Recap: 2 problems leetcode → sliding window technique

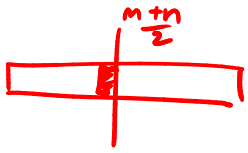
- set container
- map container

Binary search:

median of 2 sorted arrays

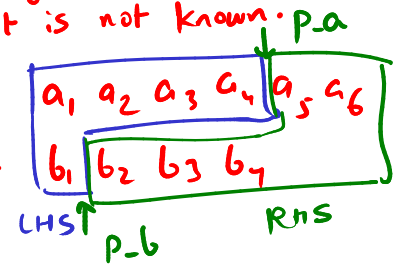
↳ complicated approach.

1.



fixed window (k) ← size of { subarray / substring }

it is not known.



m+n ⇒ odd

median = max(a[p_a], b[p_b])

p_a → guess : 0 to m-1

p_b = (m+n)/2 - p_a

m+n ⇒ even

~~int~~ float median = [max(a[p_a], b[p_b]) + min(a[p_a+1], b[p_b+1])] ÷ 20

p_a is correct :

ⓐ ✓ ⓑ ✓

ⓐ a[p_a] < b[p_b+1]

ⓑ b[p_b] < a[p_a+1]

} ⇒ ?

p_a is overestimating. ← high = p_a

ⓐ ✗ ⓑ ✓

low = 0

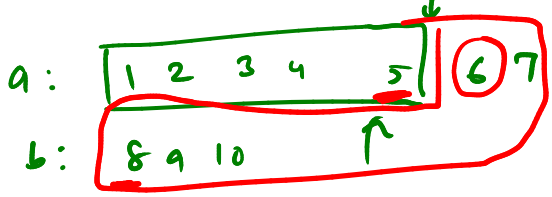
high = m-1

p_a = (low+high)/2

p_a is underestimated ← low = p_a

ⓐ ✓ ⓑ ✗

Corner Case:

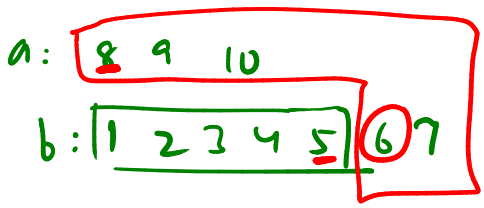


(m+n)/2 = 5

p_b = 0

1 2 3 4 5 6 7 8 9 10

5.5



Containers. Abstract Data Type. specific use case

① set.

[seq. duplicates] → i/p → blackbox (bulk of code, arrays) → o/p → [seq. unique]

O(n).

arr = [... n ...]

push all elements in the set.

↓

only unique elements

ops

SET.

Remove duplicates in O(1)

O(m·n)

no. of unique elements

BF (without set): linear search

unique_arr = [ ]

arr = [- ∅ ∅ -]

② map.

n elements

i/p → bulk of code → o/p

keys has to be unique

u1 : f1

u2 : f2

u3 : f3

Counter

C++

MAP

O(n) → Counter./Map Frequency count → O(1)

Conclusion: Black boxes have some specific use case for which they are powerful → which arrays cannot do generally in same time complexity.

How?

You sacrifice something, to gain something!

blackbox

array

arr[i] → O(1)

array

X

loop on array

```
for(int i=0 ; i<n ; i++)   ← for loop
while (i<n): ...
```

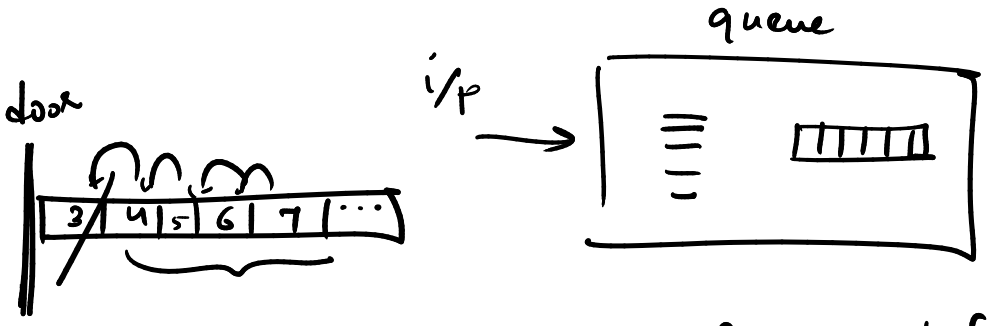
containers / collections:

```
for (auto x : container)   ← for each loop
    x
```

FCFS

③ Queue. 3 4 5 6 7

3 ops: O(1)



1. add element from the back.
2. What's the first element you inserted? What element should I serve?
3. Remove that first element in O(1).

O(n) array

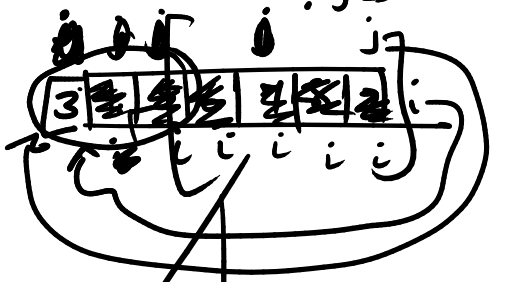
array x

front a[:]

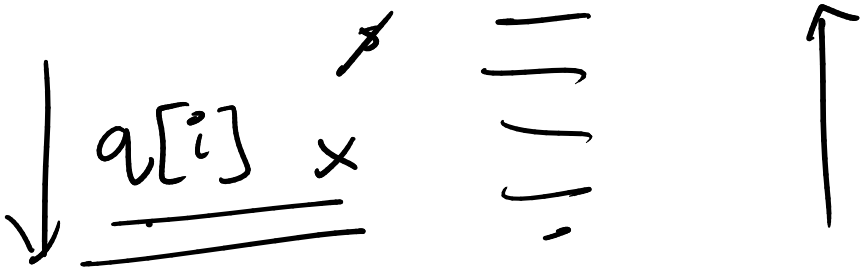
3

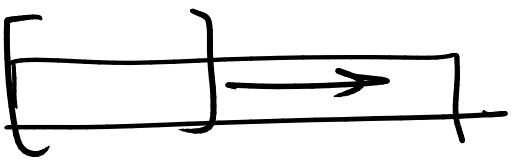
Remove the 1st element

i ⇒ start
j ⇒ end



4 8 2 3



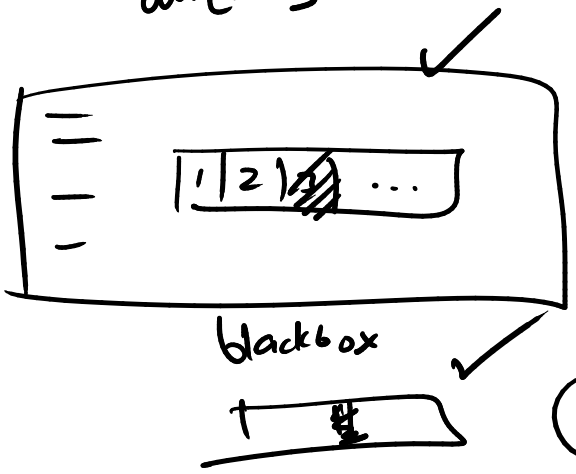


LCFS

1 2 3 ... a[n-1]

3 ops. O(1)

④ stack

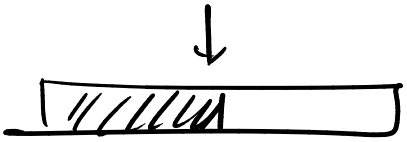


① Add element from the back

② Tell me the last element.

③ Remove last element.

array is dynamic



amortization

Computer scientists.

FIFO : first in first out

LIFO last in first out

Fact:

stack
C++ Java
queue

stack x

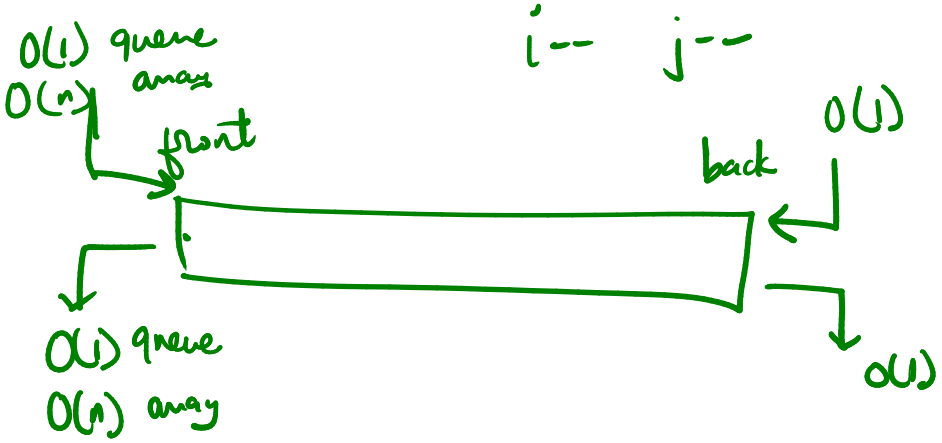
Python ⇒ use lists
deque

Remove an element from queue → enqueue dequeue

deque

i and j

i++ j++
i-- j--



→ doubly ended queue