BIG OH NOTATIONS PRACTICE:

Big-O Notation gives an upper bound of the complexity in the worst case, helping quantify performance as the input size becomes arbitrarily large.

- O(n+c) → O(n) we neglect constant
- O(nc) → O(n) we neglect constant
- Big-Oh always depends on highest exponential.
  e.g : $15n^2 + 6n^3 \rightarrow O(n^3)$

O(1):

```
a := 1
b := 2
c := a+5*b
```

SINCE these are only assignment statements, its time complexity will be O(1)

```
i := 0
while i<10 do
      i := i + 1
```

Loop is independent of n

O(n):

```
i := 0
while i<n do
      i := i + 1
```

f(n) = n

O(f(n)) = O(n)

```
i := 0
while i<n do
      i := i + 3
```

f(n) = n/3

O(f(n)) = O(n)

$O(n^2)$:

```
for (i:=0;i<n;i++)
    for (j:=0;j<n;j++)
```

$f(n) = n*n = n^2$

$O(f(n)) = O(n^2)$

```
for (i:=0;i<n;i++)
    for (j:=i;j<n;j++)
```

Suppose n = 5

i	j	total j
0	0-4	n
1	1-4	n-1
2	2-4	n-2
3	3-4	n-3
4	4	1

n+(n-1)+(n-2)+(n-3) +….. 1 (acc to Sum of AP → $(n^2+n)/2$

→$O(n^2)$

O(logn):

For binary search, complexity is O(logn)

$O(2^n)$:

```
Int fun(int  n)
{
 if (n<= 1) return n;
 return fun(n-1)+fun(n-1);
}
```

If we have a number that starts by performing one operation and then doubles the number of operations performed with each iteration, then the number of operations performed in the nth iteration is 2^n.