# Video URL: <https://youtu.be/paI9fDG_ks4>

# Video URL Problems: <https://youtu.be/LJnhWFiwnCo>

# Probability:

I have fair coin which has 50% probability of heads and 50% probability of tails.

I flipped the coin and I see result as below

T T T T \_

When I flipped the coin 5th time what will I see here.

It could be head or tail, and both have 50% probability.

Each flip is independent of each other.

Even though we have all tails before doesn’t mean tail is more likely or head is more likely.

These events are independent of each other.

What is the probability that we see this sequence of flips – TT?

½ \* ½ = ¼

25% probability.

# Problem 1: Array = {1, 2, 3, …, 52}

Each of these numbers in array maps to deck of cards.

1 🡪 A Spades

2 🡪 2 Spades

3 🡪 3 Spades

Our array is representing a deck of card.

Create a shuffle function where array is the input. It shuffles and return shuffled array.

**What is a valid shuffle?**

How many permutations or ordering could be possible for the deck of 52 cards?

Total number of outputs = 52! Permutations.

A = {1, 2, 3, 4, 5, …, 52}

After shuffle

A = {1, 2, 3, 4, 5, …, 52}

Is it possible to have the same output? Yes, it is still a valid shuffle.

Probability of every output must be same = 1/52!

# Algorithm to implement this

**private static int**[] shuffle(**int**[] a) {  
  
 **for**(**int** x = 0; x < a.**length**; x++) {  
 Random random = **new** Random();  
 **int** max = a.**length**;  
  
 *//This is for picking random number between x and max(exclusive).  
 //nextInt take the bound and pick random number between 0(inclusive) to  
 // bound(exclusive)* **int** i = random.nextInt(max - x) + x;  
  
 *swap*(x, i, a);  
 }  
 **return** a;  
}  
  
**private static void** swap(**int** x, **int** y, **int**[] a) {  
 **int** temp = a[x];  
 a[x] = a[y];  
 a[y] = temp;  
}  
  
**public static void** main(String[] args) {  
 **int**[] a = **new int**[52];  
  
 **for**(**int** i = 0; i < 52; i++) {  
 a[i] = i + 1;  
 }  
  
 System.***out***.println(**"After shuffle "** + Arrays.*toString*(*shuffle*(a)));  
}

# Algorithm B

Some engineer comes and propose different algorithm

**private static int**[] shuffleB(**int**[] a) {  
 **for**(**int** x = 0; x < a.**length**; x++) {  
 Random random = **new** Random();  
  
 **int** i = random.nextInt(a.**length**); *//0 to 52(exclusive)  
 swap*(x, i, a);  
 }  
 **return** a;  
}

Why you wouldn’t use this?

We have 52! Outputs. Our goal should be that probability of every out should be 1/52!.

**Math Perspective.**

enum {

A,

B,

C

}

function(enum x) {

A 🡪 0,

B 🡪 1,

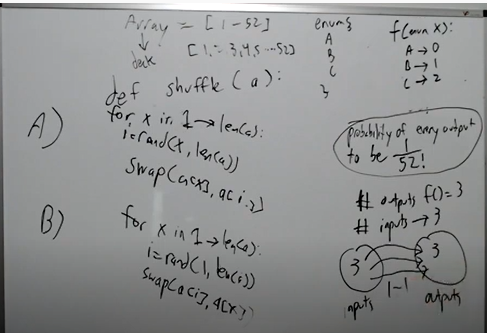
C 🡪 2

}

How many potential outputs = 3

Number of inputs = 3

1:1 mapping



**Example 2**

function(int x) {

return x;

}

Number of inputs = 2^32

Number of outputs = 2^32

2^32 <-> 2^32

**Example 3**

function(random(int min, int max)) {

return x;

}

Number of inputs = max – min

Number of outputs = max – min

**Example 4**

function() {

return random(int min, int max);

}

Number of outputs = max – min

Number of inputs = none which is 0

Are we changing anything now by moving random into the function?

How many outputs we have, same number of outputs and same number of inputs.

**For our Algorithm**

Number of outputs = 52!

x = 1 random(0, 52) 🡪 52

x = 2 random(0, 52) 🡪 52

Number of inputs 52^52

Can we do? 52^52 🡪 52!

Not every input has same probability

**Example 1**

function(x) {

x = 1 🡪 1

x = 2 🡪 1

x = 3 🡪 2

x = 4 🡪 2

}

Number of inputs = 4

Number of outputs = 2

2 inputs for every 1 output

If we call random(1, 4), what is the probability we get 1?

½

**Example 2**

function(x) {

x = 1 🡪 1

x = 2 🡪 1

x = 3 🡪 1

x = 4 🡪 2

}

Number of inputs = 4

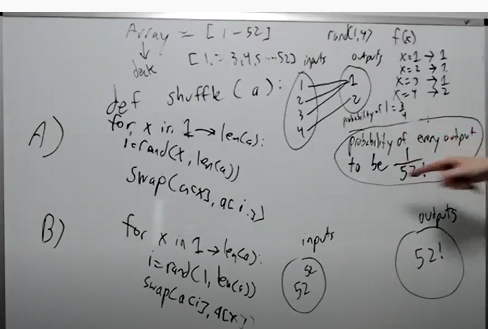
Number of outputs = 2

If we call random(1, 4), what is the probability of getting 1?

¾

If we call random(1, 4), what is the probability of getting 2?

¼



Probability of getting 1 is higher than getting 2. We can say this.

Inputs Outputs

52^52 52!

It is possible to do n 🡪 1 mapping?

What should n be?

52^52/52!

n has to be integer. Its impossible to have fractional mapping

52^52/52! is not an integer.

(2 \* 2 \* 13)^52/(1 \* 2 \* ..50) \* (17 \* 3) \* 52

(52)^51 \* (2 \* 2 \* 13)/(1 \* 2 \* 3 \* ..51) \* (17 \* 3) \* 52

This is fractional or decimal and not integer.

**Algorithm A**

**private static int**[] shuffle(**int**[] a) {  
  
 **for**(**int** x = 0; x < a.**length**; x++) {  
 Random random = **new** Random();  
 **int** max = a.**length**;  
  
 *//This is for picking random number between x and max(exclusive).  
 //nextInt take the bound and pick random number between 0(inclusive) to  
 // bound(exclusive)* **int** i = random.nextInt(max - x) + x;  
  
 *swap*(x, i, a);  
 }  
 **return** a;  
}

x = 0; random(0, 52) 🡪 52 results

x = 1; random(1, 52) 🡪 51

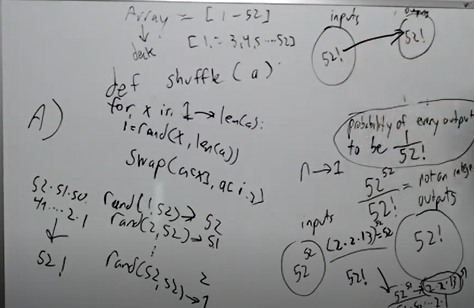
x = 2; random(2, 52) 🡪 50

x=51; random(51, 52) 🡪 1

Input: 52!

Outputs: 52!

1:1 mapping



If we have same number of inputs and same number of outputs, then we can say that every output has

Equal probability.

# More Intuitive way of thinking

If I give you deck of card and random number generator.

Random number generator is a bag.

Deck of cards goes into the bag.

Rand(1..52) and pick one and place it on the table.

We got 3 = {3,

Next time we call rand(51), it will not have 3

Pick 7 = {3, 7

Next time when we call rand(50), it will not have 3 and 7

Imagine picking the card out is removing from the bag.

Algorithm B is picking the card and then keeping on the table and again putting it back in the bag.

What if we choose the same card again? Then we have a problem. Therefore algorithm B doesn’t

Work.

# Problem 2

Set of two functions given.

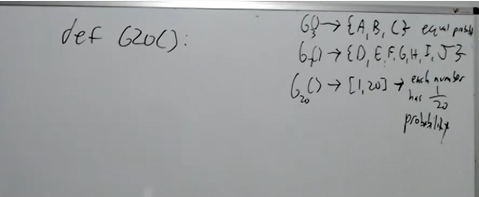
G3() 🡪 {A, B, C} randomly

G7() = {D, E, F, G H, I, J} randomly

G20() = {1, 2, 3, …, 20} randomly with equal probability

G20() returns number between 1 to 20 with equal probability – 1/20

The only function you can call in our implementation of G20() is G3() and G7().



3 \* 7 = 21

Inputs 21 -- Outputs 20

It’s not one to one mapping.

21 possible inputs to function.

We can map it out.

What we do if we get 21?

If we end up in this case, then call the function again.

**char** x = *G3*();  
**char** y = *G7*();  
  
**if**(x == **'C'** && y == **'J'**) {  
 **return** *G20*();  
}  
  
String key = String.*valueOf*(x) + String.*valueOf*(y);  
**return** *map*.get(key);

**What is the problem with this approach?**

It is very unlikely that when we call G3() and G7() and we get ‘C’ and ‘J’, but it can happen.

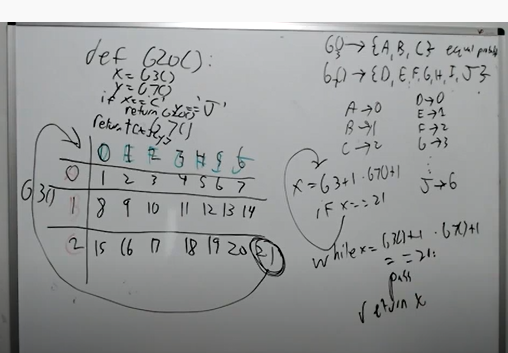
If it does happen then we keep doing that for unbounded amount of time.

Worst case scenario is we run forever. But if we did get an output, it has equal probability of 1/20.

**You don’t need mapping table.**

So you get 0, 1, 2 back for ‘A’, ‘B’ and ‘C’ respectively.

And you calculate by manipulating the numbers using formula.



Explaining the interviewer why we are going to have problem.

Think of probability questions as number of inputs 🡪 number of outputs

and try to create mapping between them.

# Problem – Popular number

a = {1, 5, 7, 7, 7, 1}

A number is popular if it occurs greater than > 25% of the time.

If we have 100 numbers, a number is popular if it occurs 26 or more time.

5 appears 1/6 times = 18 %

1 appears 2/6 = 33.33 %

7 appears 3/6 = 50%

Output: {1, 7}

If the array is unsorted, create map of numbers and their count.

Time Complexity: O(n)

Space Complexity: O(n)

**If the array is sorted**

a = {1, 1, 5, 7, 7, 7}

Is it possible to do better than O(n)?

Number is popular if it occurs > 50% of time

Here is our sorted array. In sorted array duplicates are together.

3 number, 10 numbers, 4 numbers

If we only care for the number that appears greater than 50% of time.

Where that number can be? It will be in the center.

How many numbers can occur greater than 50% of time? Only 1 time

If we are 50% then popular number is at the center.

Our popular number is at n/4.

If it’s 25% then we will check n/4, n/2, 3n/4 those are potential popular numbers.

3 candidates number are at indices {a[n/4], a[n/2], a[3n/4]}

For each candidate number x

Left = findLeft(a, 0, n, x); ----O(logn) Binary Search

Right = findRight(a, 0, n, x); ----O(logn) Binary Search

If(right – left + 1 > n/4) {

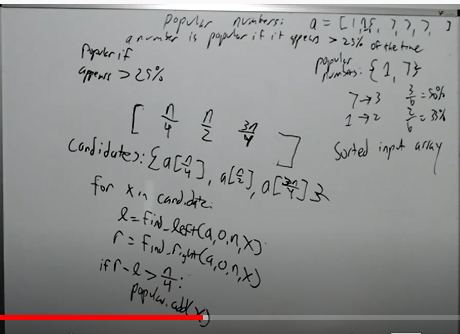
Popular.add(x);

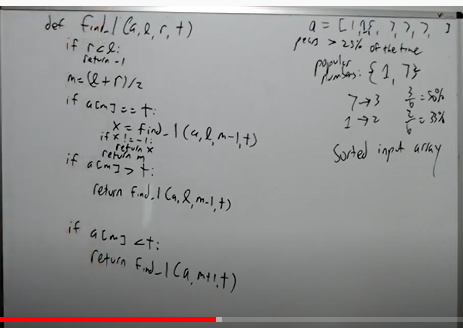
}

Binary search used to find left and right.

Time Complexity: O(100/p - 1 \* logn)

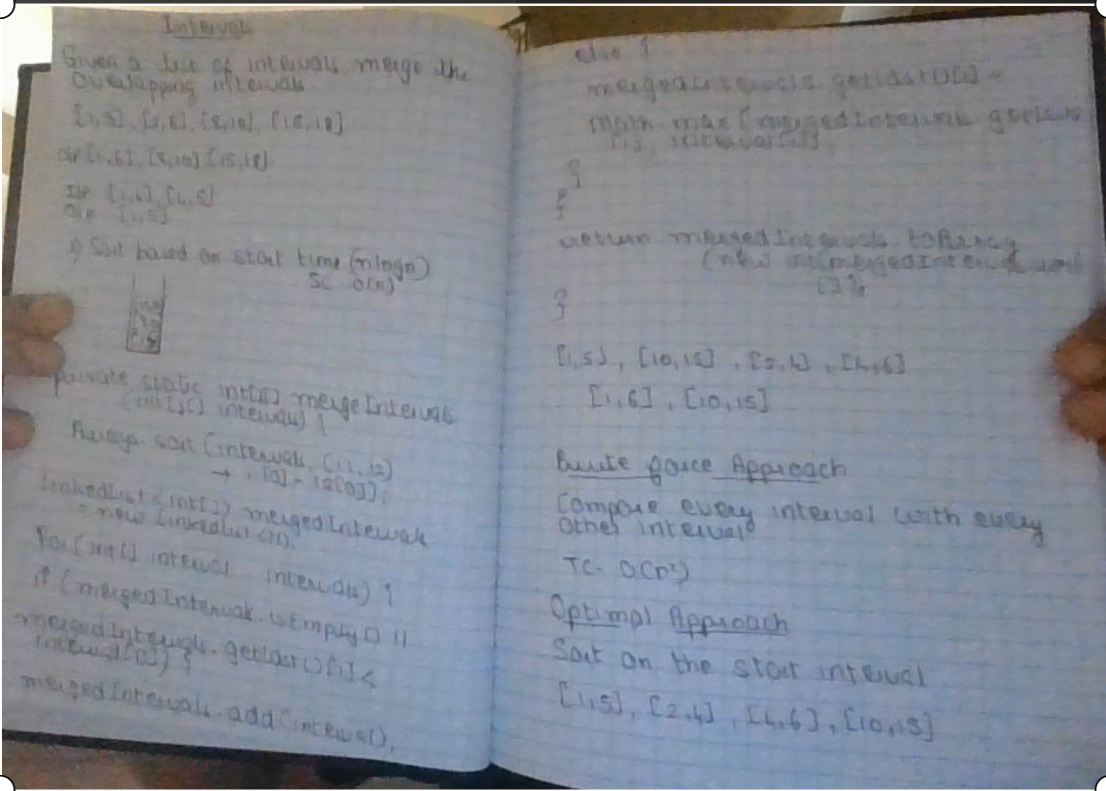
There will be 100/p – 1 candidates.

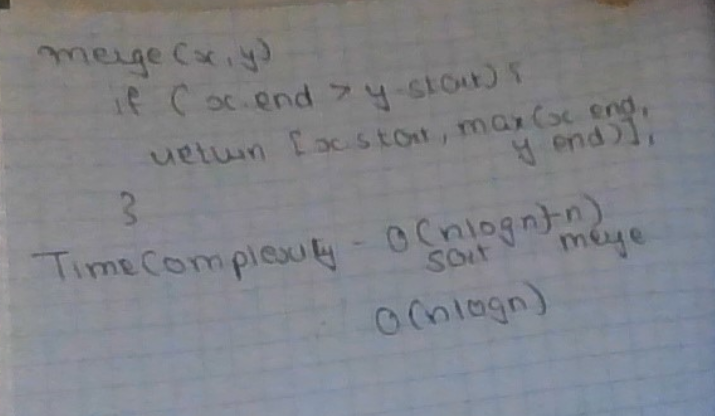


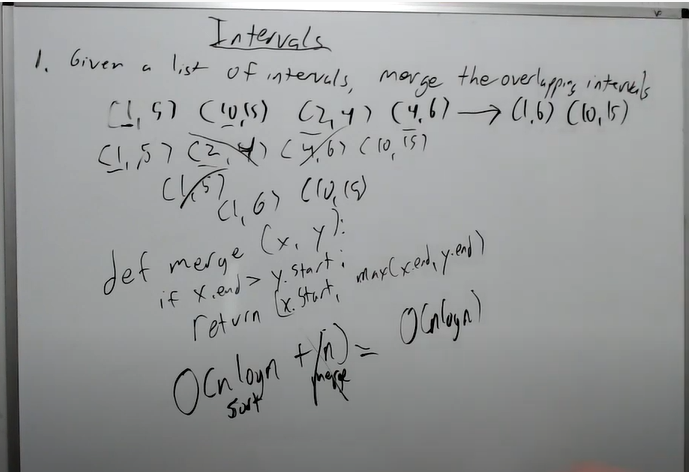


# Interval Probles and Magic Math problems which is composite data structure problem.

# Problem: Given a list of intervals, merge the overlapping intervals.







# Problem: Given a list of intervals. Given a stream of number identify how many intervals each number is in.

Example: (1, 5), (12, 15), (2, 4)

1 🡪 1

2 🡪 2

3 🡪 3

17 🡪 0

13 🡪 1

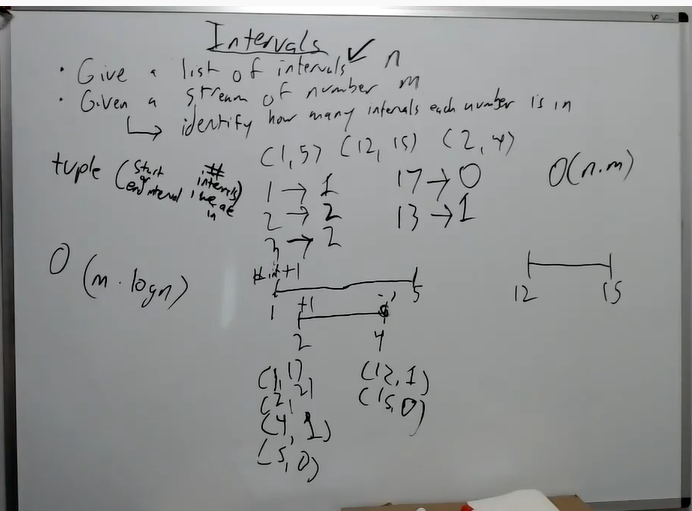
**Brute Force Approach**: Check the stream of number in each interval.

n intervals and m numbers then the Time Complexity: O(mn)

**Optimal Approach:** Create frequency array, do +1 for start and -1 for end +1 if end is inclusive.

Time Complexity: O(n + m) where n = number of intervals and m = max interval

Space Complexity: O(m) – where m = max interval



# Problem: Implement data structure with following operations.

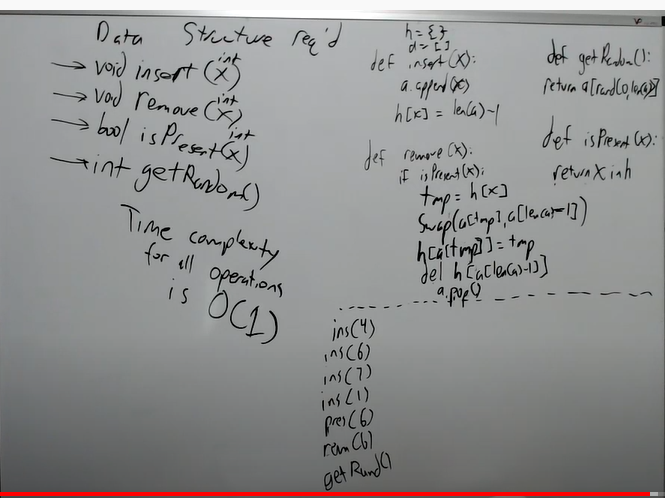
void insert(x)

void remove(x)

boolean isPresent(x)

int getRandom()

Time Complexity for all operations is O(1)



Composite data structure problems fit in Ad hoc.

# Interviewing Tips

1. IK
2. Leet code
3. Be vocal during interviews
4. Practice mock interviews topic based.
5. Schedule all interview in 1 week
6. Pick companies you are not targeting for practice
7. Timer based problem
8. Write down test cases and step through the code
9. White board practice
10. You cannot ace all interviews.