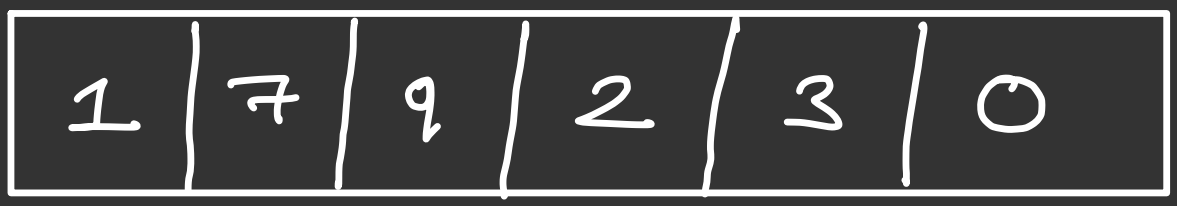
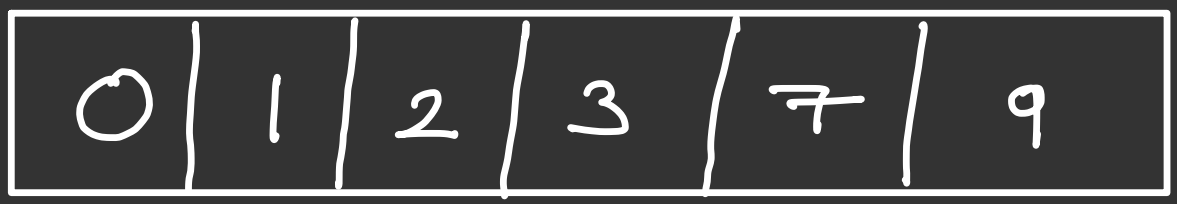
Sorting

Lecture 16

Selection Sort



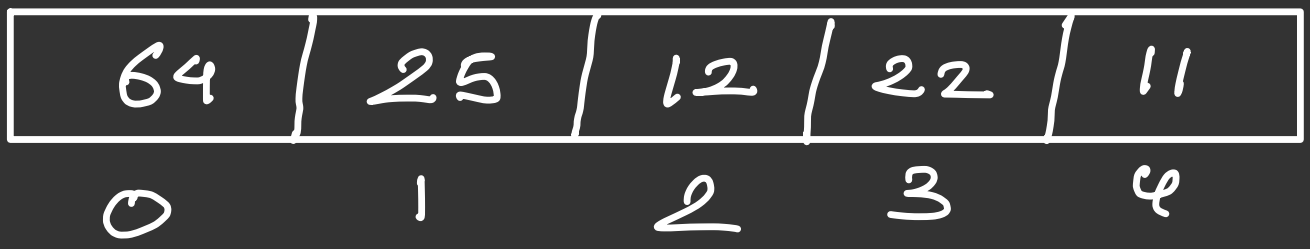
Sorted Array ⟶



⟶ different rounds

⟶ in each round select the smallest elements and place it in correct position.

example: (n=5)

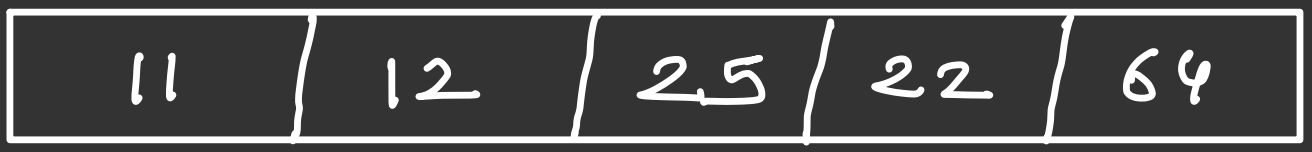


Round 1:-

$i = 0$

Round 2:-

$i = 1$

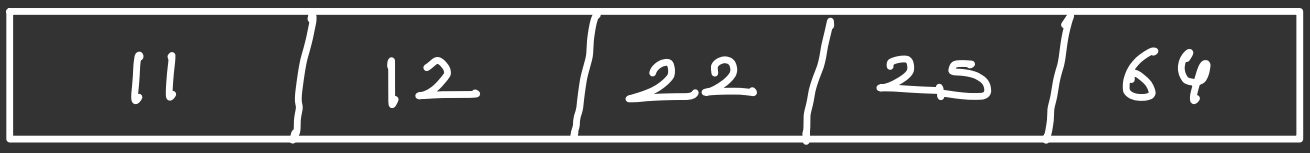


Round 3:-

$i = 2$

Round 4:-

$i = 3$



Thus, number of rounds $= n - 1$

⟶ selection_sort.cpp.

Space Complexity

↳ $O(1)$

Time Complexity

array size = n

then, first time loop ⟶ $n-1$

$n-2$

1

1

1

3

2

1

$= 1 + 2 + 3 -- \; n-2 + n-1$

$= \frac{(n-1+1)(n-1)}{2} = \frac{n(n-1)}{2}$

$= \frac{n^2-n}{2}$

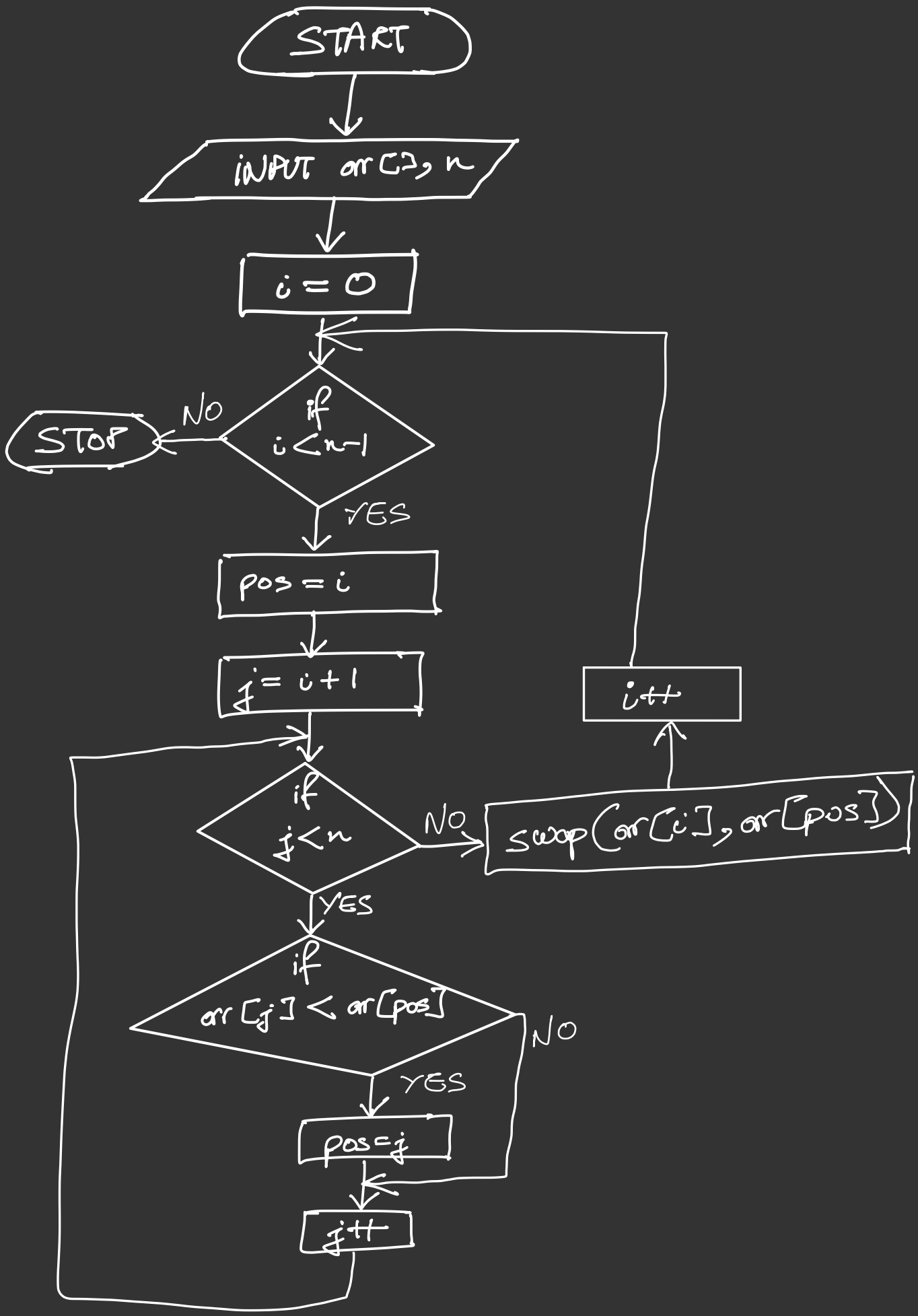
$O\left(\frac{n^2-n}{2}\right) = O(n^2)$

★ for best case as well as worst case, complexity $= O(n^2)$

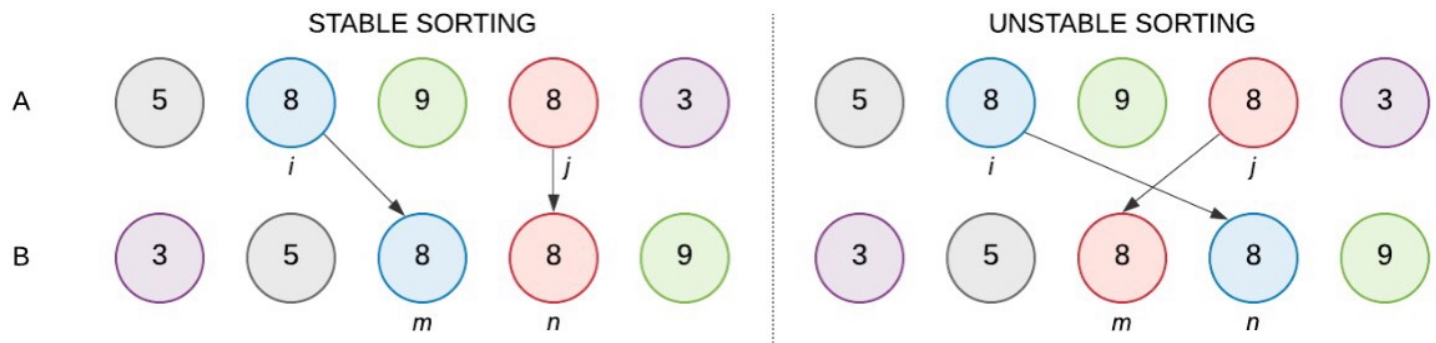
Use Cases

→ When the size of array / vector / list is SMALL.

FlowChart


START
INPUT arr[], n
i = 0
if i<n-1
NO
STOP
YES
pos = i
j = i+1
i++
if j<n
NO
swap(arr[i], arr[pos])
YES
if arr[j] < arr[pos]
NO
YES
pos=j
j++

Stable / Unstable Sorting



Stable Algorithms

- Merge Sort, Tim sort, Counting Sort, Insersion sort, Bubble Sort

Unstable Algorithms

- Quick sort
- Heap Sort
- Selection Sort

Lecture 17

Bubble Sort

→ check the adjacent elements

10	1	7	6	14	9

Round 1:

{1, 10, 7, 6, 14, 9}

{1, 7, 10, 6, 14, 9}

{1, 7, 6, 10, 14, 9}

{1, 7, 6, 10, 14, 9}

{1, 7, 6, 10, 9, 14}

1st largest element found and placed.

{1, 7, 6, 10, 9, 14}

Round 2:

{1, 7, 6, 10, 9}

{1, 6, 7, 10, 9}

{1, 6, 7, 10, 9}

{1, 6, 7, 9, 10}

Round 3:

{1, 6, 7, 9} x 3

Round 4:

{1, 6, 7} x 2

Round 5:

{1, 6}

* So ith round in Bubble Sort, you place the ith largest element corectly.

→ Stable Algorithm

→ bubble_sort.cpp.

Time Complexity

$1 \longrightarrow n-1$

$2 \longrightarrow n-2$

$3 \longrightarrow n-3$

$\vdots$

$n-1 \longrightarrow 1$

$$\Rightarrow \frac{n(n-1)}{2} \Rightarrow \frac{n^2-n}{2}$$

$$\Rightarrow O\left(\frac{n^2-n}{2}\right) = O(n^2)$$

Space Complexity

↳ O(1)

☆ To check if the code can be further optimised?

⟶ best case

⟶ worst case

Best case ⟶ when the array is already sorted, thus no swapping occured.

$a<b, b<c, c<d$

Thus, $a<b<c<d$

So, if no swapping, JUST BREAK OUT of the loop.

Thus, complexity for best case ⟶ $O(n)$

In Place Algorithm

→ are those who don't need any auxillary datas structure in order to transform the inputs datas.

✡ Making changes in the input array itself, instead needing an another empty array to store elements.

example:

8, 22, 7, 9, 31, 5, 13

8, 7, 22, 9, 31, 5, 13

8, 7, 9, 22, 31, 5, 13

8, 7, 9, 22, 5, 31, 13

8, 7, 9, 22, 5, 13, 31

7, 8 , 9, 22, 5, 13, 31

7, 8 , 9, 5, 22, 13, 31

7, 8 , 9, 5, 13, 22, 31

7, 8 , 5, 9 , 13, 22, 31

7, 5, 8 , 9 , 13, 22, 31

5, 7, 8 , 9 , 13, 22, 31

Lecture 18

Insertion Sort

→ card example

→ card will come one by one, you have to hold them in sorted order.

Example:

10	1	7	4	8	2	11

`k = arr[0];`

Round 1

$i = 1 \quad 1 < 10$

1	10	7	4	8	2	11

Round 2

$i = 2$

$7 < 10$

$7 > 1$

1	7	10	4	8	2	11

Round 3

$i = 3$

$4 < 10$

$4 < 7$

$4 > 1$

1	4	7	10	8	2	11

三

$i = 6$

1	2	4	7	8	10	11

→ Consider the first element as the sorted array

→ Now, divide the array into two
  - → sorted Array
  - → Non-sorted array

☆ We shift, not swap.

Why?

→ Adaptable Algorithm

→ Stable Algorithm

Space Complexity → O(1)

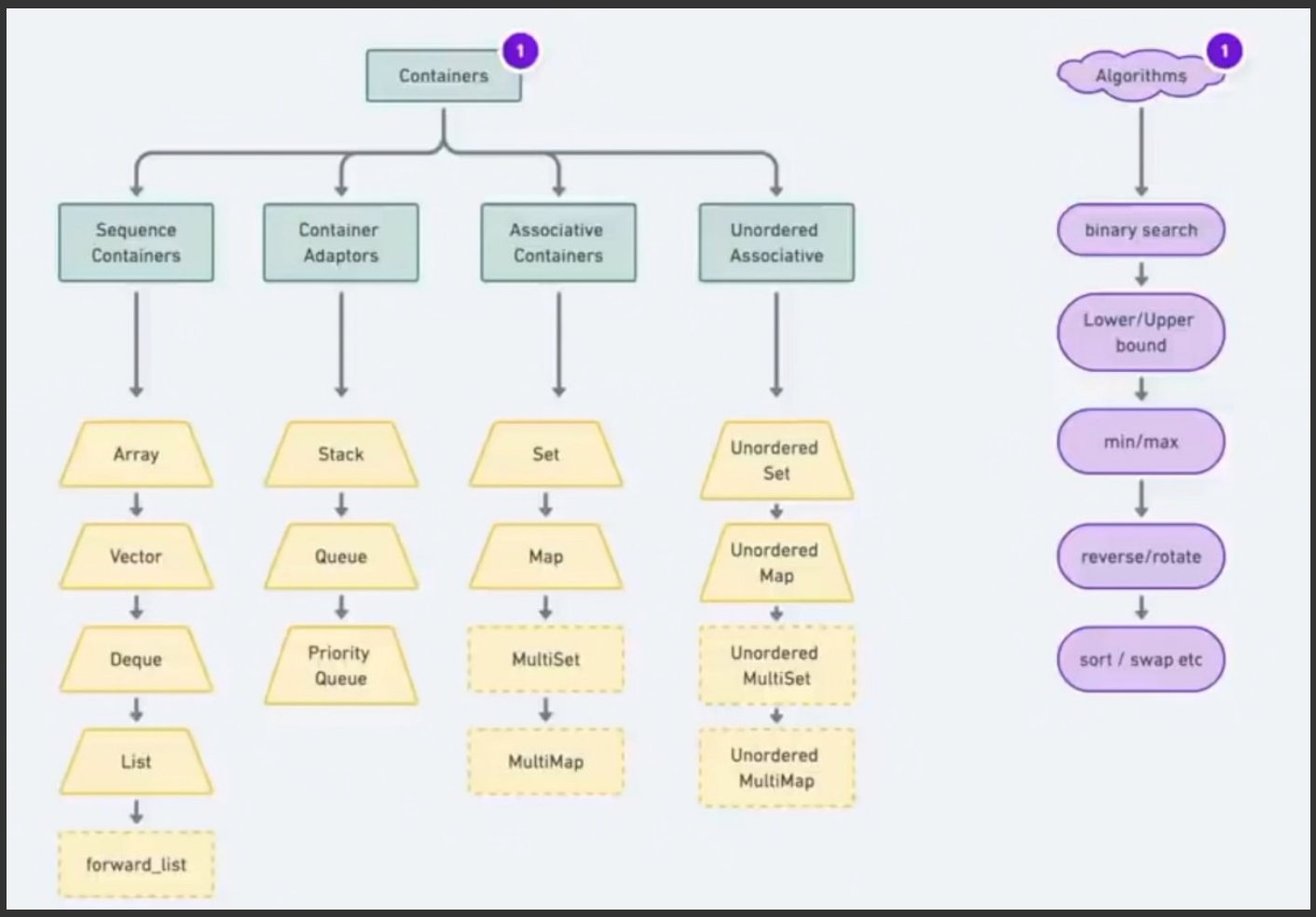
Time Complexity
- → $O(n^2)$

Best Case → $n-1 = O(n)$

Lecture 19

C++ STL

- Containers
- Algorithms



include <array>

array <int, 5> a ;

↳ declaration of array 'a' of 5 elements.

a.size(); ⟶ size of the array (or) number of elements

a[1] ⟶ referring to the element in the array at '1' index

a.at(1) ↗

a.front() ⟶ returns the first element

a.back() ⟶ returns the last elements

Vector

→ dynamic array

☆ When the vector is defined for 4 elements and we try to store the 5th elements

→ vector doubles its size by creating another vector of double the size and coping all the elements of the current vector to that vector.

#include <vector>

vector <int> v;
↳ declaring the vector

v.capacity();
↳ how much memory has been assigned for the elements.

v.size(); → how many elements are assigned.

v.push_back(1);
↳ adding element '1' to the vector

v.at(2) → to access elements

v.front()

v.back()

v.pop_back()
↳ removes the last elements
↳ size decreses, not the capacity

```
for(int i:v)
{
cout << i << " ";
}
```

→ accessing vector elements

★
```
vector<int> a(5,1)
```

→ size = 5
→ all elements have 1 initialised.

To copy vector a in vector last

→ `vector<int> last(a);`

★ Sorting of vector

→ `sort(v.begin(), v.end());`

include <deque>

→ deque_stl.cpp

~~Skipped~~ remaining for now

include <algorithm>

→ algorithms_stl.cpp

→ Binary Search

↳ ~~has~~ has to be a sorted array

→ upper bound

→ lower bound

→ Max
→ Min
→ Swap
} for two elements

→ Reverse

→ Rotation

→ Sort

↳ based on IntroSort

↳ Quick Sort

↳ Heap Sort

↳ Insertion Sort

✡ Lower & Upper Bound

↳ if that elements were to be present or maybe is present then the index value it would have been.

Lecture 20

Reverse the Array

→ reverse_array.cpp

Merge Sorted Array — LeetCode

→ merge_sorted_array.cpp

Move Zeros — LeetCode

→ move_zeros.cpp

Lecture 21

LeetCode - 189
Rotate Array

Logic ⟶

1	2	3	4	5	6
0	1	2	3	4	5

k=3

(k+i) % n

⟶ (5+3) % 6

⇒ 8 % 6 = 2

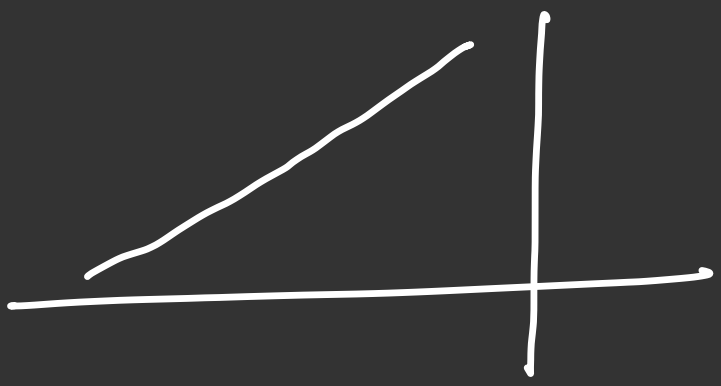
⟶ Shift at index = 2

⟶ rotate_array.cpp

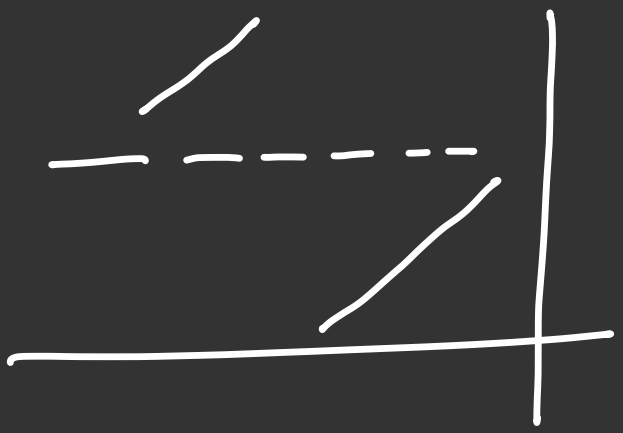
Sorted and Rotated Array

arr[] = {1, 2, 3, 4, 5}

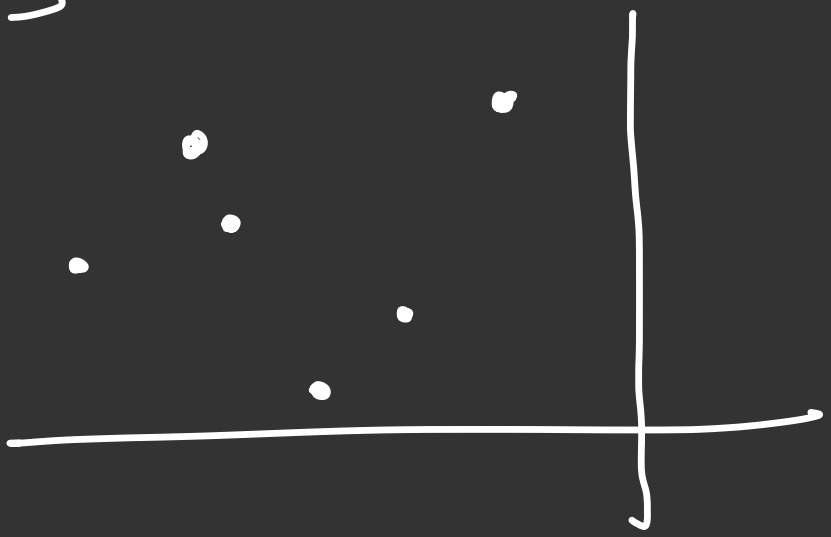
{1, 2, 3, 4, 5}



{4, 5, 1, 2, 3}



{4, 7, 5, 1, 3, 8}



Cyclic

↳ arr[i-1] > arr[i]

Also, cyclic → indexing

arr[n-1] > arr[0]

Another case

{1,1,1,1} → all elements are equal

→ check_rotated_sorted.cpp

Add 2 Arrays

Case 1:

```
  123
  456
-----
  579
```

Case 2:

```
   1
  12345
      6
-------
  12351
```

Case 3:

```
   1 4
   5 4 8 9
-------
   5 4 9 3
```

Case 4:

```
   1
   9 8 7
   9 2 1
-------
 1 9 0 8
-------
```

⟶ sum_of_2_arrays.cpp

Lecture 22

Char Arrays

Strings ⟶ 1-dimensional char Arrays

```
char a='z';
```

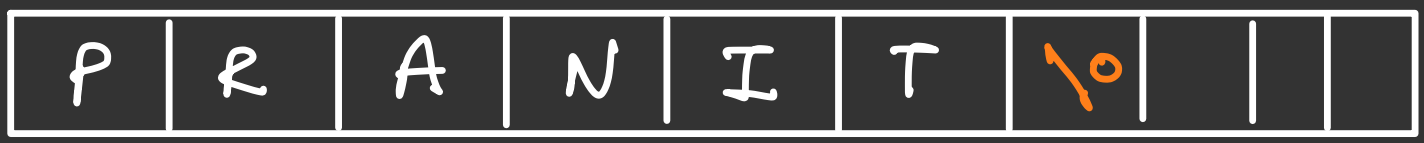
↳ char Array

Syntax: `char ch[10];`

i/p ⟶ `cin >> ch;`

o/p ⟶ `cout << ch;`

PRANIT



Null character ⟶ '\0'

↳ ASCII value = 0

↳ Acts as a terminator

☆ Automatically gets added in the end.

✍ If you give "PRANiT Modi" as inputs, it will only take PRANiT as inputs

Reason ⟶ cin stops execution when space (" ") tab ("\t") enter comes. ("\n")

✍ While Printing, it only prints till before null character

⟹ "BA\0BAR\0"

↳ this may be stored

↳ but only BA is printed.

Leet Code

Reverse String

⟶ reverse_string.cpp

Check Palindrome

→ same as reversing the string:

1. Take 2 pointers, from both the ends of the string
2. Check if the characters are same or not, keeping a counter variable to keep a check
3. i++; j--;
4. Return

→ palindrome_charArray.cpp

(CASE SENSITIVE)

⭐ Converting character to Lower Case

```
char ch;
if(ch >= 'a' && ch <= 'z')
    return ch;
else
    char temp = ch - 'A' + 'a';
    return temp;
```

Check Palindrome, that only consides alphabets and numbers and skips the rest

⟶ palindrome_not_caseSensitive.cpp

String

string str = "hello"

→ this does not contains '\0' (null character)

length → str.length();

add a character in the end

↳ str.push_back('!');

to remove last character

↳ str.pop_back();

Good Resource

↳ cplusplus.com

⚠ To include spaces while input

↳ getline(cin, str);

option for delimeter also eg: '\0'

→ string.cpp

Key Differences b/w

Char Array	String
→ collection of variables of char datatype	is a class and variables of strings are objects of class string.
→ Array boundaries are easily overrun	Boundaries will not overrun
→ char ch[20];	string str;

Reverse words (LeetCode Premium)

example:

i/p ⟶ "My name is"

o/p ⟶ "yM eman si"

1. Reverse the word before each time there is a space
2. Lastly after the loop reverse the last word (store the last space info)

⟶ reverse_words.cpp

Max Occurance of Character from A to Z

Using int array of 26 elements

⟶ max_Occurance_Char.cpp

In Built functions of char Array

```
char ch1[20];
```

→ length

```
int l= strlen(ch1);
```

→ compare two

```
strcmp(ch1, ch2);
```

↳ 0 = equal
↳ !0 = not equal

→ copy

```
strcpy(destination, source);
```

For String

```
string s;
```

→ `s.length();`

→ `if(s1 == s2)`

→ `s1 = s2;`

→ in_built_functions_charArray.cpp

Replace Spaces with '@40'

→ replace_spaces.cpp

Remove all occurances of a Substring — LeetCode 1910

example:

daabcbaabcbc

abc

aab

abc

dabaabcbc → 1

dababc → 2

dab → 3

→ remove_all_occurances.cpp.

String Concatination

```
s1.append(s2);
```

Substring

```
s1.substr(pos, len);
```

pos ⟶ index from which to start

len ⟶ number of characters to be included

if len > s1.length(), characters are taken till the end.

Permutation in String — LeetCode (567)

Example:

s1 = "eidbaooo"
s2 = "ab"

1. Make an int arr[26] and store character counts of s2

2. Window of the size s2 and check for same letters.

⭐ What I Used ⟶

Creating that window everytime and counting every time.

Other way ↳ as each time its incrementing by one just minus the count at index left and add the count for index added.

This method significantly reduces the the Runtime

⟶ permutation_in_string.cpp

#LeetCode – 443

String Compression

example:

a a L=2

i = 0+2

ch = a

count = 2

⟶ string_compression.cpp

Optimized Solution ⟶

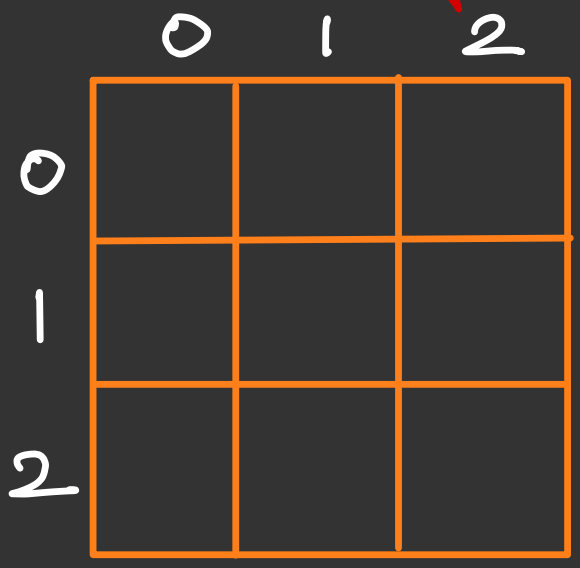
```
int i=0;
int ansIndex =0;
int n= chars.size();
```

a b b b c

0 1 2 3 4

```
while (i<n)
{
    int j=i+1;
    while (j<n && chars[i]==chars[j])
      j++;
    chars[ansIndex++]=chars[i];
    int counts=j-i;
    if (counts>1)
    {
       string cnts=to_string(counts);
       for(char ch:cnts)
       {
           chars[ansIndex++]=ch;
       }
    }
    i=j;
}
return ansIndex;
```

Lecture 23
2-D Arrays



how you store

how you visualise

$\Rightarrow c \times i + j$

c = number of columns

i = row index

j = column index

this is how it is still stored inside memory, but we represent like int arr[3][3];

arr[2][3];

↑ row ↑ column

arr[2][1] = {1, 2};

(or)

arr[2][1] = {{1}, {2}};

→ column and row wise inputs and printing

→ 2d_array.cpp

Linear Search in 2d Array

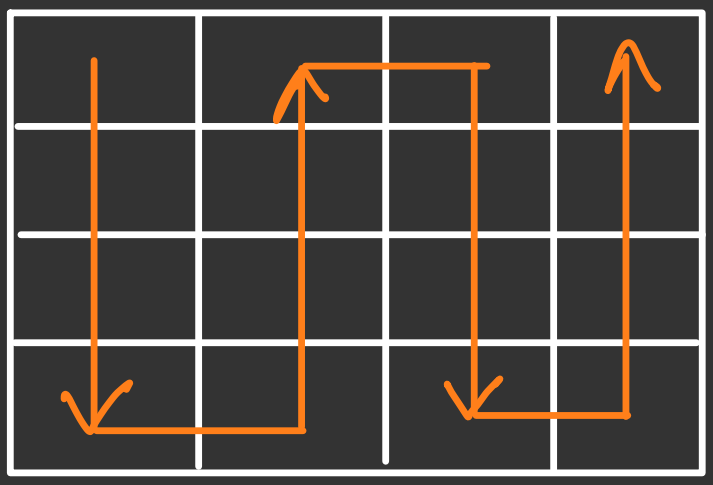
→ linear_search_2d.cpp

★ If you are passing 2d array in function, then when defining function at least define the column number in the parameter.

→ Row wise Sum + largest sum of which row ?

→ row_wise_sum.cpp

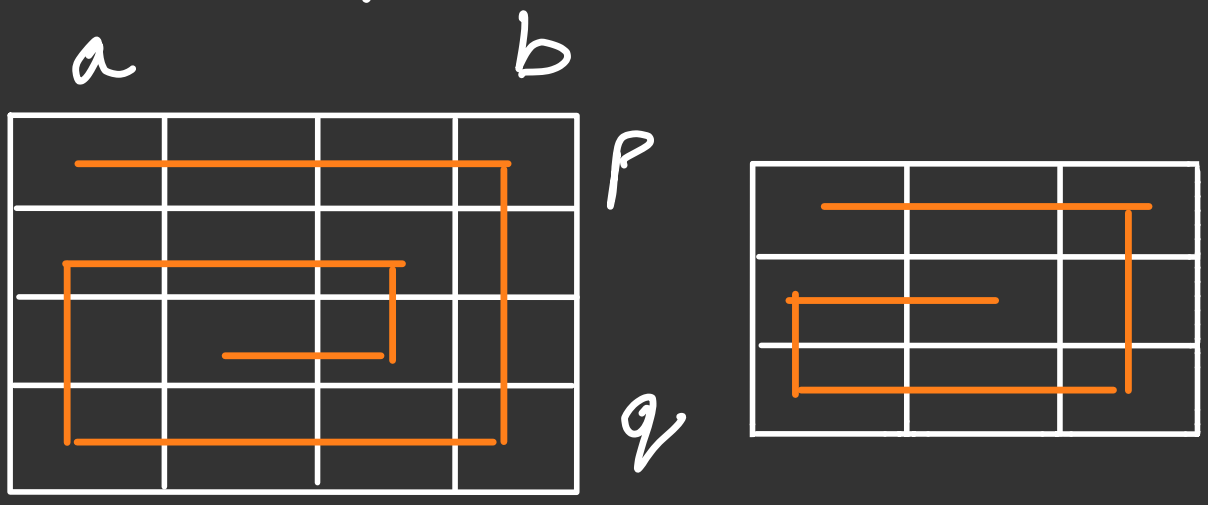
→ Prints like Wave

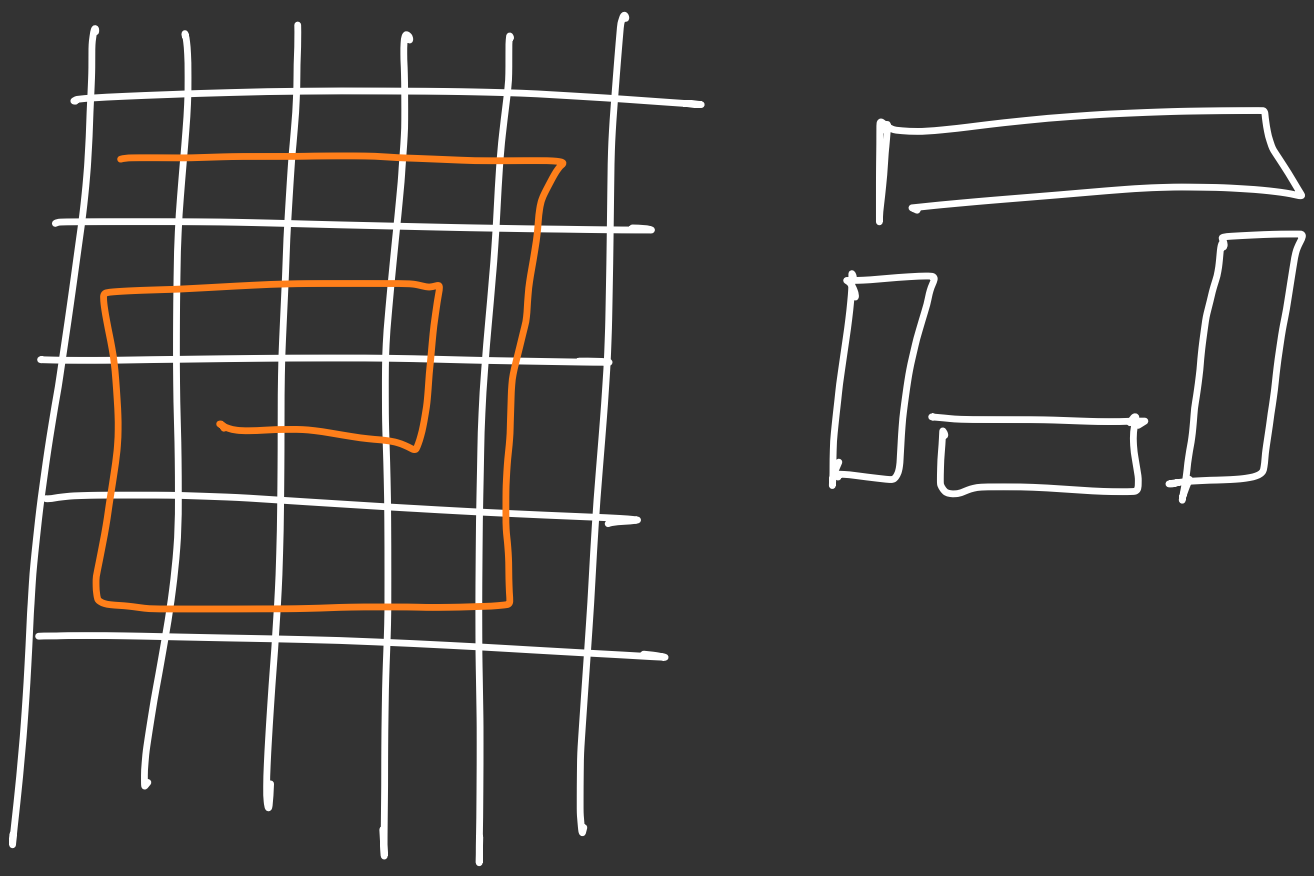


→ prints_like_wave.cpp

Spiral Matrix — LeetCode #54

→ spiral_matrix.cpp.





a = + 2

b = + 0

q = 2 1

Rotate Array by 90° (in-place)

LeetCode — 48

The Logic ⟶

1. Rotate each element one by one
2. After the outer ring elements have taken their position, the move to the inner ring
3. No of moves for each element decreases by two as we do each ring inwards.

⟶ rotate_90.cpp

Binary Search in 2D Array — I

LeetCode - 74

(m x n) matrix

My Approach

- focus on the first column
- find the row in which the element should be there
- now traverse that row again using binary search

Oms

Approach 2

→ treat is a linear array

```
start = 0
end = row * col - 1;
mid = s + (e-s)/2;
midVal = matrix[mid/col][mid % col];
```

→ search_2d_1.cpp

Search a 2D Matrix II

LeetCode - 240

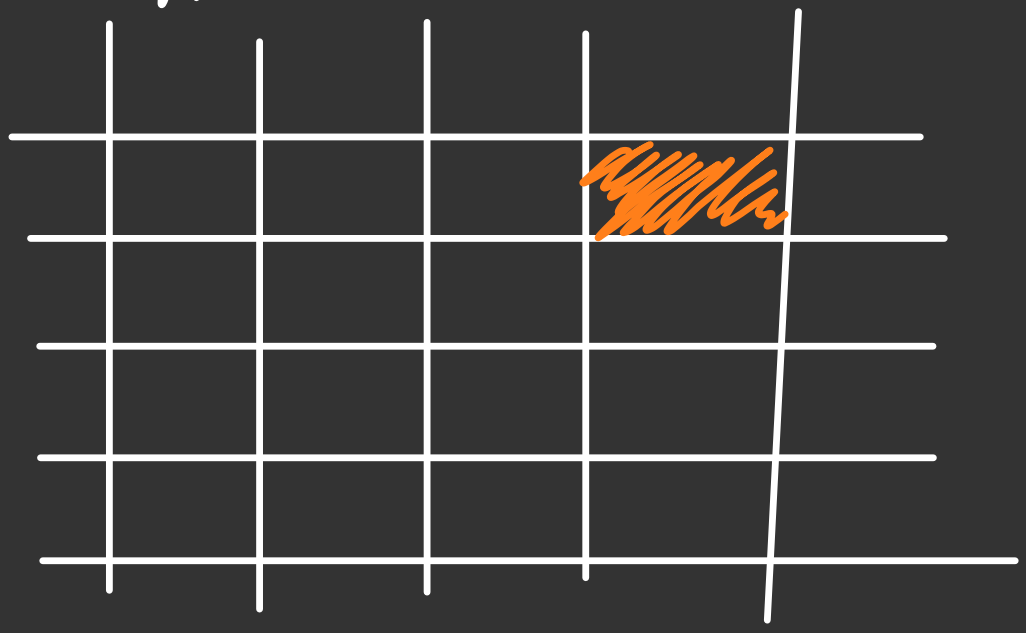
→ each row sorted left to right

→ each column sorted top to bottom.

→ search_2d_2.cpp

☆ My approach had loopholes.

New Approach →



focus on this element = e,

target > e

↳ wont be in the row

thus row++;

target < e → wont be in thats column

column--;

Lecture 24

Count Primes

LeetCode – 204

Sieve of Eratosthenes

→ Start with 2, make all its multiple below n as not prime.

→ Next do it for 3

→ Automatically, next no. is 5, as all non prime one's gets eliminated

→ Do this till n

Time Complexity

$\left(\frac{n}{2} + \frac{n}{3} + \frac{n}{5} - - -\right)$

$\Rightarrow n\left(\frac{1}{2} + \frac{1}{3} + \frac{1}{5} - -\right)$

HP of prime numbers

$\Rightarrow n(\log(\log n))$

★ by using Tailor Series

→ count_primes.cpp

★ Segmented Sieve

→ Instead of counting the number of primes, you have to print them.

GCD (Greatest Common Divisor)
@ HCF (highest common factor)

$$gcd(a,b) = gcd(a-b, b)$$
$$= gcd(a\%.b, b)$$

example:

$$gcd(72, 24)$$
$$= gcd(48, 24)$$
$$= gcd(24, 24)$$
$$= gcd(0, 24)$$

$\longrightarrow$ 24 is the answer

$\longrightarrow$ gcd_hcf.cpp

Modulo Arithmetics

$a\%.n \longrightarrow [0 \text{ —— } (n-1)]$

$\Longrightarrow (a+b)\%.m = a\%.m + b\%.m$

$\Longrightarrow a\%.m - b\%.m = (a-b)\%.m$

$\Longrightarrow a\%.m * b\%.m = (a*b)\%.m$

Fast Exponention

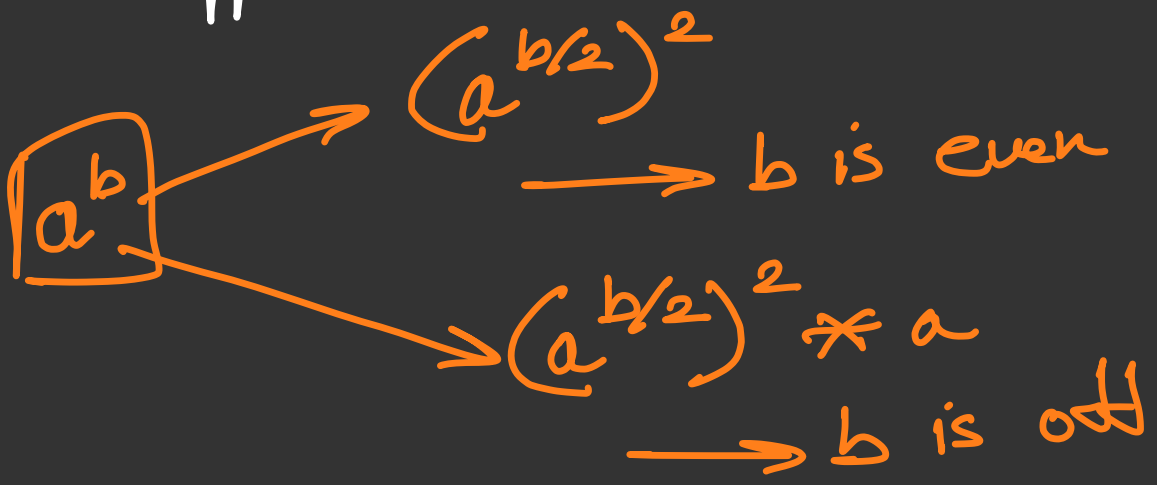
Old Approach ⟶

a^b

```
res = 1;
for(1 ⟶ b)
{
    res = res * a;
}
```

TC = O(b)

New Approach ⟶



TC = log b

Example : ⟶

2^5

$\Rightarrow 2 \times 2 \times 2 \times 2 \times 2$

res = 2

a = 2x2

b = 2

a = 2x2x2x2

b = 1

res = 2 x 2 x 2 x 2 x 2

b = 0

return res;

☆ Multiply by 1LL to type cast it to long, but still remains integer.

⟶ fast_exponentiation.cpp.

☆ Other topics to study

- → Pigeon Hole Principle
- → Catalan Number
- → Inclusion-Exclusion principle
- → factorial of a number using % m

example: 212! % m

m = 10^9+7

☆ Probability Questions in DSA