* Hamming Weight ? 1011

The number of 1's / Set bits in the binary of any number is it's hamming weight.

5, 0, 11, 8 (2, 0, 3, 1)

```
int hammingWeight( int n) {

}
```

n = 11 (int) → 1011

0101
1000

* Constraints → Don't use %

n % 2 == 0

~~128 64 32 16 8 4 2~~ (1)   4+2+1 = 7

if n&1 == 1

1 0 0 0
1 1 1

0001
0101
0001

11 % 2 == 0
11 / 2 = 5
5 % 2 == 0
5 / 2 = 2
2 % 2 == 0
2 / 2 = 1
1 % 2 == 1
1 / 2 = 0 Stop

count = 0
c++ → 1
c++ → 2
c++ → 3

101X

n&1

[1011] ✓
0001
0001 → T

n = n >> 1

101 ✓
001
001

10 ✓ →
01
00

count++ = 1
++ = 2
++ = 3

1 → 0
1 stop
1

Object Oriented Programming

* Benefits:
  - → Data Security
  - → Memory Efficiency
  - → Code Readability
  - → Code Reusability
  - → Time Efficiency
  - → Can be used to solve real world problems.
  - → Scalability
  - → Robust Code Structure
  - → Proper relation between the entities / data members.

"this" keyword refers to the instance variables of the class.

Overloading → changing the no. of parameters
→ changing the return-type of the parameters.

Entity → Employee → class

String → name
String → email
int → salary
} → Attributes → fields → Properties

employeeDetails ( ) → methods

Because, it describes the behaviour of the object.

* A class is a template or blueprint to create objects or instances of an application. It is just a prototype and has no memory of it's own.

* Employee e1 = new Employee( );
  e1 ↳ object

Instance Key ← e1. ⇉ [properties or method]

* An object is an instance (key) of the class. With the help of the object we can access all the data inside the class. The object occupies the heap memory.

DRY → Don't Repeat Yourself

1 → 5 → 3 → (2)
2 → 10 → 3
100 → 500 lines → 200

O(n²)
O(n)
O(log n)
O(n log n)

[Optimization]

final keyword → variable → constant / no change
final class → can't inherit
final parameter → constant / no change
final method → no override allowed

c = T

```
final void display( ) {
    Sout("BIET");        → override ✗
}
```

4 Pillars of OOPs :→

① Encapsulation → Wrapping the code & the data members inside a block [Class], so that they are not accidentally modified, is called Encapsulation. It is achieved by using the "private" access modifier.

To access them outside the class we use two public methods :

(i) getters ↳ retrieve / fetch attributes

(ii) setters ↳ setting / assigning attributes