add 1 element in the array = $O(n)$ problem



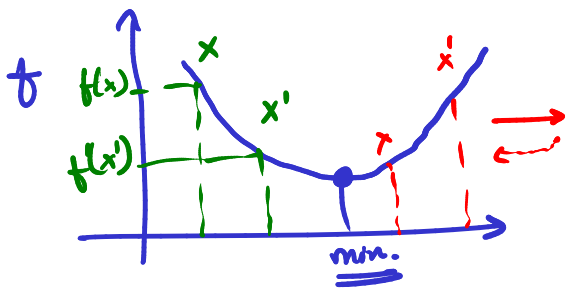


Use case : ① If no more new elements will be added and search op needs to be performed only after sorting, then BS is used.

② Don't think BS is for searching an element in the array.

- Get value of x, f(x) ⇒ note down
  x' new value, f(x') x' > x

↘ you are going right. →



- $x' > x$ and $f(x) < f(x')$ ⇒ you are in wrong direction.

↗ you should go back.

Consider BS as more generic search : It helps you find optimal value x in a range in $O(\log_2 n)$ time/iterations.

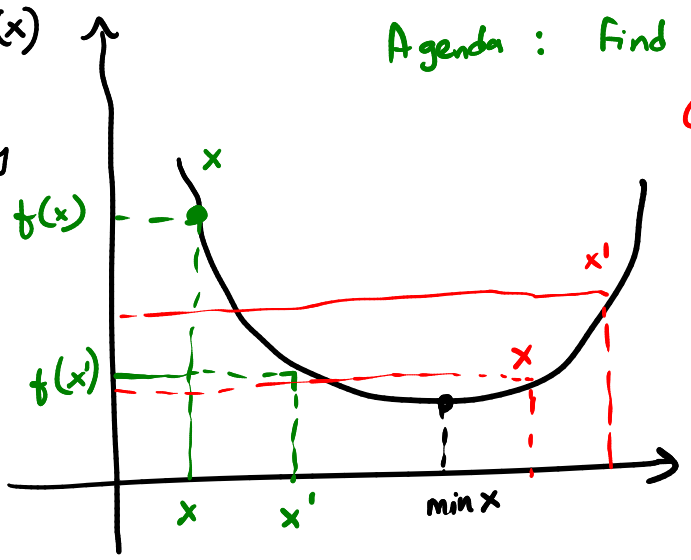
Repeat

$x < x'$ → compulsory

Case 1 : $x \rightarrow x'$

The value of f^n has reduced.

I am in correct direction. (To right)



Agenda : Find minima

Case 2 : $x \rightarrow x'$

The value of f^n increased.

I am in wrong direction. (To left)

Directions in which you should move

2 Examples :

Q.1 n piles of coins. coins = [2, 3, 5, 2, 1]

1 2 (2) 3 5

3 operations :

✓ (A) You can add 1 coin to any pile.

✓ (B) You can remove 1 coin from any pile

this wasn't there !! ⇒ (C) You can pick a coin from one pile and put it on another.

✗

In how many min ops, you can make all piles equal ?

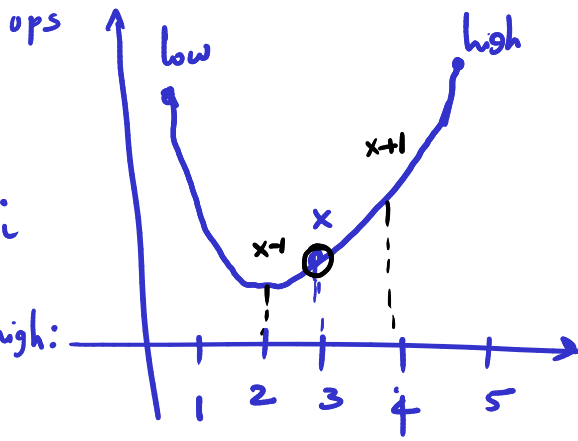
→ # coins.

Sol^n : 1st instinct : central tendency → ~~Mean~~ ✗ median ✗ ~~mode~~ ✗

Range : [min(arr), max(arr)] ⇒ [1, 5]

low = 1 high = 5

mid = $\frac{1+5}{2}$ = 3

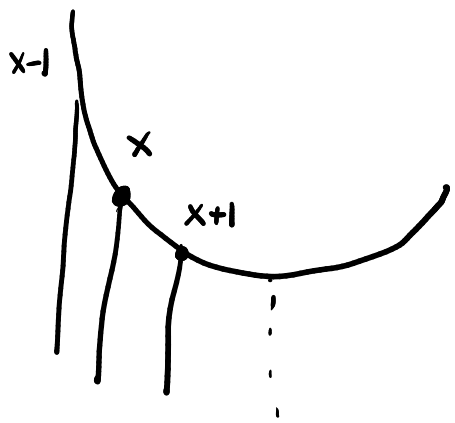


f: cost (x, arr) : cost += |x − arr[i]| for all i

f(low) f(mid) f(high) : low <= high:

x ⇒ mid x+1 x−1

(1)

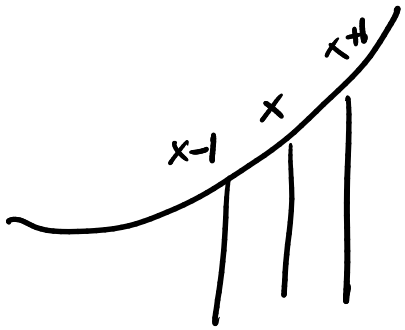


f(x-1) > f(x) > f(x+1) ⇒ x is mid

low = mid

f is cost f^n

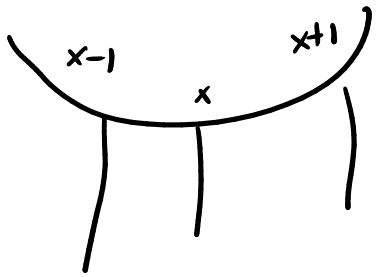
(2)



f(x-1) < f(x) < f(x+1)

high = mid

(3)



f(x-1) > f(x) < f(x+1)

return x

Q.2 N chefs Each chef takes q_i time to make a dish.

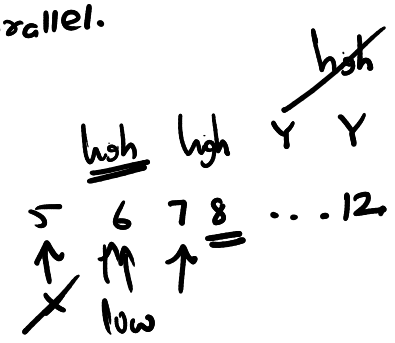
3 chefs arr[] = {2, 3, 5} # dishes I want = 6 //

min time required to make these dishes. chefs can work in parallel.

Sol?

min_time = 1 low

max_time = 12 high



x min # dishes in x minutes = $\sum_i \frac{x}{arr[i]}$

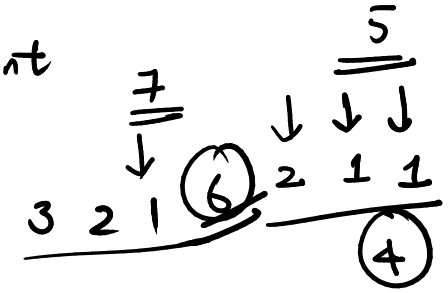
while low <= high

if # dishes in x min >= # dishes you want

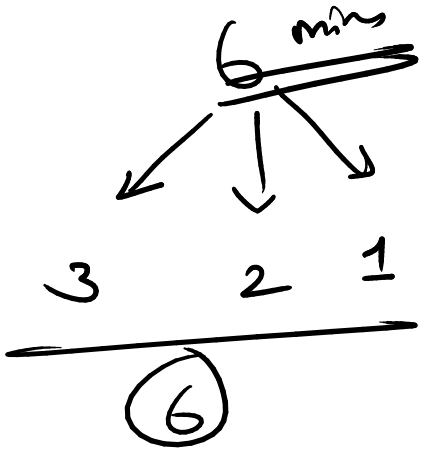
high = x - 1

else

low = x + 1



low ⇒ ans.



int mid = 0;

while low <= high:

mid =

{

mid → ans.

— x — x —

Idea 3 Search TC : $O(\log_2 n)/O(1)$

Insertion * : $O(\log_2 n)/O(1)$

??

TO BE CONTINUED