to store a[i] = n words
to store n = 1 word
to store i & s = 2 words

∴ space complexity = n+3

* O - notation (upper bound)
F(n) = O(g(n))

In this F(n) lies on or below c g(n) where c is +ve constant.

Big O gives us a formal way of expressing upper bound.

* Ω - notation (lower bound)
F(n) = Ω (g(n))
Ω - omega.

in this F(n) on or above c g(n) where c is +ve constant.
omega gives us a Formula way of expressing lower bound.

* θ - notation (same order)
F(n) = θ (g(n))
In this F(n) lies between c_1.g(n) & c_2.g(n) where c_1 & c_2 are constant.
the theta notation is more precise than the both big oh & omega notation

$S(P) = c + Sp(\text{instance})$

$S(P) \rightarrow$ space complexity
$c \rightarrow$ fixed part
$Sp(\text{instance}) \rightarrow$ variable part

example ①

1) Algorithm abc(a,b,c)

```
{
  return a+b+b*c + (a+b+c) /(a+b) + 4.0;
}
```

For every instance 3 words are required to store variables : a, b & c

$\therefore$ space complexity = 3.

example ②

Algorithm sum (a[ ], n)

```
{
  s = 0;
  for i = 1 to n do
    s = s + a[i];
  return s;
}
```