* First | Last | Total Using Binary Search: →

arr = { 1, 2, 3, 3, 3, 4, 5, 6 }   k = 3
(indices 0 1 2 3 4 5 6 7)

$f_0 = 2$, $l_0 = 4$
{ to = l − f + 1
4 − 2 + 1
= ③ }

mid = (s+e)/2 = 7/2 = 3

⇒ (F/O) Cases:

```
ans = -1;
if (arr[mid] == k){
    ans = mid;
    e = mid-1;        // extreme left
}else if (arr[mid] < key){
    s = mid+1;
}else {
    e = mid - 1;
}
```

(L/O) Cases:

mid = (s+e)/2 = 7/2 = 3

```
if (arr[mid] == k){
    ans = mid;
    s = mid+1;        // extreme right
}else if (arr[mid] < key){
    s = mid +1;
}else {
    e = mid +1;
}
```

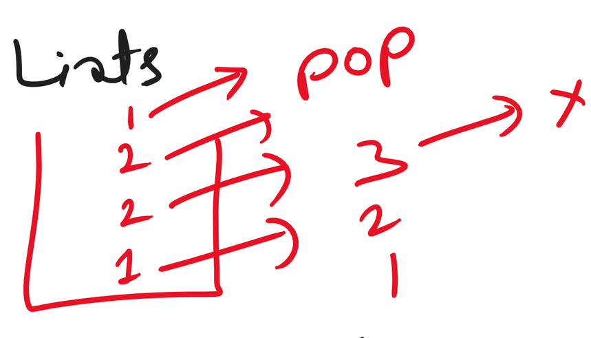
* Jagged Array → 2D Array with unequal rows.

Data Structures: →

↓

Linear

Array → 1D, 2D, MD
Stacks LIFO (Reverse)
Queues
Linked Lists

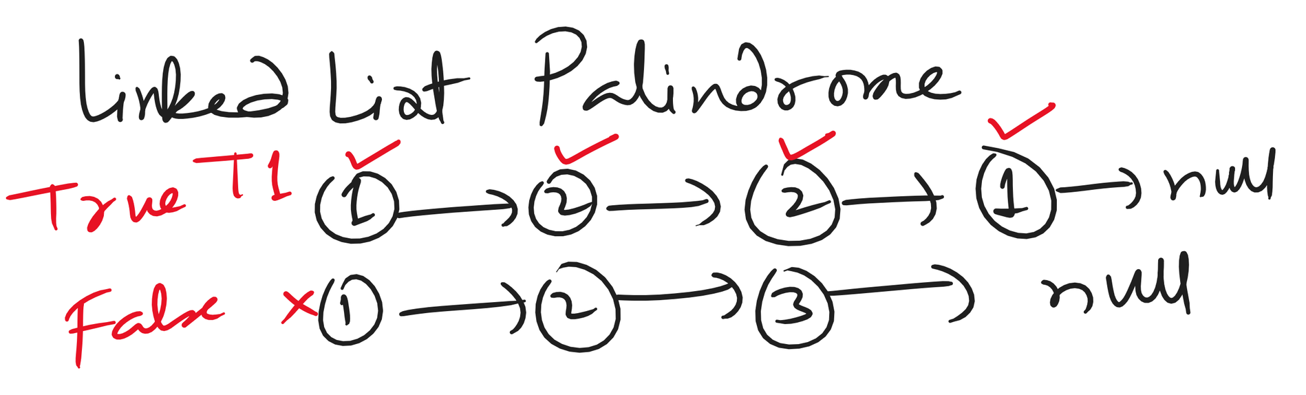


Balanced Parentheses

↓

Non - Linear

Linked List Palindrome

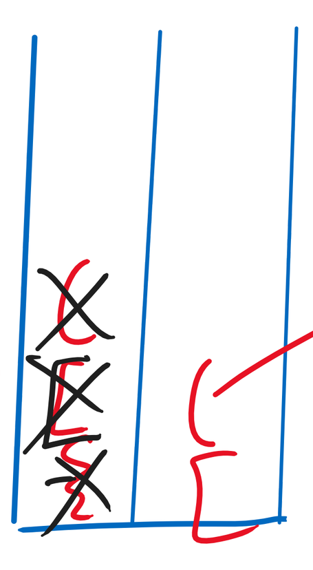


String s1 → " { [( )] } "   s2 " [( ] }] { "

if s[i] == { or [or (
  s.push s[i]

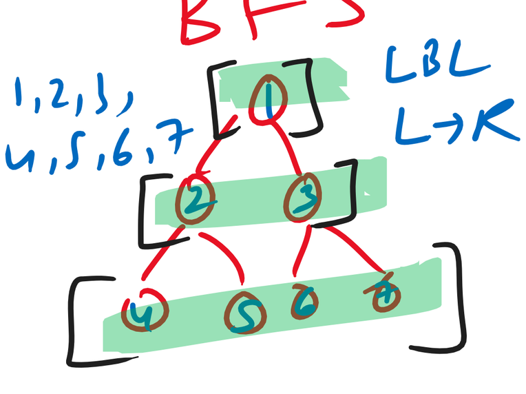
if s.top == (, [, {
  s[i] ==) ,] , }
s.pop()

empty (true)



s.top == (false

(LOT) Tree
BFS



(Reverse) ⇒ Stack
(LOT) BFS queue

* Given an integer N → Generate Binary Numbers 1 to N in this exact given format →

N = 5   1, 10, 11, 100, 101

Initial Queue: queue<string> = "1";   queue = ["1"]

Step1: pop "1" push "10" & "11"   queue = [10, 11]

Step2: pop "10" push "100" & "101"   queue = [11, 100, 101]

Step3: pop "11" push "110" & "111"   queue = [100, 101, 110, 111]

Step 4: pop "100" push "1000" & "1001"   queue = [101, 110, 111, 1000, 1001]

Step 5: pop "101" push "1010" & "1011"

Queue {
Adding Elements: → add( ) | offer( )
Removing Elements: → remove( ) | poll( )

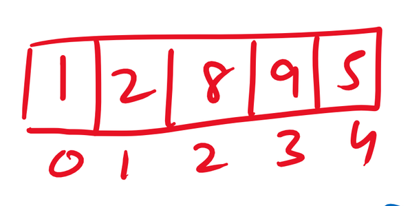
add( ) / remove( ): strict & throw exceptions
* If strictly handles capacity

offer( ) / poll( ): not strict & ignore if fails.

Wild Cards   <?>   Generics ***
templates ***

Arrays (Access)

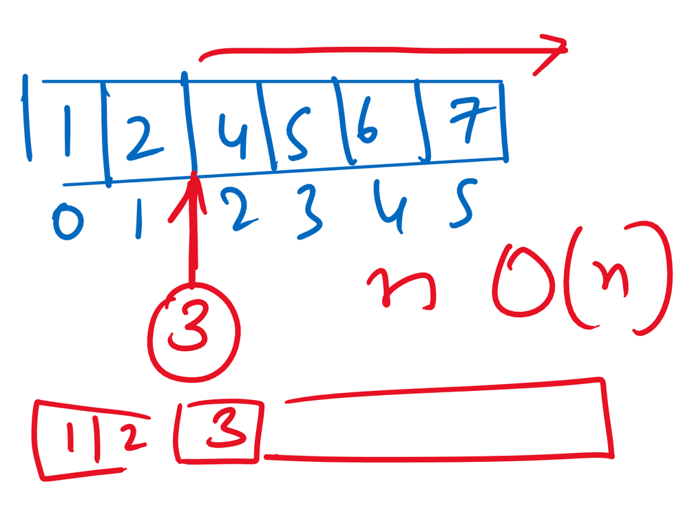
(Search)



3rd element = arr[2]
index   O(1)

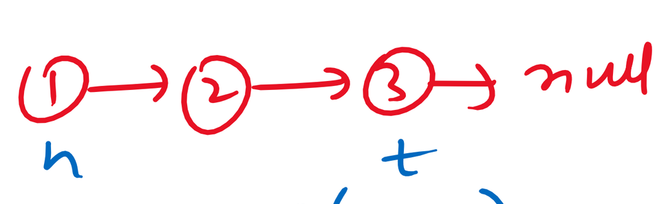
(Insert)

Slow O(n)



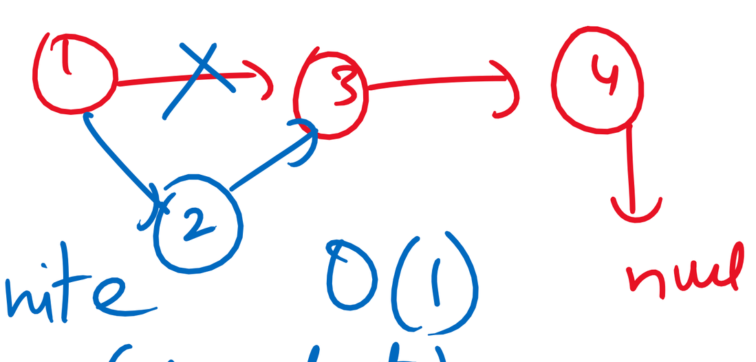
Linked Lists

(Search)
3rd node list



O(n)

(Insert)



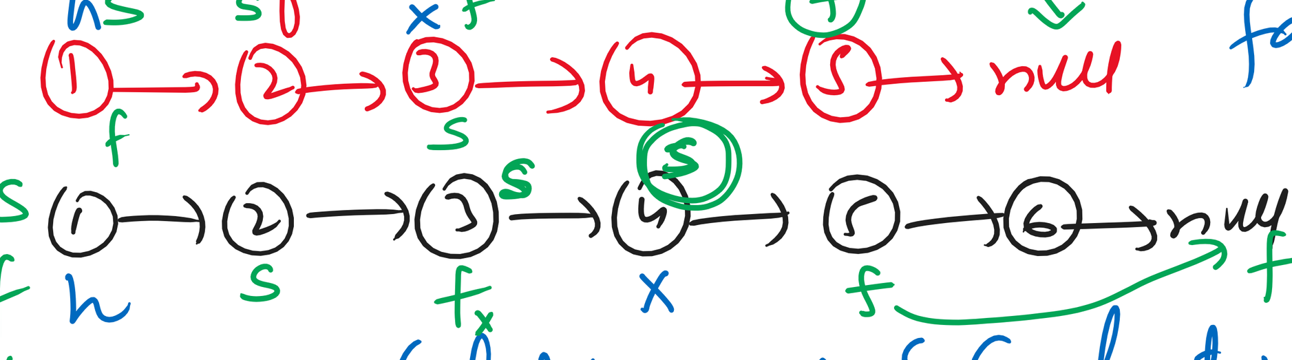
finite O(1) (Constant)

Linked List Important Questions:

(Floyd's Algo)

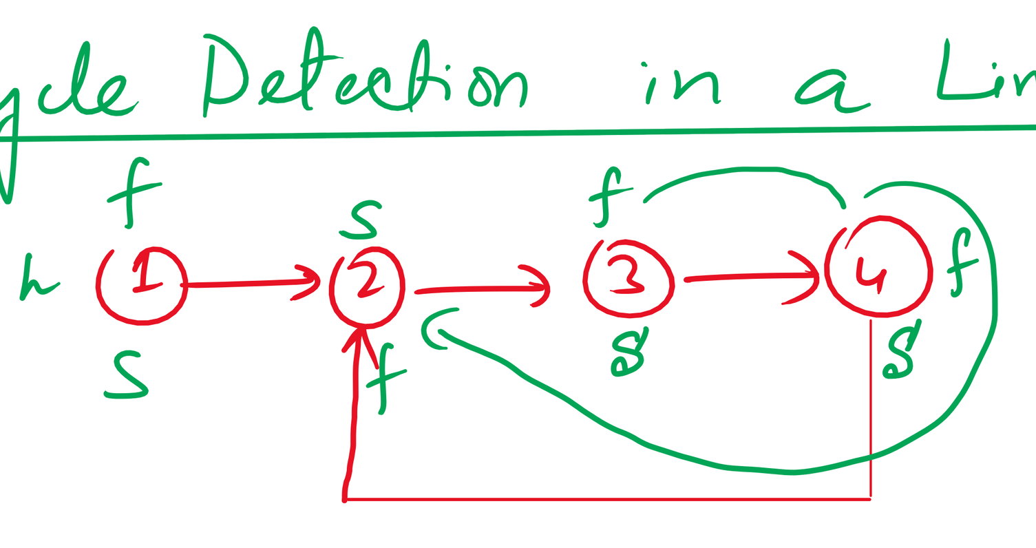
slow = head;
fast = head;

* Middle of a linked list




```
while (fast != null && fast.next != null) {
    slow = slow.next;         → 1step
    fast = fast.next. next;   → 2steps
}
return slow;
```

Cycle Detection in a Linked List: →



four → next = two;

```
fast=head;
slow=head;
while (fast != null &&
       fast→next != null)
{
    slow = slow.next;
    fast = fast.next.next;
    if (slow == fast){
        true
    }
}
```

* { HashMap + Hashtable + Hashing }
Practical Example

C++ (template) (typename)
(30 − 40) LeetCode