1→2→3 → 3→2→1→ NULL.

→ 1→2→3→NULL

3→2→1→ NULL ⇒ → 1→2→3→NULL.

1→2→3→3→2→1→ NULL.

1→2→3→3→2→1→ NULL

[fast→next→next != NULL.] even.

[fast→next != NULL] odd.

head mid head2

1→2→3 → 3→2→1→ NULL.

↳ NULL

head2 = mid→next

mid→next = NULL.

mid

1→2→3→4 3→2→1 → NULL

1→2→3→4→ NULL.

1→2→3→NULL.

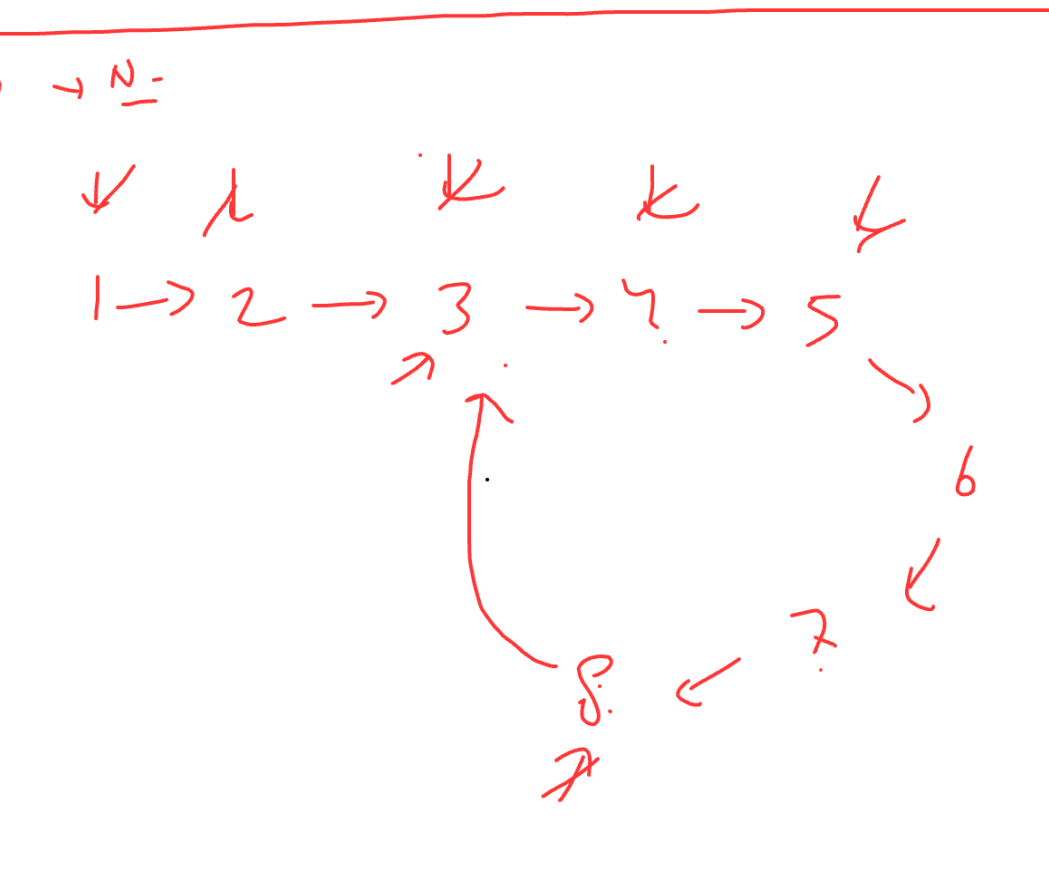
1→N

N.

1→2→N

1→3→N

1→2→3→N.



visited
1
2
3
4
5
6
7
8

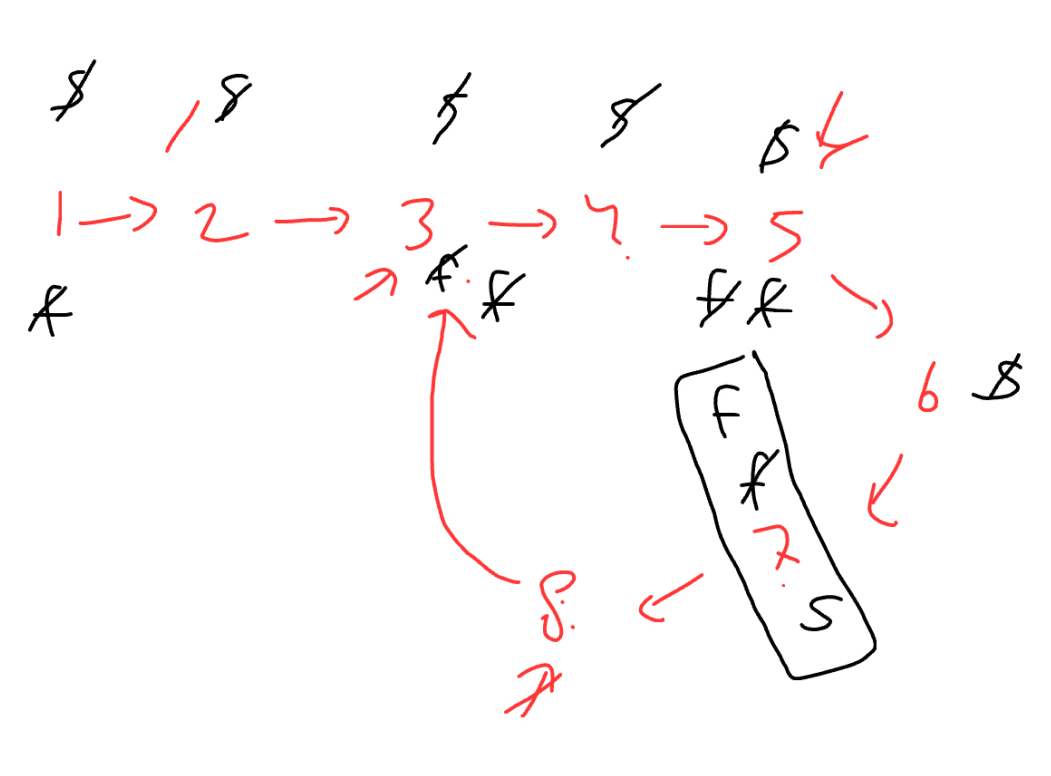
TC: O(n)

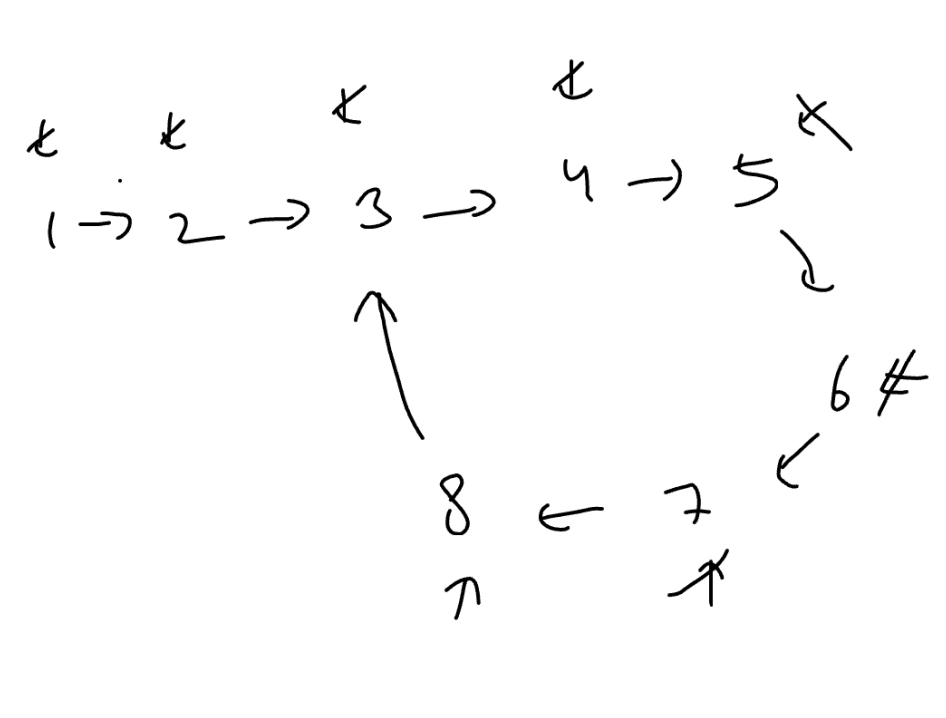
AS: O(n)

	t	t-1
slow:	x	x-1
fast:	x+1	(x+1)-2 = x-1

50

100.



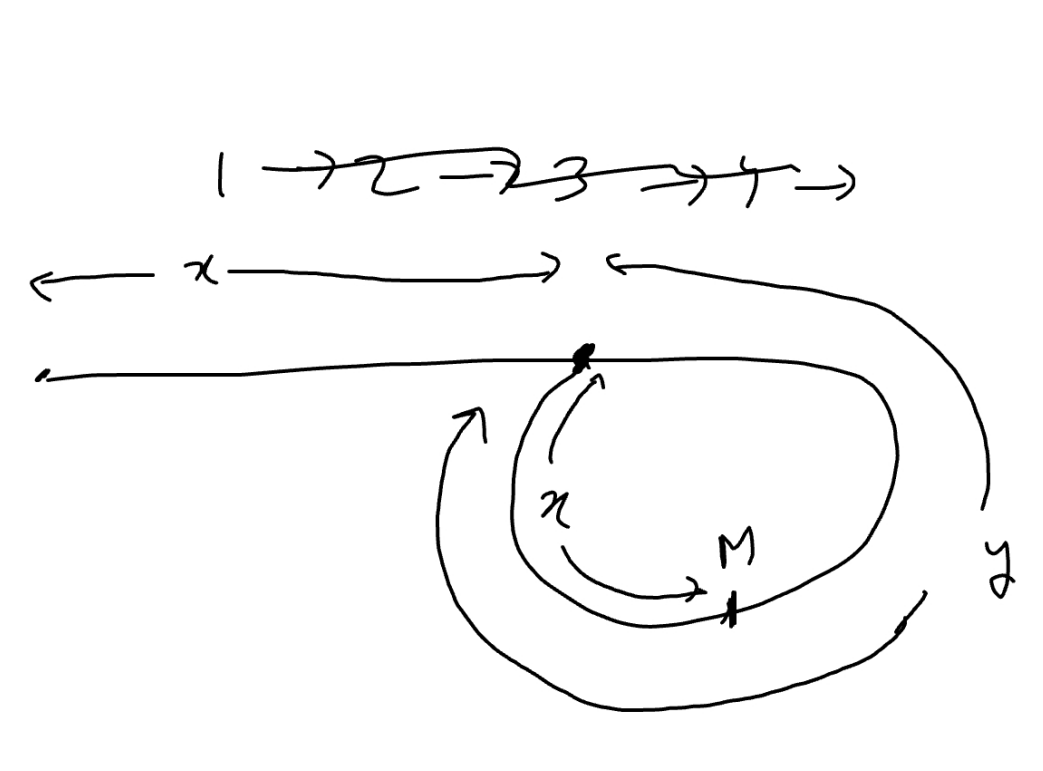


visited
1
2
→3
4
5
6
7

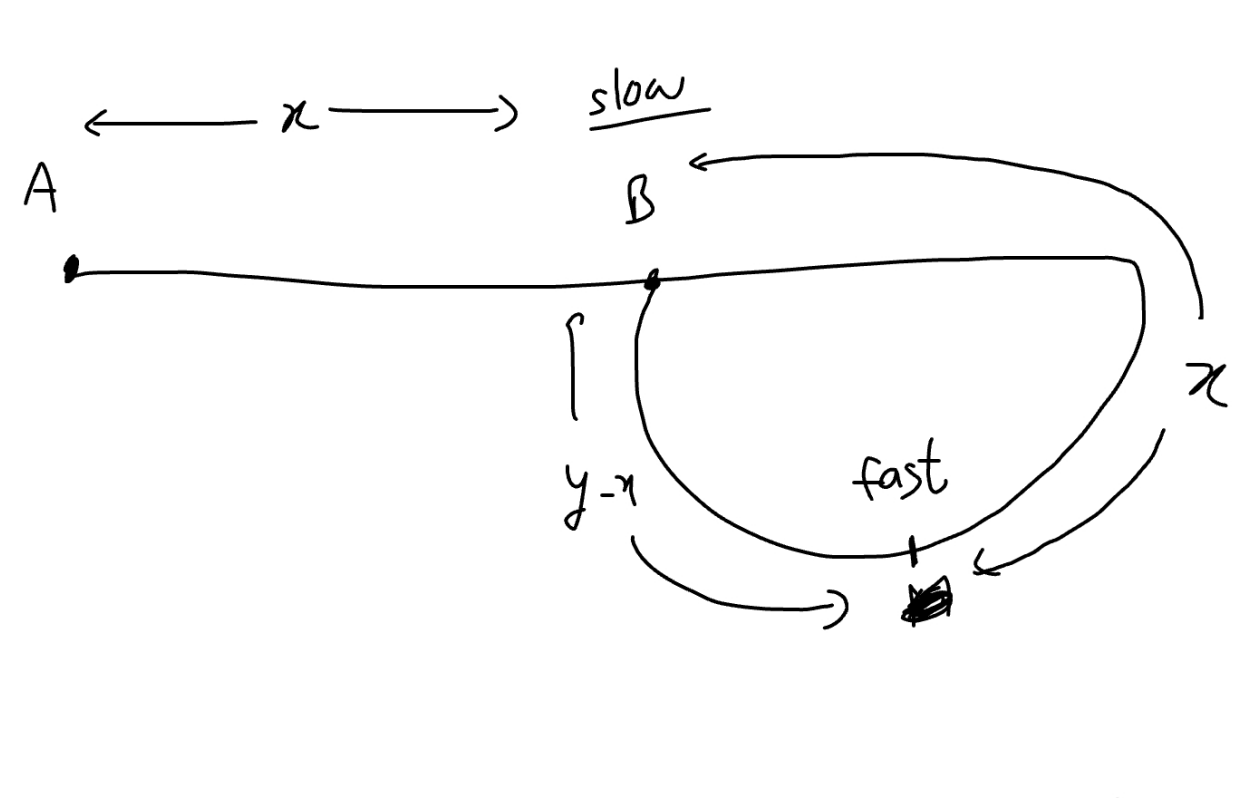
[TC: O(n)

AS: O(n)].

Floyd's Cycle Detection Algo.



t.



slow is x steps behind fast

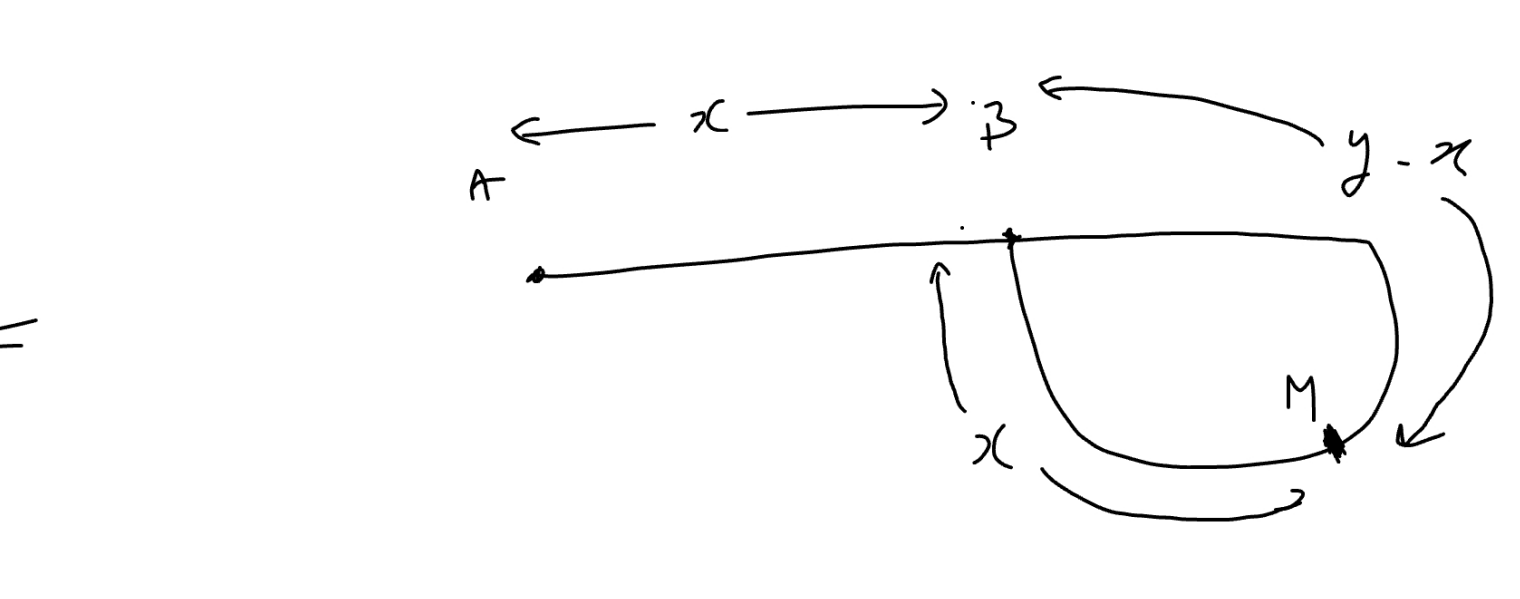
fast is (y-x) steps behind slow.

⇒ fast catches up slow at the rate of 1 node per unit time.

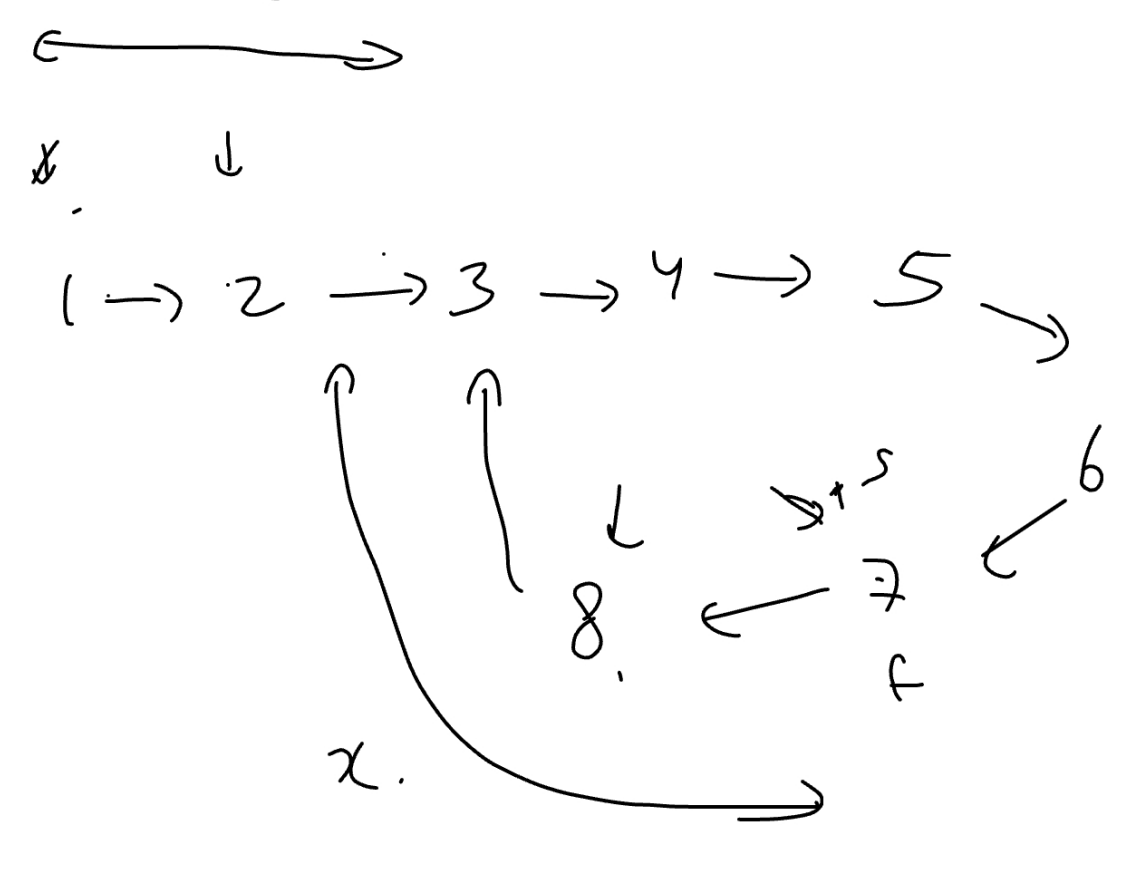
⇒ How much time does it take for fast to catch slow?

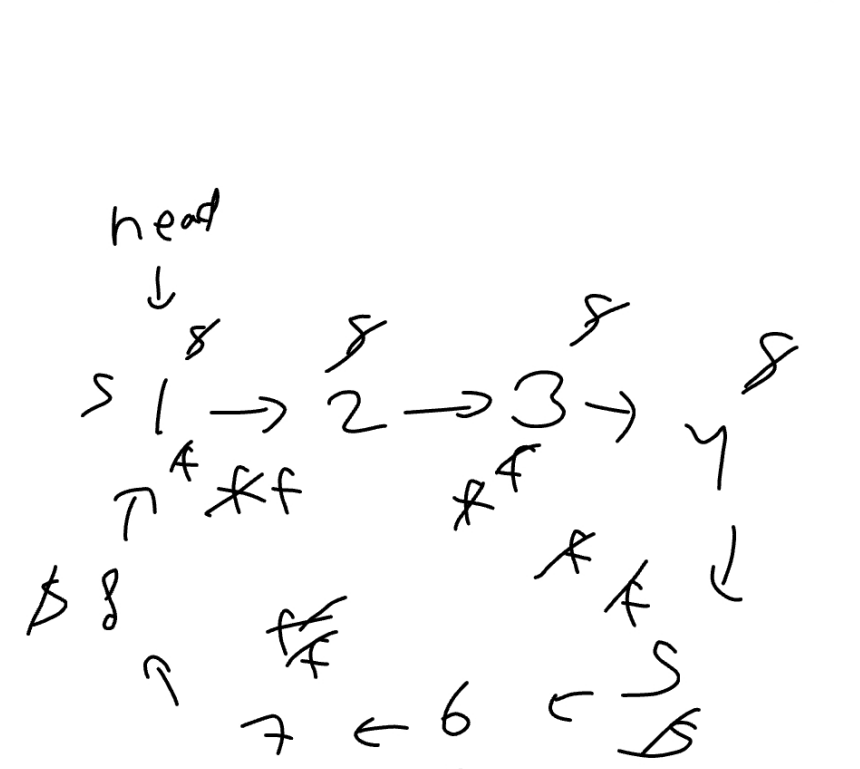
⇒ (y-x) units of time.

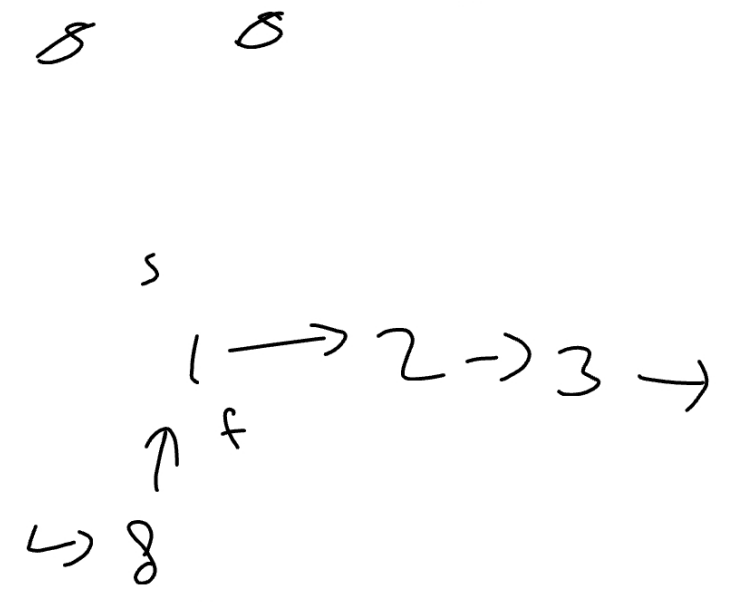
t+(y-x)

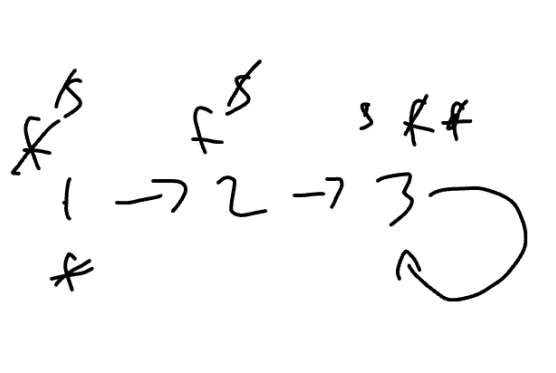


y-(y-x) = x









STACK

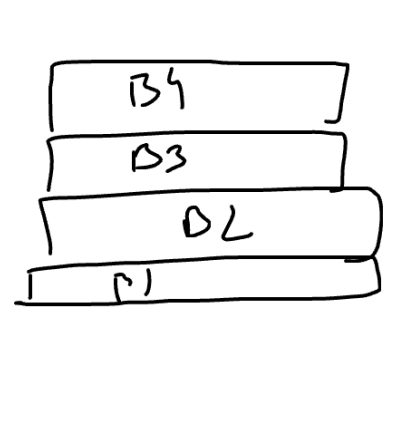
LIFO → Last In First Out.

B4

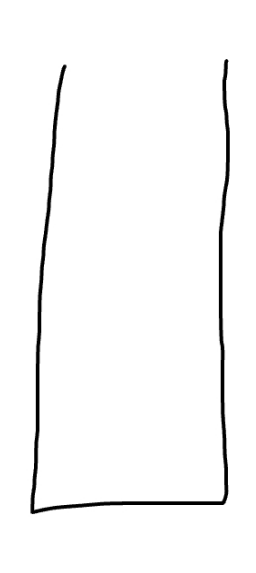
B3

B2

B1



B4, B3, B2, B1.



Implement Stack

Using Arrays

stack

push

pop

top

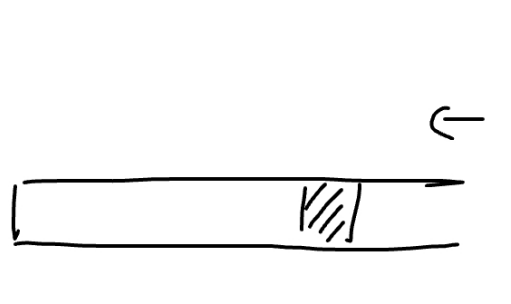
size.

Dynamic Array:

Insert at end.

Pop from the end.

Return last elem.



Static Array

Insert at size-1, size++

size--

arr[size-1].

Using LL.

Stack

Push

Pop

top

LL

Insert at Beginning.

Delete from Beginning.

return head.

size: ~~2~~ ~~4~~ ~~5~~ 3 2 3

0	1	2	3		
1	2	4 3	10		— — —

1→2→4→8→16→32→64 . . . . . .

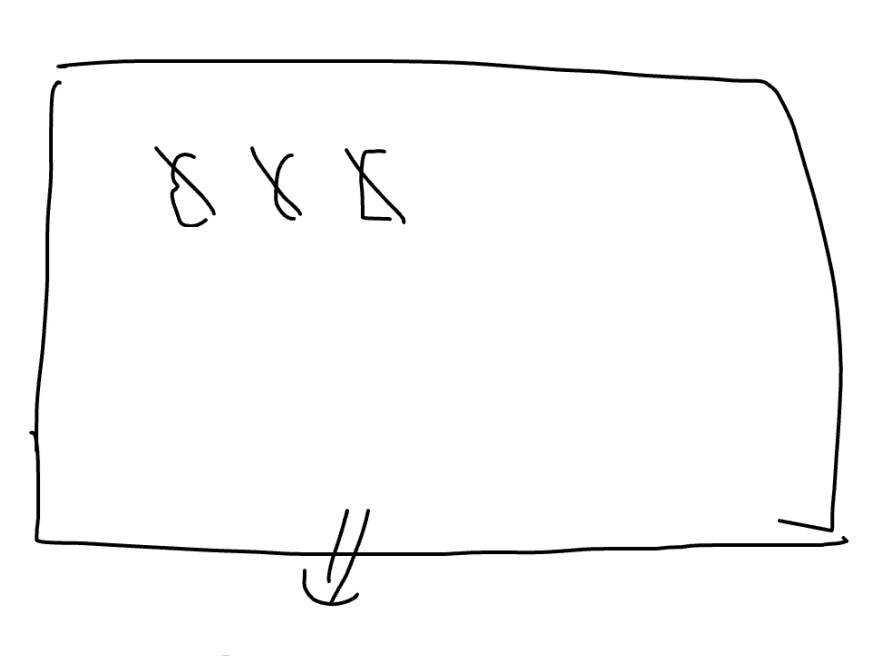
([ )] → False.

([ ]) → True.

( ) [ ] → True.

[(([ ])]

DS ??



[ ] ({ })

X

X

X

[( ])

(⇒ False.

[

TC: O(n)

AS: O(n).

)

"))"

( ) [

[

X

---

arr: [6, 8, 0, 1, 3]

NGE: [8, -1, 1, 3, -1]

[6, 8, 0, 3, 1]

[8, -1, 3, -1, -1].

i: 0 → n-1

j: i+1 → n-1

TC: O(n^2)

AS: O(1).

arr: [6, 8, 0, 10].

NGE: [8, 10, 10, -1].

[5, 4, 3, 2, 1, 10].

[10, 10, 10, 10, 10, -1]

[~~6, 8, 0, 3, 1~~].

[5, 3, 6, 4, 8, 2, 3, 10, 9].

[6 6 10 10 3 10 -1 -1].

8 10, ~~3~~, ~~8~~ ~~4~~ ~~6~~ ~~3~~ ~~5~~

TC: O(n)

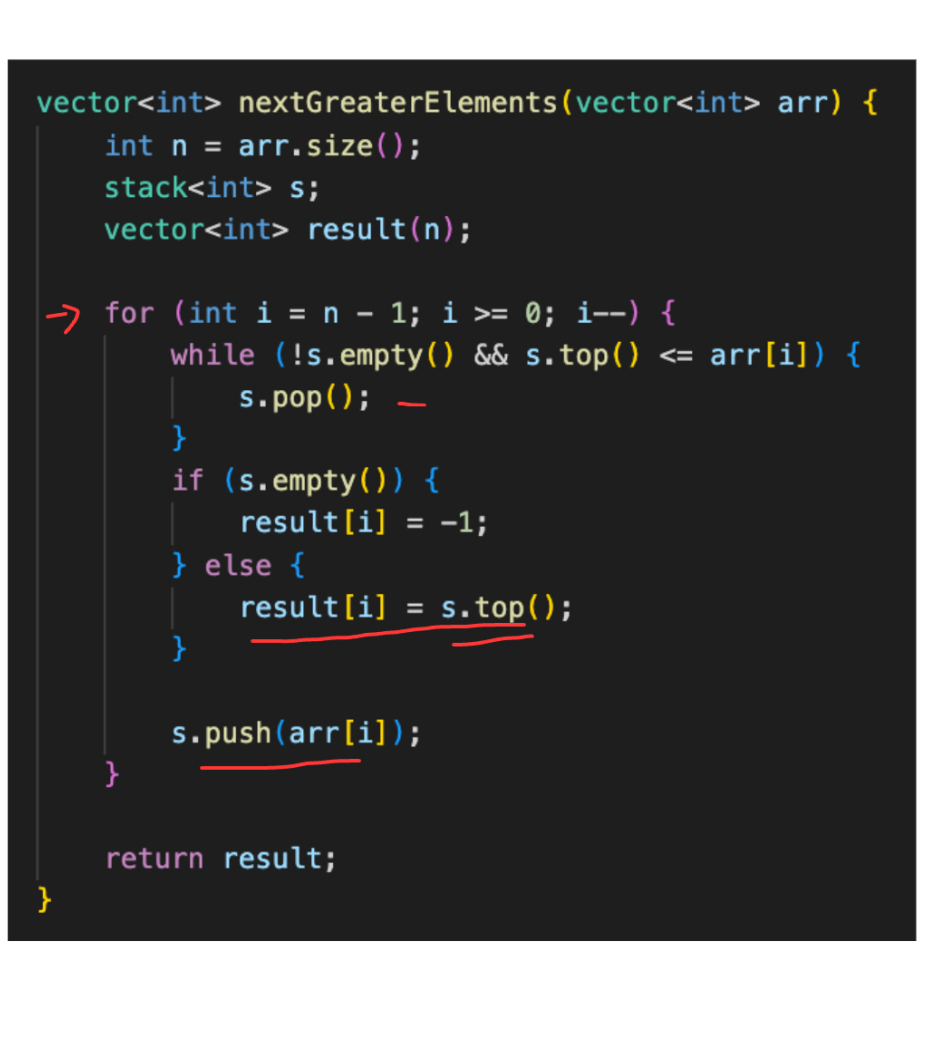
AS: O(n).

[1, 2, 3, 4, 5].

5, 4, 3, 2, 1.

[5, 3, 6, 4, 2, 3, 9, 10].

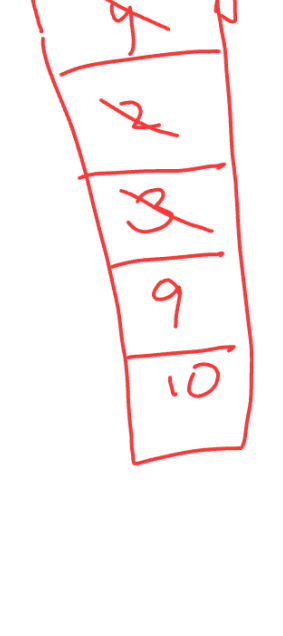
[6 6 9 9 3 9 10 -1]



```
vector<int> nextGreaterElements(vector<int> arr) {
    int n = arr.size();
    stack<int> s;
    vector<int> result(n);

    for (int i = n - 1; i >= 0; i--) {
        while (!s.empty() && s.top() <= arr[i]) {
            s.pop();
        }
        if (s.empty()) {
            result[i] = -1;
        } else {
            result[i] = s.top();
        }

        s.push(arr[i]);
    }

    return result;
}
```