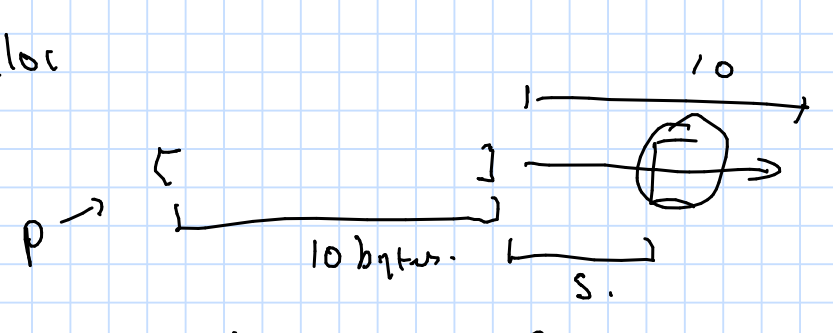
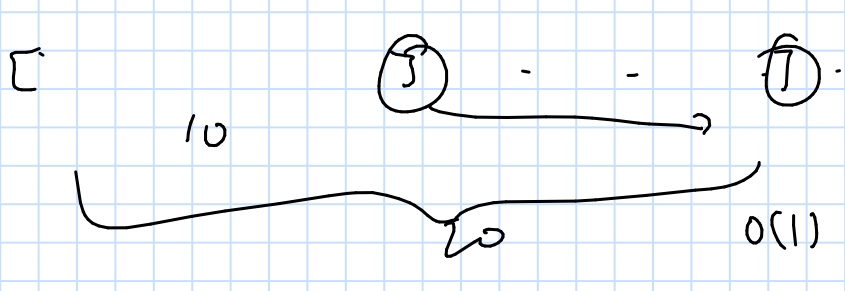
1) realloc



realloc(... 20)

→ needs to find new memory.
-) copy.
↓
O(n)



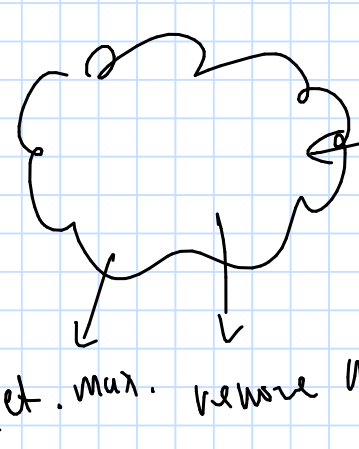
List ADT

1) get_at_index (L,i)
2) remove_at_index (L,i).
3) insert_at_index (L,i,k)

Array [ ]

LL →•→•→

Priority queue.



[6 4, 3, 2, 1, 0]

insert (6)

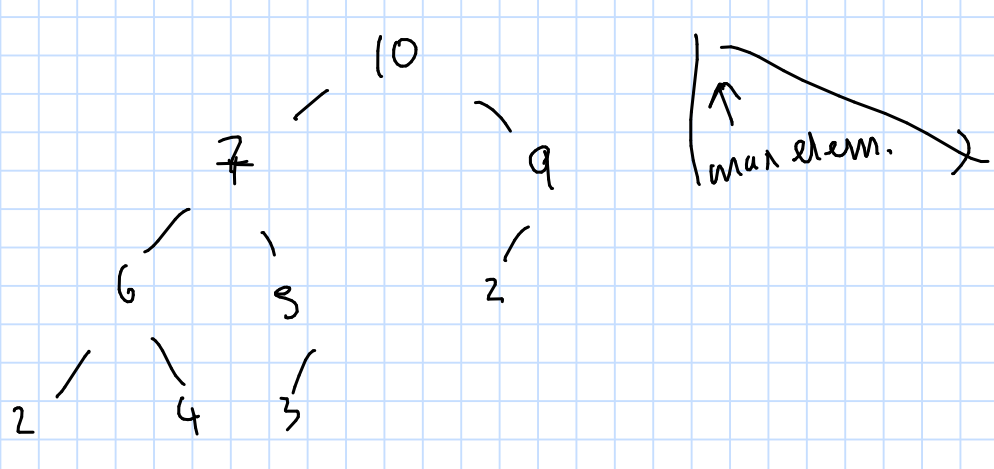
insert (0)

O(n)

heap

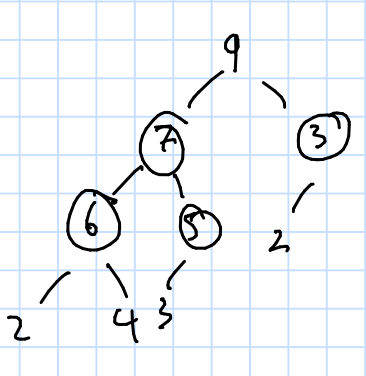
heap invariant ~~A~~ max heap

∀ node n, children of n are ≤ n



struct node {
int val ;
struct node* left ;
struct node* right ;
}

[Ø 9 7 3 6 5]
0 1 2 3 4 5 6 7 8
↑ root



Left (i) = 2i

Right (i) = 2i+1

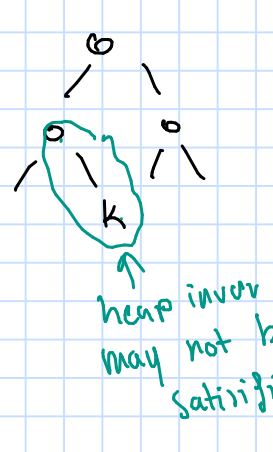
left(2) = 4

right(2) = 2(2)+1 = 5.

insert (k)

1) put k @ end.

2) heapify-up.

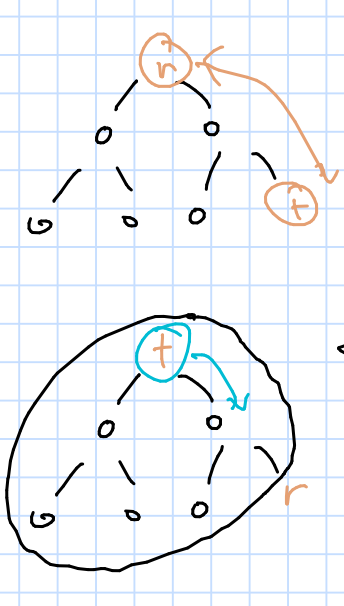


valid bin heap.

↓ on insertion we turn it into an invalid heap.

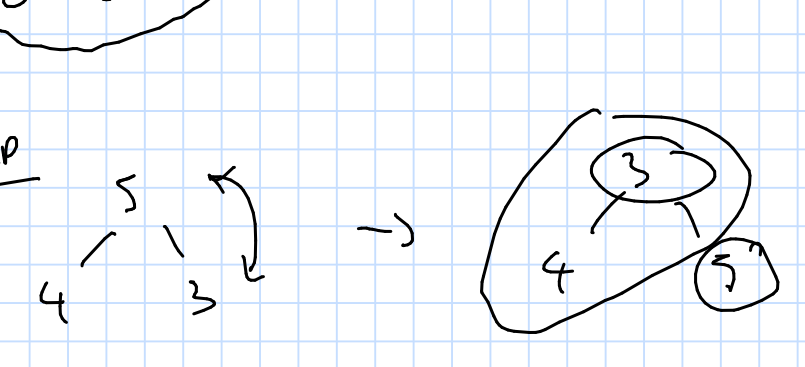
↓ goal is to turn it back into a valid heap.

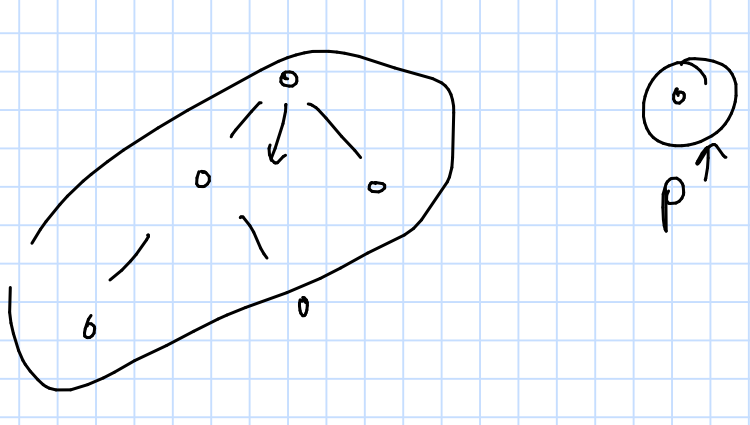
extract_min

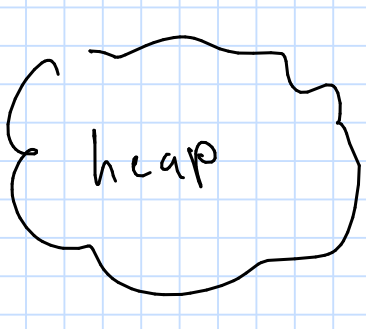


1) swap root w/ tail
2) decrement heap size
3) restore heap property.
↓ heapify-down.

max heap







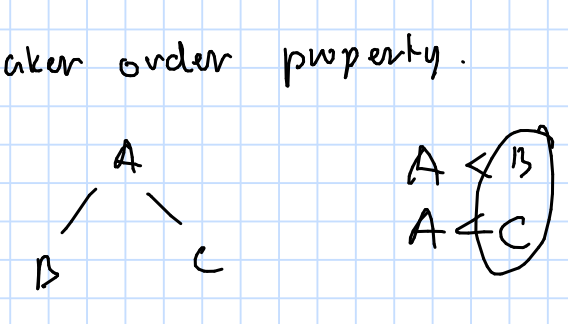
before all calls: valid
during call → invalid
↓
always return to being valid

Building heaps

1) insert(x) n times } → O(n log n)
↑ log(n)

2) O(n)? sorting is n log n

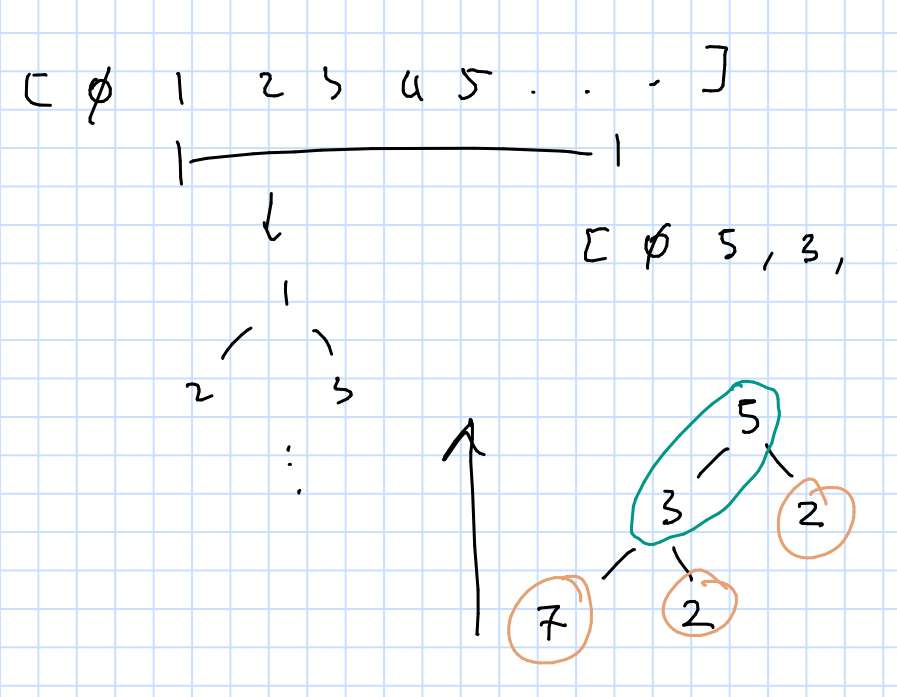
heap has a weaker order property.



A < B
A < C

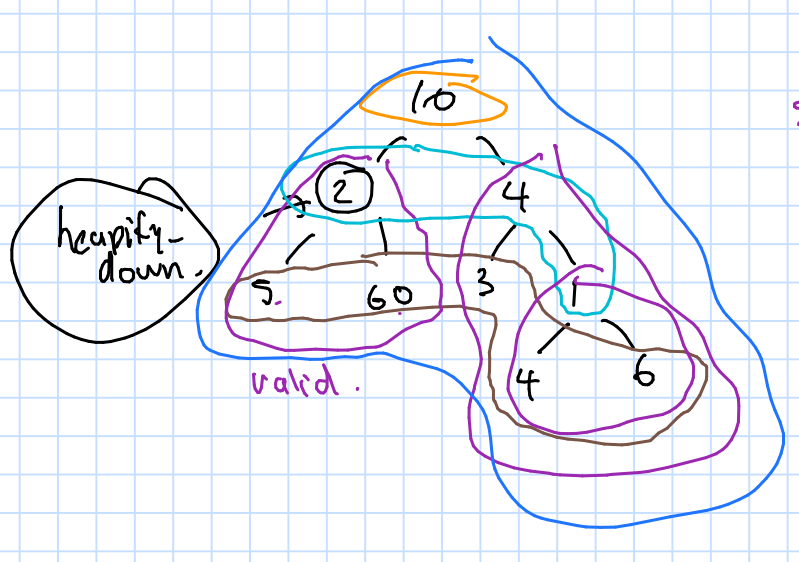
B ? C

heapify (A) turns an array into a heap



min heap

leafs are already heaps.

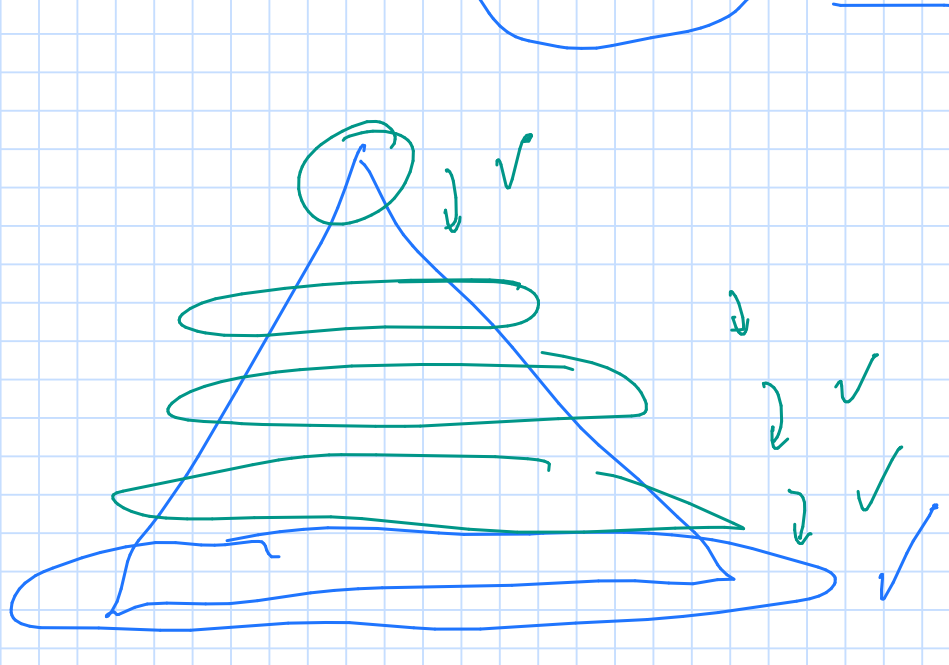


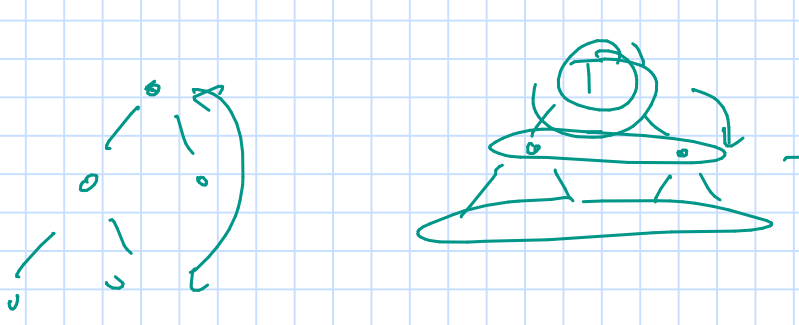
starting @ 1st internal node (parent of leaf)

call heapify down.

→ turns into valid sub heap.

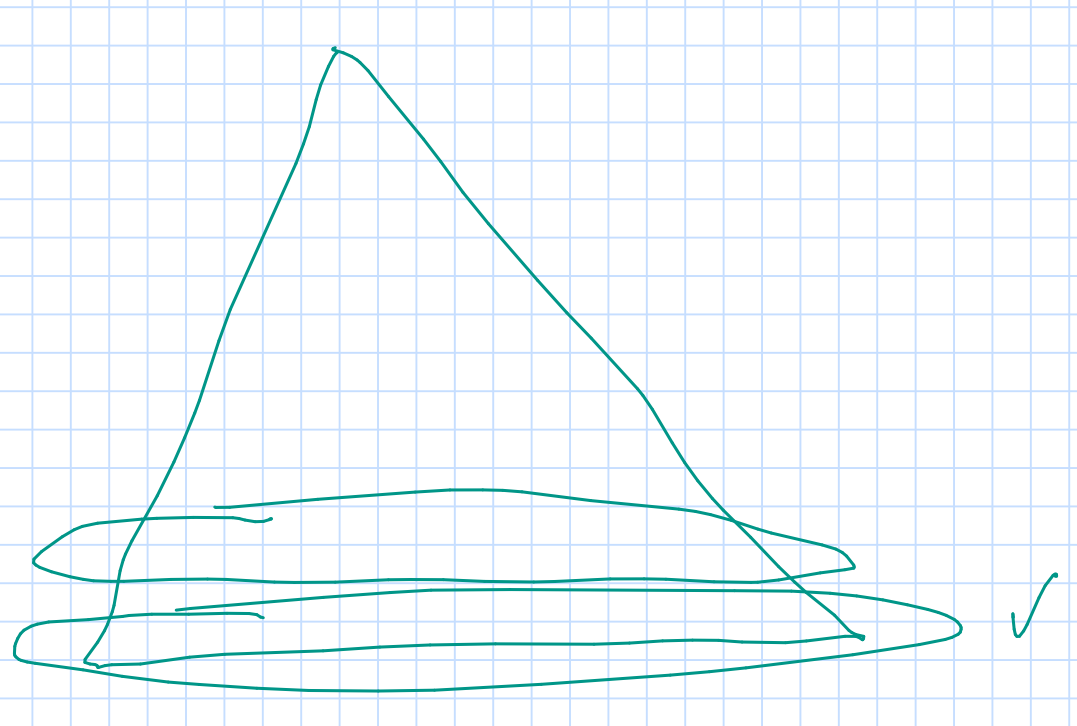
← valid heap.



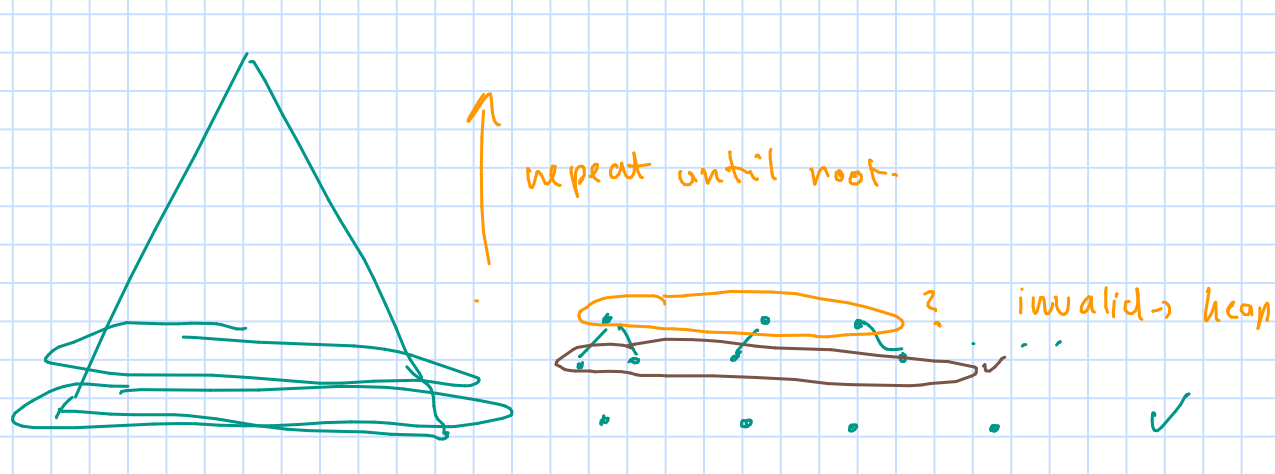


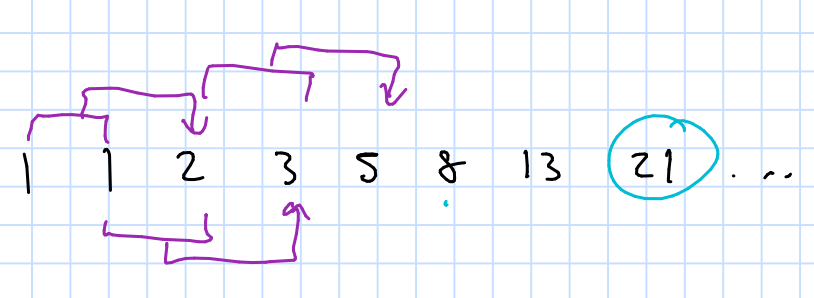
restore heap invar

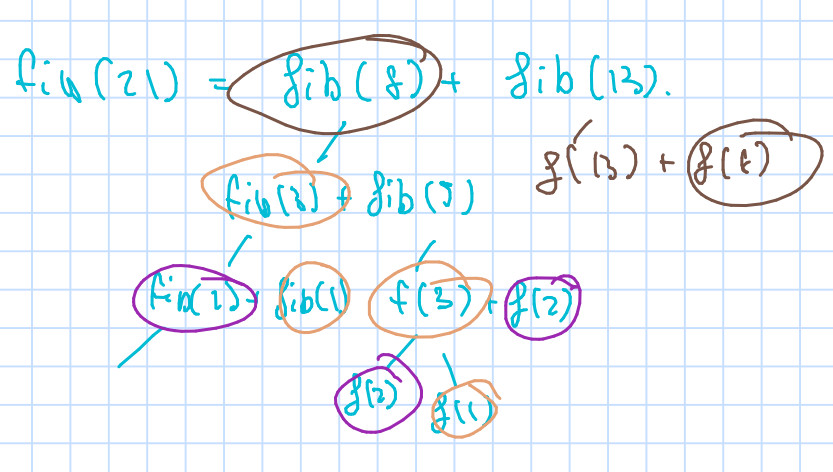
assumes heaps below are valid.



✓ size 1 ⇔ valid.







memoization

dictionary to store answers.

Fib(5) → D[5] = fib(5).

fib(13) → fib(5) + fib(8)
↙
D[5]

~~A~~ map in → out of function by storing the prev. result.

bottom-up

fib(13)
fib(8) fib(5).

↓ → finds answer + caches.

fib (13).

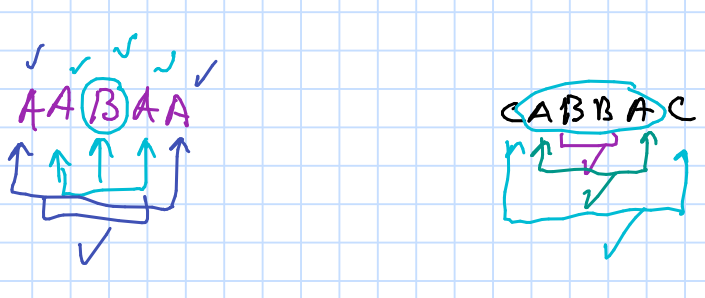
1 1 → (13)
0 1 . . . 13

1 1 2 3 5 8
0 1 2 3 4 5 6 7 8

Longest Palindromic Subsequence

S = " AABAC "
↑ ↑
i j

→ try all substrings S[i,j]. O(n²)
→ validate & keep track of longest. O(n)
O(n³)



s = " cbabcdde "

Base cases : length 1 → valid palindrome.
length 2 → valid if both are equal.
length > 2 → valid if internal is palindrome.

s = " cbabcdde "

→ j ; i ↓

i = 0, j = 2
1 1

i		0 c	1 b	2 a	3 b	4 c	5 d	6 d	7 e
0	c	✓	x	x	x	✓	x	x	x
1	b		✓	x	✓	x	x	x	x
2	a			✓	x	x	x	x	x
3	b				✓	x	x	x	x
4	c					✓	x	x	x
5	d						✓	✓	x
6	d							✓	x
7	e								✓

0, 4 → Δ = 4 ⇐ cbabc
1, 3 → Δ = 2
5, 6 → Δ = 1

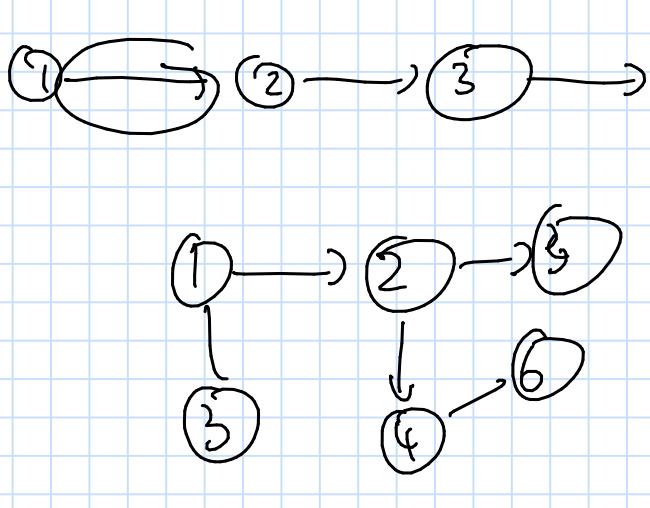
DP[i][j] is true if S[i:j] is a palindrome

length > 2 w.r.t. indices?

DP[i][j] = DP[i+1][j-1] and S[i] = S[j]

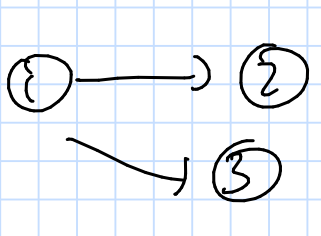
S[i,j] is a palindrome if inner is a palindrome AND the ends are the same

LL



struct node {
int val
struct node* neighbours.
}.

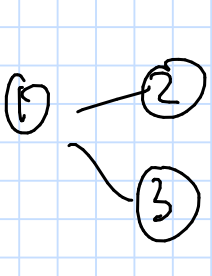
struct node {
int val
struct node* children
}.



1 → children 2, 3

2 + no children

DFS, BFS...



1 2, 3
2 1
3 1