53,17 a=17 b=2 17%2 remd a=2

17) 53 (3
51
2) 17 (8
16
1) 2 (2
2
x

0 1 1

G.C.D(53,17) = 2

Q) Write a function that calculates the greatest common divisor of 2 numbers.

Q) (12) Write a program to find fibonacci series of n terms, where n is input by user:-

0 1 1 2 3 5 8 13 21 ... → first 2 numbers have to printed seperately.

** Note:- Don't try to repeatedly instantiate Scanner class, such as in a loop.

Because in Java, when the 'Scanner' is closed, that reads from 'System.in', it closes the 'System.in' also. So, next time when loop attempts to create a 'Scanner' on 'System.in' it fails, leading to:-
'NoSuchElementException'.

04/04/2024

Time and Space complexity

Q) What is Time complexity?

→ It is a relation between Input size & Running Time.

Input size depends on the variable being taken as input. So, more the input size, more the running time of operations.

Relation b/w Input size & running time can be linear, quadratic, cubic, log, sqrt, etc.

E.g → let's understand time complexity with an example.

Example 1

```
Public Static void main (String args[])
{
    Scanner sc = new Scanner (System.in);
    int n = sc.nextInt();
    for (int i=0; i<n; i++)
    {
        System.out.println("hello");
    }
}
```

→ Here, if the input is n, the code also runs for n times. So, relation is linear.

Input → n, time complexity, 9 → n.

Now, there can be 3 cases of time complexity.

(i) Best case (ii) Average case (iii) worst case.

(i) Best case

Numbers = [1, 2, 3, 4, 5]
Search for '1'

If we start searching from the front, get 1 in the first position.

So, 1 unit of time or 1 operation.

(ii) Average case

The numbers can be arranged in random ways like (2, 1, 3, 4, 5), (2, 3, 4, 1, 5), ...

So, best case if 1 is in the 1st position,
2nd best case if 1 is in the 2nd position .. and so on,

So, $\frac{1+2+3+4}{5}$

now, if generalise this for n numbers,

$$\frac{1+2+3+\cdots+n}{n} = \frac{n(n+1)}{2} \cdot \frac{1}{n} = \left(\frac{n+1}{2}\right)$$

So, for average case, $\left(\frac{n+1}{2}\right)$ unit of time. It's kind of a linear case only, since it's directly proportional to n.

(iii) worst case

worst case is if the searched number is at the last position.

So, $([5,4,3,2], 1)$ or $([5,4,2,3], 1)$ or, ..., etc.

Anyway, we need to perform 'n' operation to get our result. So, 'n' unit of time.

Representation

For best case, we can write $\Omega(1)$ → asymptotic lower bound

For average case, we can write $\theta\left(\frac{n+1}{2}\right)$

For worst case, we can write $O(n)$.

→ asymptotic upper bound.

Example 2

```
Public static void main (String[] args)
{
    Scanner sc = new Scanner (System.in);
    int n = sc.nextInt();
    for (int i=0; i<n; i++){
        for (int j=0; j<n; j++)
        {
            System.out.println ("hello");
        }
    }
```

→ So, here i runs from 0 to n-1, i.e n times.

For every iteration of i, j runs n times.

So, i=0, j=0, 1, 2, --, n-1.

i=1, j=0, 1, 2, --, n-1

i=n-1, j=0, 1, 2, --, n-1.

So, our code runs n×n times. So, in this case time complexity is $O(n^2)$.

Example 3

```
int n = sc.nextInt();
int m = sc.nextInt();
for (i=0; i<n; i++)
{
    for (j=0; j<m; j++)
    {
        S.o.Pln ("hello");
    }
}
```

So, for i=0, j= 0, 1, 2, 3, ---, m-1
i=1, j= 0, 1, 2, 3, ---, m-1
:
i= n-1, j=0, 1, 2, 3, ---, m-1.

Therefore, the code runs maximum for n×m times.
So, time complexity is O(n×m).

Example 4

```
Public static void main (String[] args)
{
Scanner sc = new Scanner (System.in);
int n = sc.nextInt();
int m = sc.nextInt();

for (int i=0; i<n; i++)
{
System.out.println("hello");
}
for (int j=0; j<m; j++);
{
System.out.println("hello");
}
}
```

→ n times

→ m times.

→ So, for i=0 to n-1, it will n times, whereas
for j=0 to m-1, it will run m times.

∴ Time complexity is O(n+m).

Now, if n = 10^6, m = 3, then we can say O(n+m) ≈ O(n).

compare :- $O(n)$ $O(n^2)$ $O(n^3)$

For n=1, 1 1 1

For n=2, 2 4 8

For n=3, 3 9 27

⋮

For $n = 10^5$, 10^5 10 10

best code — 2nd best code — worst code

Space Complexity

a) what is space complexity ?

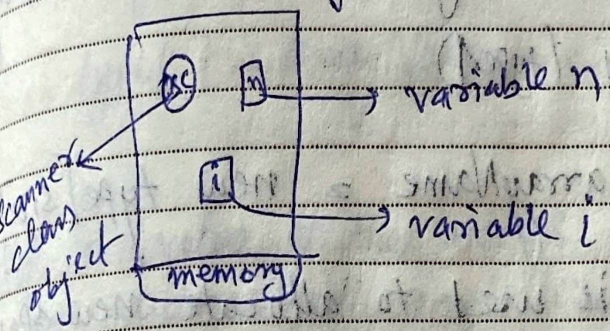
→ Space complexity defines the amount of space in the memory occupied by the code.

Let's take an example, Example 1

```
Public Static void main( String args[])
{
  Scanner sc = new Scanner (System.in);
  int n = sc.nextInt();

  for(int i=0; i<n; i++)
  {
    S.o.pln("hello");
  }
}
```

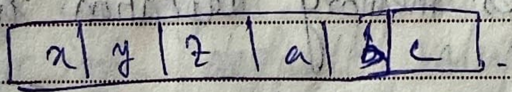
For this portion of code,

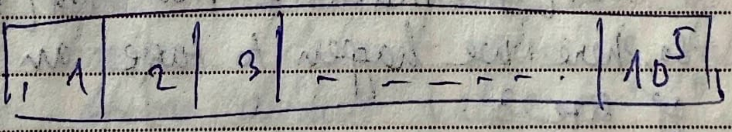


For this case, the space in the memory is constant. Even if n is very large, it still occupies than 1 space. So, in this case, space doesn't depend on the input size.

Example 2

Arrays -





In this case, the space in the memory is actually dependent on the input size. More the input size, more memory space it will occupy.

Note :- So, we can see in some case, the memory size doesn't depend on the input, whereas in other cases, it depends. We need to shorten the input size or optimize our code in those cases, where the performance of the code depends on the memory size.

Less the operating time of the code, more efficient the code is.

Less the memory size of the code, more efficient the code is.