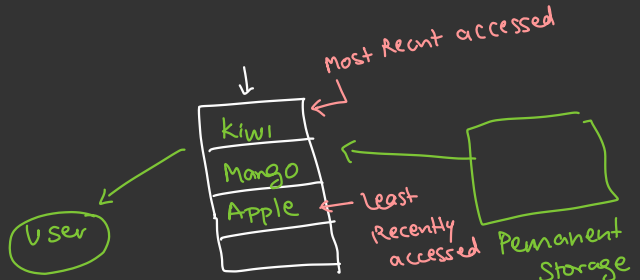
LRU

Least Recently Used "Cache"

(eviction Strategy)



Q1 – Apple

Q2 – Mango

Q3 – Kiwi

Q4 – Apple

max limit = 4

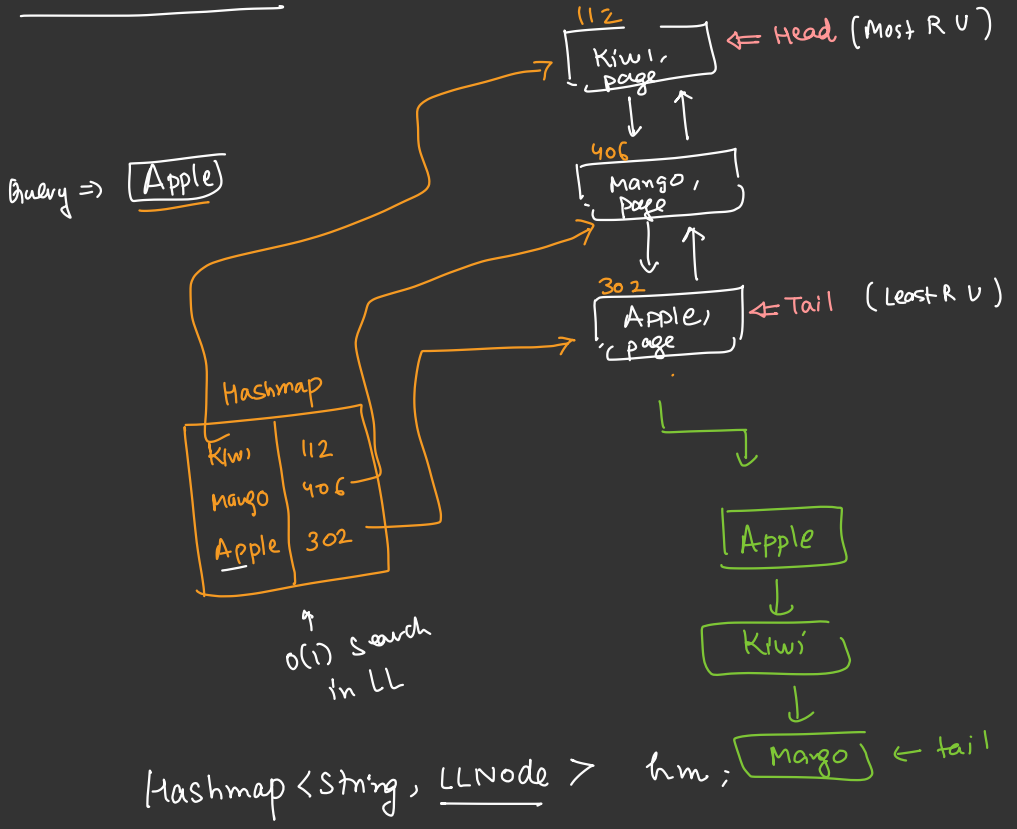
Cache

(fast Reads)

(Slow Reads)

- faster acess if same query comes in future

① Cache is not full



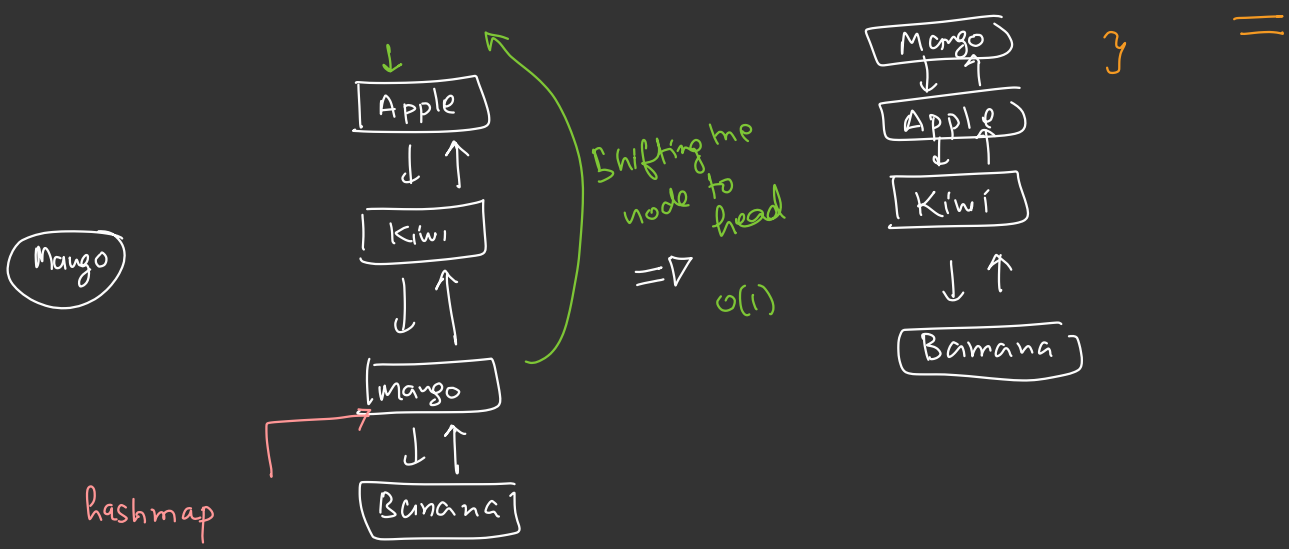
Hashmap<String, LLNode> hm;

Practical

LL Node {
String key
Page page
LLNode next,
LLNode prev,
}

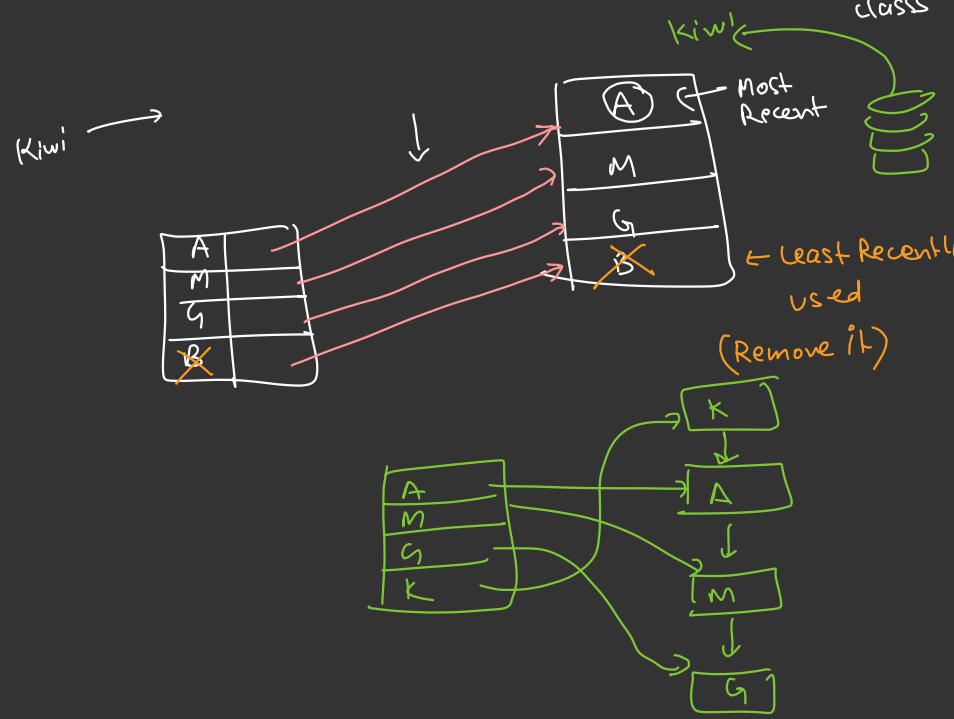
Doubly LinKed List {
LLNode head,
LLNode tail;

Shift to head (Node x)


Mango
Apple
Kiwi
mango
Banana
hashmap
Shifting the node to head
=> O(1)
Mango
Apple
Kiwi
Banana

② "Cache is full"

↳ evict (remove) the least recent used item



class LRUCache {

- maxlimit = 5
- Hashmap<String, LLNode> hm;
- Doubly Linked List dll;

faster search

order MRU

LRU

getMostRecent() {
    return dll.head;
}

3

queryString() {

    Node add = hm.get(key);
    dll.shiftToHead(add);
}

3

3

Recap of all the steps [Interviews ★ ★ ★]

1. Look for the item in the hashmap.

2. If the item is there in the hashtable, then it already there in the cache, its called "**Cache hit**"

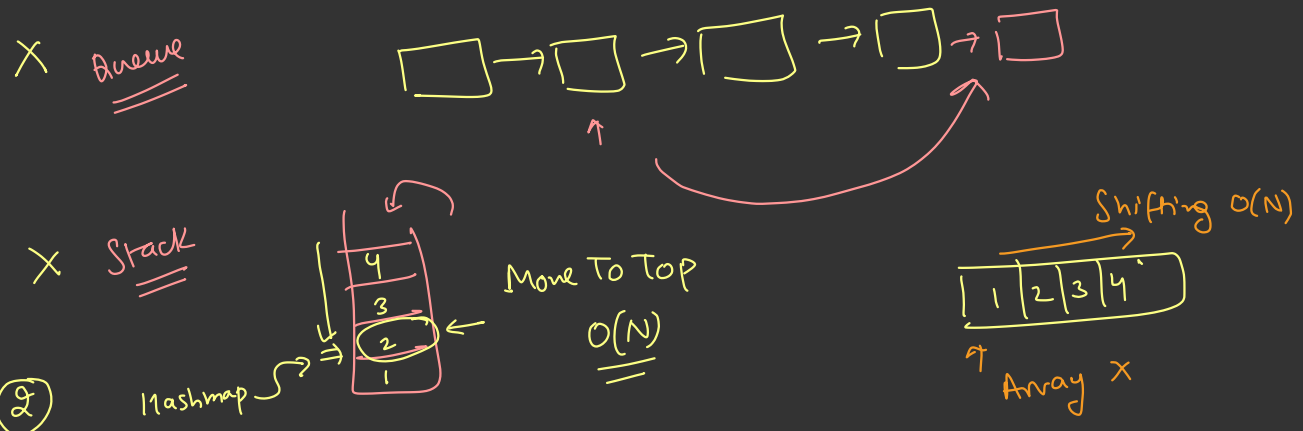
   i) Find the address/object reference of the node using hashtable. O(1)
   ii) Move the item in the linked list to the head of the linked list making the most recently used item. O(1)

3. If the item isn't the hashtable, it is of cache miss. We need to load the item into cache from permanent memory.

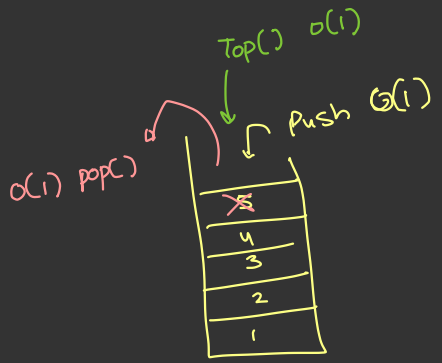
   i) If cache is **full**, use LRU strategy to remove the least recently used item.
      i) Grab the least recently used item from the tail of Double Linked List. O(1)
      ii) Remove the item at the tail, update tail to the prev node, and erase corresponding item from the hashmap. O(1)

   ii) Create a new LLNode and add it to the head of the linked list. O(1)

   iii) Update the hashmap, with key,value (string and address) of newly created node. O(1)


Queue
Stack
4
3
2
1
Move To Top
O(N)
(2)
Hashmap
Shifting O(N)
1 2 3 4
Array X

Stacks




```
isFull(){...}
isEmpty(){...}
```

10:10 Break

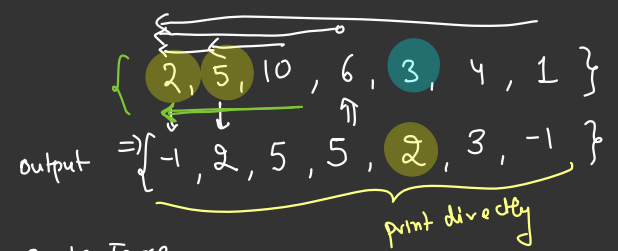
3-4 Classical problems (Tricky)

Problem(s)

Q1 Given an array of size N, find the nearest smaller number for every element that is present on left.

arr = { 1, 6, 4, 12, 3, 8 }

output = { -1, 1, 1, 4, 1, 3 }



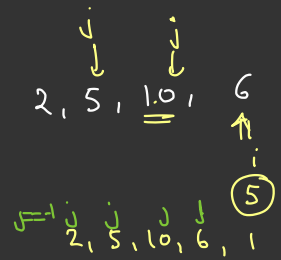
Algo-1 Brute Force

```
output = []
output[0] = -1

for ( i=1 ; i<=n-1 , i++ ) {

    for (j=i-1 ; j>=0 ; j--){
        if (arr[j] < arr[i]){
            output[i] = arr[j]; break;
    }
    if(j==-1){
        output[i] = -1
    }
```

O(N²) time

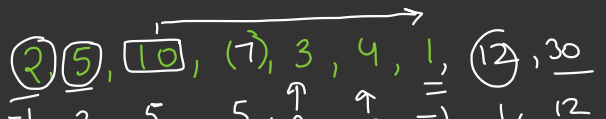
Space - O(N) → O(1) possible

3



Algo-2

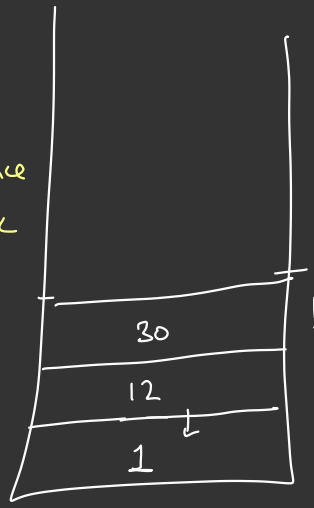
(Tricky)



O(N)

⇒ every no is pushed once into stack

and popped once from stack



3 < S.top()

10 > 5

5 > 2

10 is useless Now

O(N)

Big NO ↓ Add

⇒ Small No

↳ Remove all no's from Stack which are bigger than C.N.

① Create a Stack

② For ith element : for(i=0 ——— n-1)

```
while (!s.empty() && s.top() >= arr(i)){
        s.pop();
    }
```

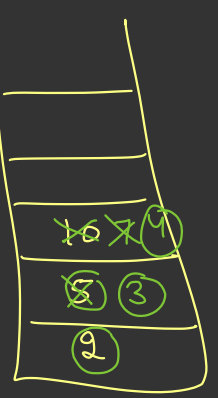
```
if(s.isEmpty()){
        output[i] = -1
    }
else{
        output[i] = s.top();
    }
```

⇒ s.push(arr[i]);

I/O

2, 5, 10, 7, 3, 4

-1, 2, 5, 5, 2, 3



Stock Span Problem

Price of a stock for N days

	D1	D2	D3	D4	D5	D6	D7
Stock[] = {	100,	80,	60,	70,	60,	75,	85 }
Span = {	1,	1,	1,	2,	1,	4,	6 }

Brute force

2 loops

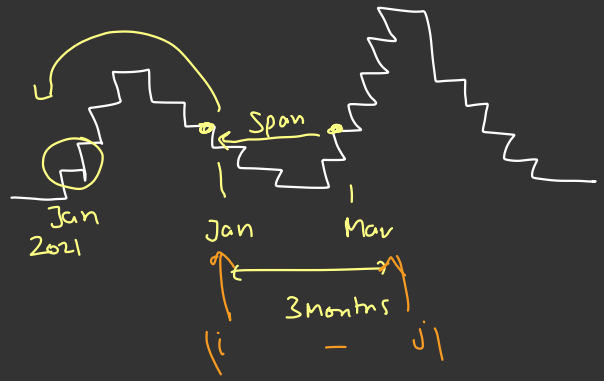
a[i]

Less = than a[i]

$O(N^2)$

```
for (i=1 ——— i<n ; i++) {
    Span[i] = 1
    for (j= i-1 ; j>=0 ; j--) {
        if (stock[j] > stock[i]) {
            break
        }
        Span[i]++
    }
}
```

Algo-2 Using stack



D1 D2 D3 D4 D5 D6 D7

Stock[] = { 100, 80, 60, 70, 60, 75, 85 }

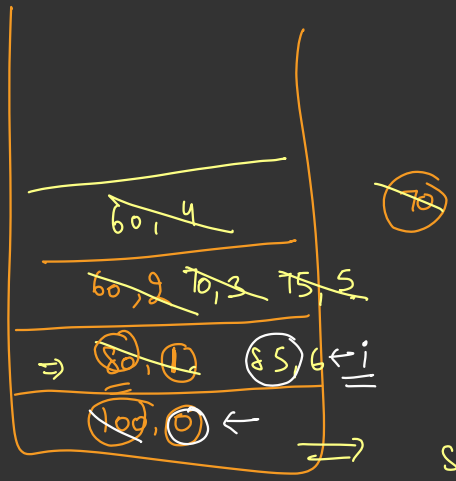
Span = { 1, 1, 1, 2, 1, 4, 6 }

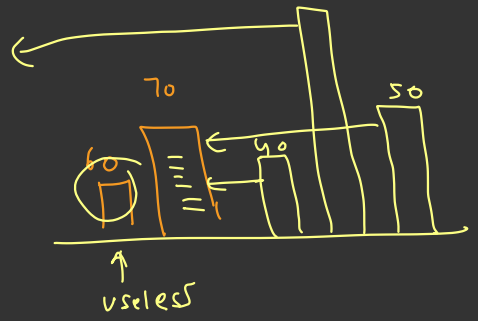
D1 D2 D3 D4 D5 D6 D7

Stock[] = { 100, 80, 60, 70, 60, 75, 85 }

Span = { 1, 1, 1, 2, 1, 4, 6 }

Day	0	1	2	3	4	5	6
Stock Value	100	80	60	70	60	75	85
(1)		1-0	2-1	3-1	4-3	5-1	6-0
		=1	=1	=2	=1	=4	=6





Less than Top
↳ Push it

Bigger
( ) Pop all small

Stack < PairOfInt > s.

Stack < int > s,
↳ idx

data → arr(i)

Code -

Index

```
Stack<Integer> s = new Stack<Integer>();
```

```
Span[0] = 1
s.push(0);
for (i=1 ——— n-1) {

    while(!s.empty() and stock[i] >= Stock[s.top()]) {
        s.pop();
    }

    Span[i] = i - st.top()   if s. is not empty
            = i + 1          else (s is empty)

    s.push(i);
```