**Q. What is stack and heap memory where we are using heaps or which language?**

**Ans : Stack Memory**:

The stack is a region of memory used for storing local variables and function call information.

It operates in a Last-In-First-Out (LIFO) manner, meaning that the most recently called function is the first one to finish and release its memory.

Stack memory is generally faster to allocate and de-allocate because it involves simple pointer manipulations.

It has a limited and fixed size, typically determined by the operating system.

Commonly used in languages like C, C++, Java (for method call frames), and many others.

**Heap Memory:**

The heap is a region of memory used for dynamic memory allocation. It's where you allocate memory explicitly and manage its lifetime.

It doesn't have a fixed size like the stack and can grow or shrink as needed during runtime.

Memory allocation and de-allocation in the heap can be more complex and slower compared to the stack, as it involves searching for available memory blocks.

It's commonly used in languages like C, C++ (via malloc, free, new, delete), Python (for objects created using new or malloc), and many others.

**Q. Write an 2d array and rotate to in 90 degree?**

**Ans :-** matrix = [[1,2], [3,4]]

row = len(matrix)

col = len(matrix[0])

for i in range(row):

for j in range(i,col):

matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

for i in range(row):

matrix[i] = matrix[i][::-1]

for row in matrix:

print(row)

**TAKING INPUT:**

row = int(input("Enter the number of rows: "))

col = int(input("Enter the number of columns: "))

matrix = []

print("Enter the elements of the matrix:")

for i in range(row):

row\_data = []

for j in range(col):

element = int(input(f"Enter element at position ({i+1}, {j+1}): "))

row\_data.append(element)

matrix.append(row\_data)

for i in range(row):

for j in range(i, col):

matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

for i in range(row):

matrix[i] = matrix[i][::-1]

print("Resulting matrix after transpose and reflection:")

for row in matrix:

print(row)

p=0

for (i=1;p<=n;i++)

{

p = p+i;

}

1 0+1 bcs p=p+i…When n is 0

2 1+2 = 3 …………n is 1

3 1+2+3=6

K 1+2+….k

Not n times assuming

When will stop when p>n

P=k(k+1)/2 bcs 1+2+….+k

P=k square > n

So n= sqrt of n

Time complexity o(sqrt(n))

**O (log n base 2):**

For (i=1;i<n;i\*2)

{

Statements;

}

ANALYZE

I=1 1 time

I=2 2 times (1\*2)

I=3 4 (1\*2)\*2 power 2

I=4 8 (1\*2)\*2)\*2 = 2 power 3

So when stops i>=n

i = 2power k

2 power k>=n

K=log n base 2

So time complexity O(log n base 2)

**Q. Create an 1 dimensional array that contain 10 to 30**

1. extract and print even number
2. 2 power values

**Ans :-**

x = int(input("Enter initial value: "))

y = int(input("Enter ending value: "))

print("Here is available even numbers")

for i in range(x, y+1):

if i % 2 == 0:

print(i)

print("Here is 2 power values")

for j in range(x, y+1):

if (j&(j-1)) == 0 and j!=0:

print(j)

**O (log n):**

For(i=n;i>=1;i=i/2)

{

Statements;

}

I

N

n/2

n/2 power 2

………..

n/2 power k

Assume i<1 it stops right?

n/2 power k<1

n/2 power k=1

n=2 power k

k= log n base 2 so O(log n)

**DRIVED FORMULAE:**

For(i=0;i<n;i++) ------O(n)

For(i=0;i<n;i+2) ------O(n)

For(i=0;i<n;i--) ------O(n)

For(i=0;i<n;i=i\*2)

* Constant **Time Complexity:** O (1)……
* Linear **Time Complexity:** O (n)…….
* Logarithmic **Time Complexity**: O (log n)……..
* Quadratic **Time Complexity**: O (n2 )…….
* Exponential **Time Complexity**: O(2n)

FINAL SUMMERY:

i++ , i-- , i+2 O(n)

i\*2 i/2 log(n) base 2

**SPACE COMPLEXITY:**

Parallel concept to time complexity , array of size n, require O(n)space.

Two-dimensional array of size n\*n : O(n2) space.

Linear Search O(1) 🡪🡪Context space complexity(same amount of space regardless of the input size n it is call context complexity) **Eg**:- some of array elements and linear search🡨🡨

Merge Sort O(n)

Depth First Search O(n)

BFS O(n)

Dynamic Programming: O(n2) or O(n\*m)

**Q. Create an array and calculate sum the elements by using function?**

**Ans:**

def sum(arr):

total = 0

for element in arr:

total +=element

return total

num = input("Enter the element: ")

input\_elements = num.split()

arr = [int(element) for element in input\_elements]

result = sum(arr)

print("The sum of elements in the array is:", result)

**LINEAR SEARCH DYNAMICALLY:**

def linear\_search(arr, target):

index = 0

for element in arr:

if element == target:

return index

index += 1

return -1

arr = input("Enter a list of numbers : ").split()

arr = [int(num) for num in arr]

target = int(input("Enter the target element to search for: "))

result = linear\_search(arr, target)

if result != -1:

print(f"Element {target} found at index {result}")

else:

print(f"Element {target} not found in the list")

Polinomial complexity: O(n2)

Space complexity grows proportionally to the square of the input size.

def generate\_list\_of\_list(n):

table\_list = []

for num in range(n):

row = []

for i in range(n):

row.append(i)

table\_list.append(row)

return table\_list

print(generate\_list\_of\_list(10))

fibonacci searies and sum of n numbers and factorial