- Recursion-3 (26 June) — ▶ Recording
  - ↳ Problem, Assignment Doubts

⇒ Linked List (Introduction) ↑ understand

+ More in Advanced Batch

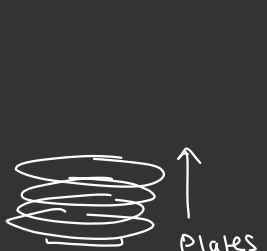
Stacks & Queues

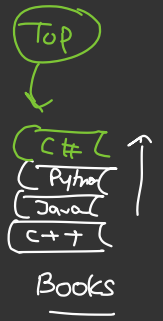
Stack

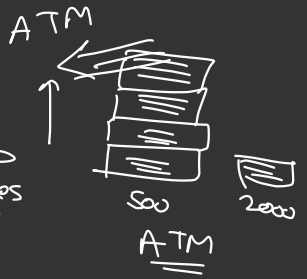
Data Structure / Container

data is stored in LIFO fashion

Last In First Out







Push / Insert() → from Top O(1)

→ Pop / Remove() → from Top O(1)

Top() → get the value from Top most object

Read

Remove the top most element.

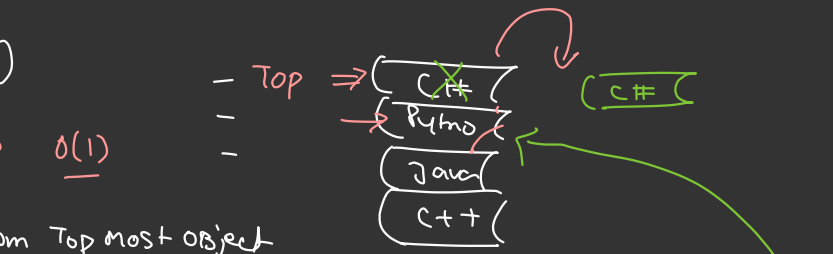
Remove a particular element

↓

Not supported directly

↓

Loop → O(N)



→ insert('C#') ✓

→ pop() ✗

→ top() Python

→ pop()

→ top() → Java

Real Life in Software

① Recursion (Implicit Stack)

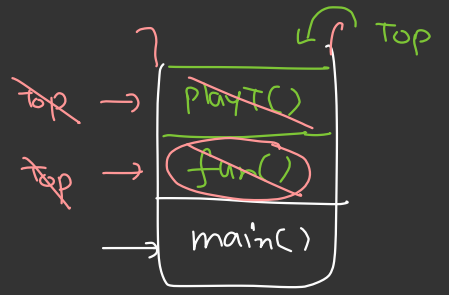
[Function Call]

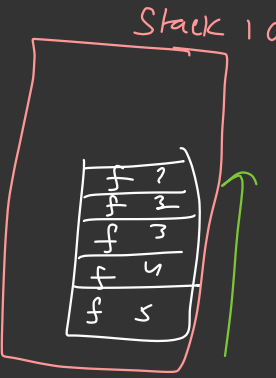
Automatically created in Memory.

```
fn(){
    fn()
}
```

playTennis() when ~~rec~~ function calls

```
playT(){
    fun(){ ≡ }
    main(){
        ≡ fun()
    }
}
```

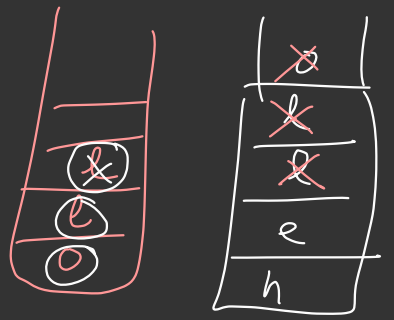




10MB, 20MB, ----,

→ ∞ Recursion
→ very large size containers

Stack Overflow



Ctrl + z

Cmd + Shift + z

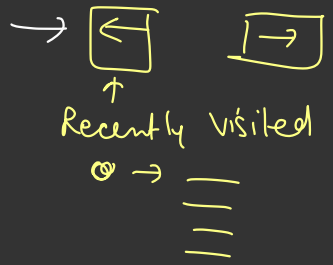
Some limit

↑

Steps we can undo & Redo

Hello

⑥ Web Browsers



Stack

open_web_page

stack.push(weburl),

Stack of Integers

push(40)
push(20)
push(8)
push(9)

→ pop() → Removed = 9

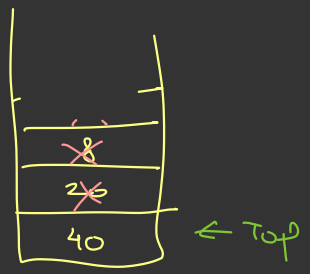
→ pop() ⇒ — = 8

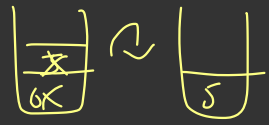
→ pop() ⇒ → 20

top() ⇒ 40 (Reads)

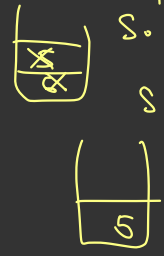
pop() → → 40

top() → error



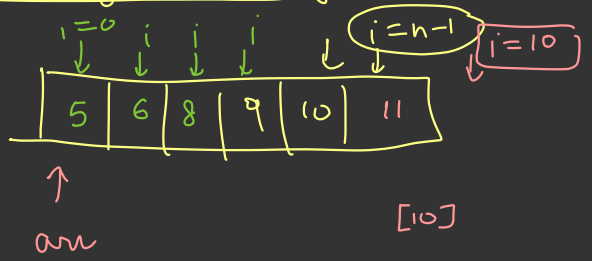


ans = S.top()
S.pop(),
S.pop()
S.push(ans)



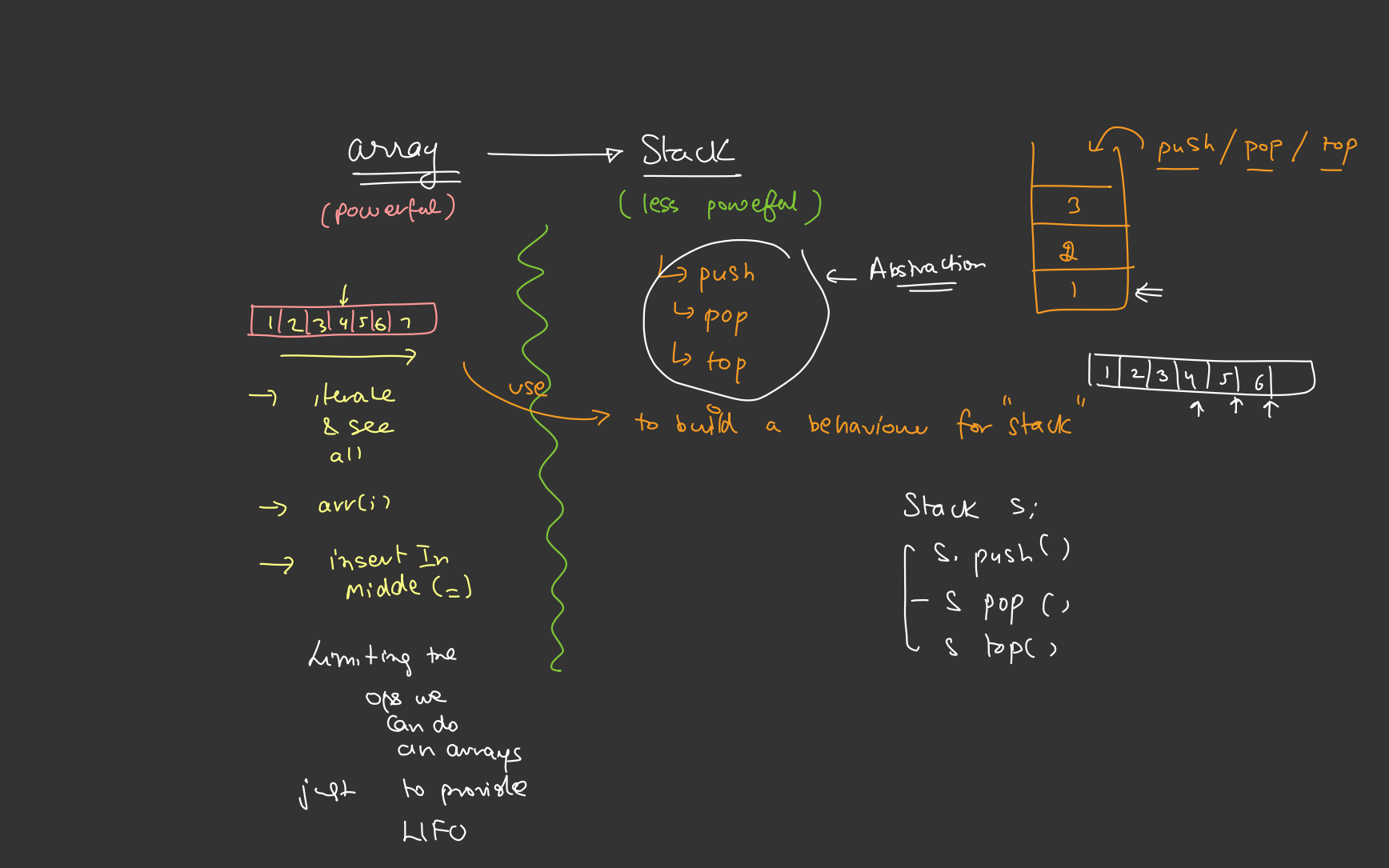
Stack Implementation

Stack using a array (Fixed / Dynamic → ArrayList)



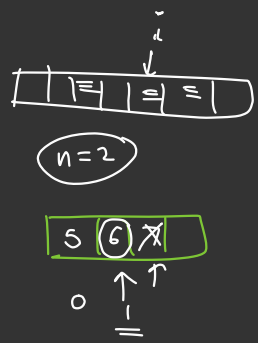

```
void push(int data){
    if(i == n){ print(stack is full);
    else {arr[i] = data;
          i++
    }
}
```

n → array.size()


array
(powerful)
Stack
(less powerful)
push / pop / top
3
2
1
push
pop
top
Abstraction
use
to build a behaviour for "stack"
1 2 3 4 5 6 7
→ iterate & see all
→ arr(i)
→ insert In middle (=)
Limiting the ops we can do on arrays just to provide LIFO
1 2 3 4 5 6
Stack s;
s.push()
s.pop()
s.top()

```
Class Stack {

    ArrayList<Integer> arr;
    Stack() { arr = new ArrayList<>(); }

    push(int data) {
        arr.add(data);
    }

    pop() {
        int n = arr.size();
        if(n==0){ print("Empty Stack"); }
        arr.remove(n-1);
    }

    int top() {
        return arr.get(n-1);
    }
}
```

hidden



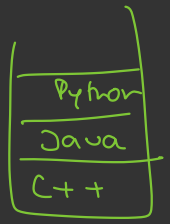
Any Stack problem can be solve using array.

Advantage

↳ Code would be simplified.

predefined

```
Stack s = new Stack();
s.push("C++")
s.push("Java")
s.push("Python")
s.pop()
```



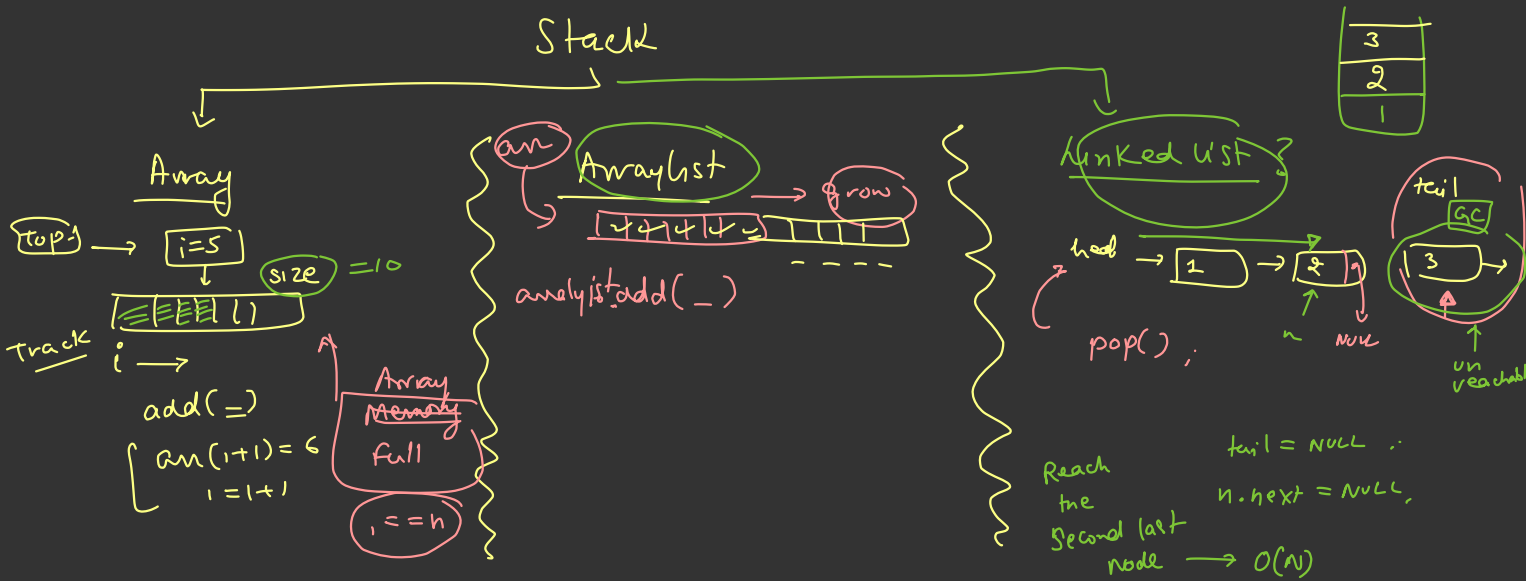
hiding the details of insertion behind the push fn

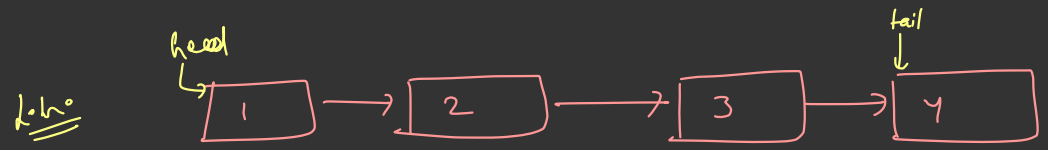

```
Array
String books[10];
int i=0,
books[i] = "C++"
i++
books[i] = Java
i++
books[i] = Python
i++
i=i-1
```

Loop

Maintain i

Enclosed in push fn of stack


Stack
Array
top
i=5
size
=10
Track
i
add(_)
arr(i+1)=6
i=i+1
Array
Memory
full
i==n
arr
ArrayList
grow
arraylist.add(_)
LinKed LIST
head
1
2
n
NULL
pop()
tail
GC
3
un reachable
Reach the Second last node → O(N)
tail = NULL
n.next = NULL
3
2
1

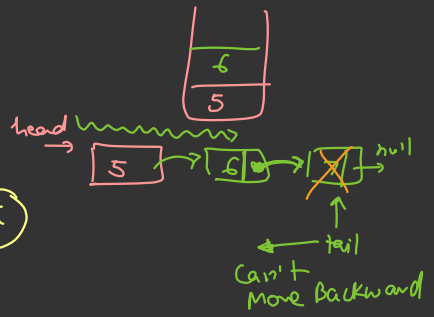


Like

Insertion

```
push( int d ){
    if(head == NULL) {  head = new Node(d)
                        tail = head;
    tail.next = new Node(d);
    tail = tail.next
}
```

O(1)



PROBLEM

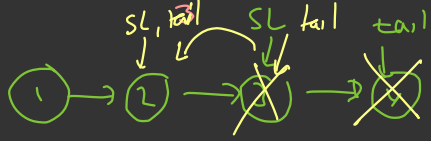
```
pop( ){
    tail = null
    SL.next = null
```

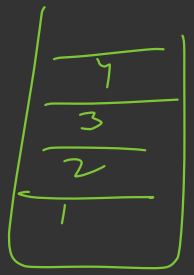
O(N) :(

Bad

iterate from the front to get SL

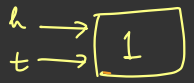
tail = SL,



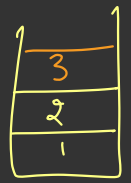


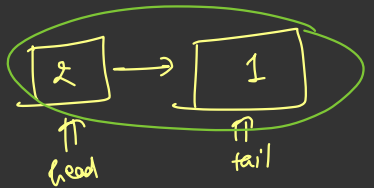
LinkedList

h → null



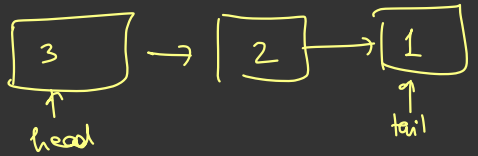
← Spush(1)

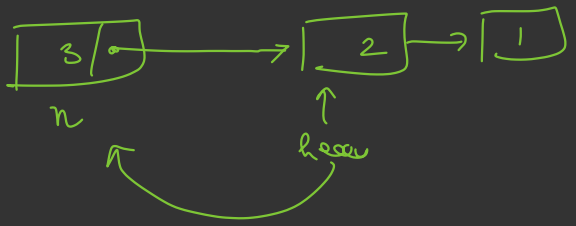




← S.push(2)

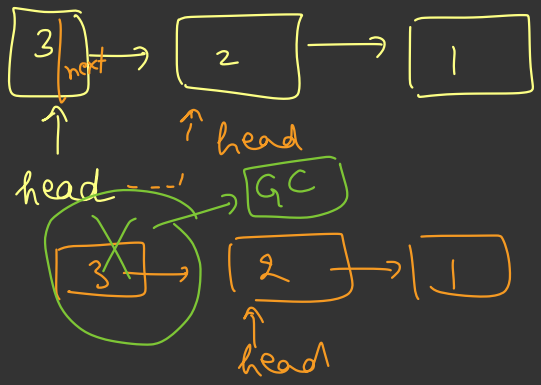
← Spush(3)

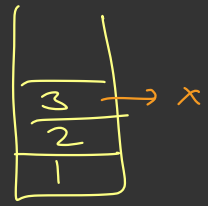




O(1)

```
insertAtHead( d ) {
    if(head==null){
        head = new Node(d);
    }
    Node n = new node(d)
    n.next = head;
    head = n;
}
```





pop(){

head = head.next;

}

O(1)

Insertion / pop at head

O(1)

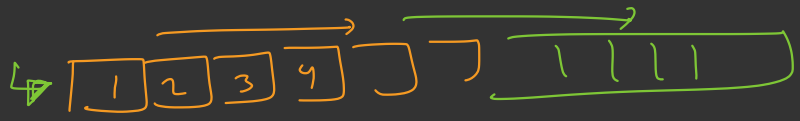
top() {

if (head != null) {

return head.data;

}

}



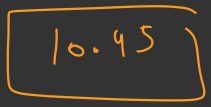
Linked List

Interview Question on Stack

↳ Design A Stack, TC of Pop / Push

↳ Solve XYZ Problem. => use case of Stack

↳ in-built stack (Java library)

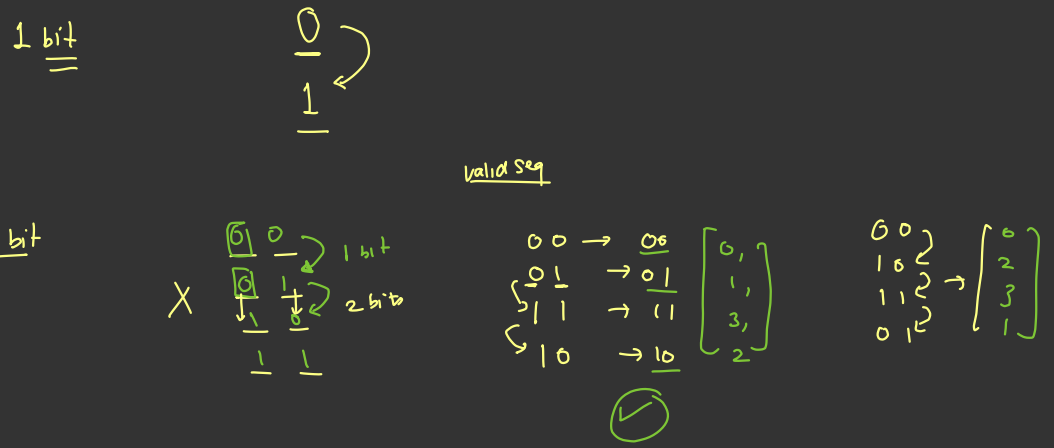


Grey Code

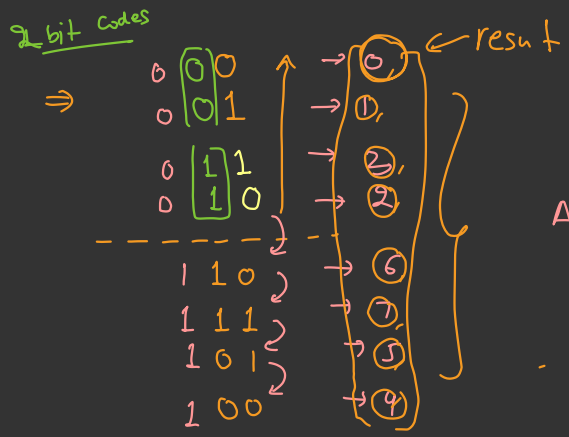
The gray code is a binary numeral system where two successive values differ in only one bit.

Given a non-negative integer A representing the total number of bits in the code, print the sequence of gray code.

A gray code sequence must begin with 0.



Way



A len codes → Bit codes of len A-1 → Bit codes of len (A-2) → ... → Bit codes of len 1

Soln =>

```
ArrayList result = {0}

int n = A
for (i=0; i<n; i++) {
    int currentSize = result.size();
    for(j= cs-1; j>=0; j--){
        result.add( result.get(j)
                    + (1<<i) );
    }
}
return result;
}
```

elements in current arrayList

in i^{th} iteration → generate all Codes of len (i+1)

A = 3

result = [0, 1]

⇒ i=0 CS = 1, j=0

⇒ i=1 CS = 2, [0, 1, 11, 10]

3, 2]

⇒ i=2 CS = 3 [0, 1, 11, 10,

1 + 1<<1

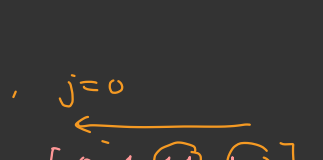
= 01
10
11

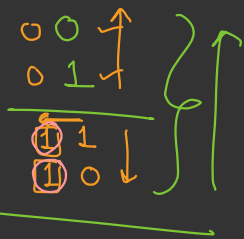
110, 111, 101, 100]

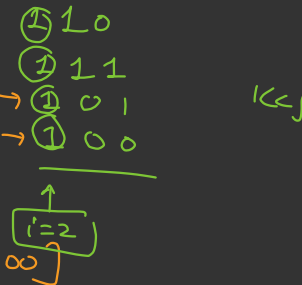
= [0, 1, 3, 2, 6, 7, 5, 4]

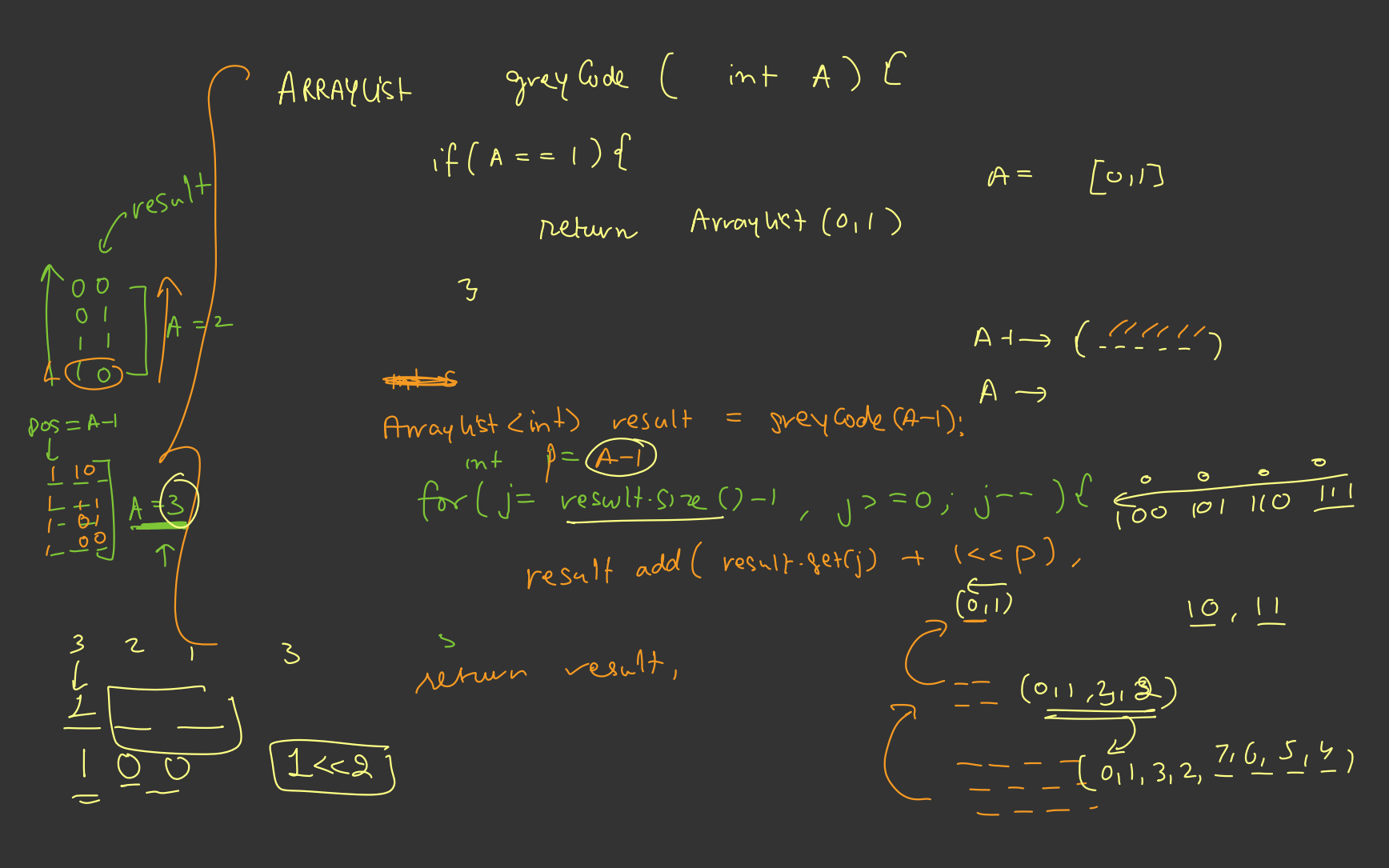
f(A) → f(A-1)

result result








ARRAYLIST greyCode (int A) {
if (A == 1) {
return ArrayList (0,1)
}
A = [0,1]
result
A = 2
pos = A-1
A = 3
ArrayList<int> result = greyCode (A-1);
int p = (A-1)
for (j = result.size()-1 , j >= 0 ; j--) {
result add (result.get(j) + 1<<p),
}
return result,
1<<2
A+1 →
A →
100 101 110 111
(0,1)
10, 11
(0,1,3,2)
(0,1,3,2,7,6,5,4)