Q1. 5 is the decimal number
2 | 5
—----
2 | 2 1
—----
2 | 1 0
Reading numbers is reverse order gives us 101 as the binary representation

2. Number 10
2 | 10
—------
2 | 5 0
—------
2 | 2 1
—------
2 | 1 0
Reading in reverse order from bottom to top gives us 1010 as the binary representation of 10.

Q2. Number 15. Converting it in binary 1111, as its more than 1 set bits in the binary representation therefore it's not power of 2.

Number 32. Converting it to binary 10000, as it has only 1 set bit, therefore its power of 2.

Q3. Solution:
Code link: ASS_CODE1.java
Output :

```
Enter the integer:
15
Given number is odd
```

Approach :

- We know that any odd number in its binary representation has the last digit (from right) as 1.
- We used that concept and simply used the AND operator and operated the given number with 1.
- We know the binary representation of 1 is also 1 and if the result of AND operator is 1, we can be sure that the given number is also odd since AND only returns 1 when both the operating bits are 1.

Q4. Solution:
Code link: ASS_CODE2.java
Output:

```
Enter the integer:
5
The number of set bits in the given number are 2
```

Approach:

- We have extracted the last digit (from right), of the number using the AND operator. By operating AND operator on 1 and the number the corresponding bits of 1 and the number will be ANDed.
- Whatever will be the result of AND we will add it to our "count" variable because the result can only be 0 and 1. So if there is a set bit it will automatically get summed up in the variable "count".
- Once we are done with the last digit, we need to check upon other digits as well.

- We know the right shift will remove the last bit from the right end and a new bit will take its place.
- This way we can check for each bit whether it is set or not.

Q5. Solution:
Code link: ASS_CODE3.java
Output:

```
Enter the number of elements you want to store: 5
Enter the elements of the array:
3 4 4 1 1
The odd occurring element is 3
```

Approach:
Some key observations about xor operator:

- XOR of 'x' with 0:
- x ^ 0 = x
- XOR of 'x' with itself even number of times:
- x ^ x = 0
- XOR of 'x' with itself odd number of times:
- (x ^ x ^ x) = (x ^ (x ^ x)) = (x ^ 0) = x
- (x ^ x ^ x ^ x ^ x) = (x ^ (x ^ x) ^ (x ^ x)) = (x ^ 0 ^ 0) = x
- This shows taking xor of the same number an odd number of times results in the number itself whereas taking xor even number of times will result in 0.
- So, if we take XOR of all array elements, even appearing elements will cancel each other, and we are left with the only odd appearing element.
- We will simply return that element.