Introduction :

The code in the solution 1 and 2 count the number of divisors for a given number

1) For the solution 1 we wrote a function that takes a parameter n and count a number of devisors for a given number by checking the n%d=0 ( if this number if divide d so d is a divisor of n ) after that we calculate the sum of d all these under the condition d<=n .
2) For the solution 2 we count the number of devisor's by doing the same thing in the solution 1 but in the different condition which ( d*d<=n) .

3) & 4) In general we get the follow estimation :

```
%%timeit
def count_divisors(n):
    count = 0
    d=1
    while d <= n :
        if n%d==0 :
            count+=1
        d+=1
    return count
```

```
140 ns ± 0.278 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
```

```
%%timeit
def count_divisors(n):
    count = 0
    d=1
    while d * d <= n :
        if n%d==0 :
            count+=1 if n/d == d else 2
        d+=1
    return count
```

```
138 ns ± 1.32 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
```

Wich means in general the solution 2 is faster than the first one .

For n = 10000 :

```
%%timeit
def count_divisors(n):
    count = 0
    d=1
    while d <= n :
        if n%d==0 :
            count+=1
        d+=1
    return count
count_divisors(10000)
```

```
2.64 ms ± 103 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
%%timeit
def count_divisors(n):
    count = 0
    d=1
    while d * d <= n :
        if n%d==0 :
            count+=1 if n/d == d else 2
        d+=1
    return count
count_divisors(10000)
```

```
33 µs ± 1.49 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

For n = 500000 :

```
%%timeit
def count_divisors(n):
    count = 0
    d=1
    while d <= n :
        if n%d==0 :
            count+=1
        d+=1
    return count
count_divisors(500000)
```

```
135 ms ± 2.79 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
%%timeit
def count_divisors(n):
    count = 0
    d=1
    while d * d <= n :
        if n%d==0 :
            count+=1 if n/d == d else 2
        d+=1
    return count
count_divisors(500000)
```

```
234 µs ± 6.31 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

Big-O notation:

1) T(n)=3n**3 + 2n**2 + ½ n  + 7

1) Methode 1

We have:

$$T(n) = 3n^3 + 2n^2 + \frac{1}{2}n + 7$$

if $n \geq 1$

since: $2n^2 \leq 2n^3$

$\frac{1}{2}n \leq \frac{1}{2}n^3$ and $7 \leq 7n^3$

So:

$$T(n) \leq 3n^3 + 2n^3 + \frac{1}{2}n^3 + 7n^3$$

$$\leq \frac{25}{2}n^3$$

So $\lambda = \frac{25}{2}$

$\Rightarrow T(n) = \Theta(n)$

Methode 2

We have: $\Theta(f(n)) \leq \lambda f(n)$

$$\lim \frac{T(n)}{f(n)} = \lim \frac{3n^3 + 2n + \frac{1}{2}n + 7}{n^3}$$

$$= 3$$

$\lambda = 3$

So $T(n) = \Theta(n^3)$

2)

To show that n**k **is not O(n****(k-1)), we need to show that there does not exist a positive constant C such that n****k <= Y * n****(k-1) for all n greater than some n0.

Assume that n****k is O(n****(k-1)), then by definition of big-O notation, there exists a constant C > 0 and a positive integer n0 such that:

n****k <= Y * n****(k-1) for all n >= n0

Dividing both sides by n**(k-1), we get:

n <= Y

However, this is a contradiction because n is an unbounded variable that can grow arbitrarily large. Therefore, there is no constant C that can satisfy the above inequality for all n >= n0, and we have shown that n****k is not O(n****(k-1)).

## Multiplication of matrices :

## With C

```
#include<stdio.h>



   void multiply(int r1, int c1, int r2, int c2);



   int main()

   {

    int i,j,k,r1,c1,r2,c2;
```

```
 printf("Enter row and column of first matrix\n");
 scanf("%d%d", &r1, &c1);
 printf("Enter row and column of second matrix\n");
 scanf("%d%d", &r2, &c2);
 multiply(r1,c1,r2,c2);
 return 0;
}

void multiply(int r1, int c1, int r2, int c2)
{
 int i,j,k;
 float a[10][10], b[10][10], mul[10][10];
 if(c1==r2)
 {
  printf("Enter elements of first matrix:\n");
  for(i=0;i< r1;i++)
  {
   for(j=0;j< c1;j++)
   {
    printf("a[%d][%d]=",i,j);
    scanf("%f", &a[i][j]);
   }
  }
  printf("Enter elements of second matrix:\n");
```

```
for(i=0;i< r2;i++)
{
 for(j=0;j< c2;j++)
 {
  printf("b[%d][%d]=",i,j);
  scanf("%f", &b[i][j]);
 }
}
for(i=0;i< r1;i++)
{
 for(j=0;j< c2;j++)
 {
  mul[i][j] = 0;
  for(k=0;k< r2;k++)
  {
   mul[i][j] = mul[i][j] + a[i][k]*b[k][j];
  }
 }
}
printf("Multiplied matrix is:\n");
for(i=0;i< r1;i++)
{
 for(j=0;j< c2;j++)
 {
```

```
printf("%f\t", mul[i][j]);
}
printf("\n");
}
}
else
{
printf("Dimension do not match for multiplication.");
}
}
```

***QUIZ***

1) The time complexity for the following fragment is :

A) O(n)

2) The time complexity for the following fragment is :

D) O(log_$_k$(n))

3) The time complexity for the following fragment is :

C) O(n*m)