28 September,2023

Continue….

Type II:

For(i=0;i<n;i++)……n+1 times

{

For (j=0;j<n;j++)…..n\*(n+1) times

{

Statements; n\*n…..n square

}

}

Time Complexity= O(n square)

Nested for loops, the time complexity is O(n square)

Q: Implement a 2D Array and rotate the matrix 90 degree.

In C, Java contains Primitive Datatypes i.e, Predefined datatypes. We declare variables first, where as in Python , non primitive datatypes, we don’t declare variables.

Q: What is stack and heap? which languages are using these?

Stack used for local variables and function calls where as Heap used for dynamic memory allocation and in data structures and creating objects.

Pseudocode for adding two numbers:

Set n1, n2

n3: n1+n2

print: n3

Q: Implement a 2D Array and rotate the matrix 90 degree.

9 3 rotate 90 (anti clockwise) 3 6 rotate 90(clock) 10 9

10 6 9 10 6 3

10 15 20 20 300 3 1 100 10

100 200 300 rotate 90(anti clockwise) 15 200 2 rotate 90(clock) 2 200 15

1 2 3 10 100 1 3 300 20

Logic is Transpose the matrix and Reverse the rows, for clockwise.

For(c-1,-1,-1) 🡪values are 4,3,2,1,0

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

{

For(j=0;j<i:j++)

{

Statements;

}

}

i j

0 nothing

1 0(1(stop))

2 0,1(2(stop))

When 0 🡪0 times

1🡪1 time

2🡪2 time

Total=(0+1+2)=3 which is sum of n natural numbers.

(1+2+3+…..+n)=(n)(n+1)/2

=(n square +n)/2

Time Complexity is O(n^2)

Type III:

p =0

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

{

p=p+i;

}

i p

1 0+1

2 1+2=3

3 1+2+3=6

...

k 1+2+…+k

p =k(k+1)/2

p=k square>n

So, n=sqrt of n

Time Complexity is O(sqrt(n)).

Type IV:

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

[

Statements;

}

i=1 1 time

i=2 2 times(1\*2)

i=3 4 times((1\*2)\*2=2 power 2)

i=4 8 times(1\*2\*2\*2=2 power 3)

So when stops i>=n

i =2 power k

2 power k>=n

k =log n base 2

Time Complexity is O(log n base 2)

Q:Create an 1D array. It should contain numbers between 10-30.

a. Extract and print even numbers.

b.2 power values

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

{

Statements

]

Time Complexity is O(log n base 2)

Type V:

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

{

Statements;

}

n/2 power k

i<1(stops)

n/2 power k<1

n/2 power k=1

n=2 power k

k=log n base 2

So, O(log n)

For(i=0;i<n;i++) 🡪O(n)

For(i=0;i<n;i+2) 🡪O(n)

For(i=n;i>1;i--) 🡪O(n)

For(i=1;i<n;i=i\*2) 🡪O(log n base 2)

For(i=1;i<n;i=i\*3) 🡪O(log n base 3)

For(i=n;i>1;i=i/2) 🡪O(log n base 2)

Name of the Complexities:

1. Constant O(1)
2. Linear O(n)
3. Logarithmic O(log n)
4. Quadratic O(n^2)
5. Exponential O(2^n)

No iterations, then Constant O(1)

Eg: Array extracts first value for O(1)

def pick(data):

return a[0]

if \_\_name\_\_== ‘\_\_main\_\_’:

a=[100,20,30,45,90]

print(pick(a)) Output is 100

Eg: a=[10,12,13,100,16,10,1000,24] for O(log n)

For i in range(0,len(a),3);

Print(a[i]) Output is 10,100,1000

Space Complexity:

\*Parallel concept to Time Complexity.

\*Array of size n require O(n) Space.

\*Two Dimensional Array of size n\*n require O(n^2) Space.

\*Array allocates a single memory, so called sequential, where as Linked List can jump,it is not called sequential.

\*Linear Search O(1) space(best case) bcz space not depending on values we are searching, not looking at values.

\*Binary Search O(1)space(best case)

\*Merge Sort O(nlogn) space(best case)

\*Depth First Search O(n) space

\*Breadth First Search O(n) space

\*Dynamic Programming O(n^2) or O(n\*m)

Constant Space Complexity: Same amount of space regardless/irrespective of the input size n called Constant Space Complexity.

\*Linear Search are examples.Bcz Space not depending on array values.

Logarithmic Complexity: O(log n) takes space proportional to the log of the input size.

\*Quick Sort is example, bcz recursion takes extra space(stack)

Linear Complexity: O(n) When an algorithm takes space directly proportional to input size.

Log-Linear Complexity: O(nlogn) When the space complexity of an algorithm grows proportional to the input size and a logarithmic factor.

\*Merge sort is example, bcz one more loop externally added.

Polynomial Complexity: O(n^2)Space Complexity grows proportional to the square of the input size.

\*Nested List is Example.

\*Space Complexity of factorial recursion, sum recursion is O(n).