Maths for DSA

Bit-Manipulation :

And Operator (&):

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

Observation : A & 1 = A, for example:

110010100 & 111111111 = 110010100

Or Operator (|):

A	B	A
0	0	0
0	1	1
1	0	1
1	1	1

Xor Operator (^):

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Observation :

1. A ^ 1 = ~ A
2. A ^ 0 = A
3. A ^ A = 0

Left Shift Operator (<<) :

$(10)_{10} = (1010)_2$

$(1010)_2 << 1 = (10100)_2$

$(10100)_2 = (1(2^4) + 1(2^2))_{10} = (16 + 4)_{10} = (20)_{10}$

$\therefore a << 1 = 2a$

$\therefore a << b = a(2)^b$

Right Shift Operator (>>) :

$(25)_{10} = (11001)_2$
$(11001)_2 >> 1 = (1100)_2 = (12)_{10}$

$\therefore a >> b = \frac{a}{2^b}$

Complement Operator (~) :

$[\sim 100]_2 = [011]_2$

Number Systems :

1. Decimal (base 10) → 1, 2, 3, 4, 5, 6, 7, 8, 9
2. Binary (base 2) → 0, 1
3. Octal (base 8) → 0, 1, 2, 3, 4, 5, 6, 7
4. Hexadecimal (base 16) → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

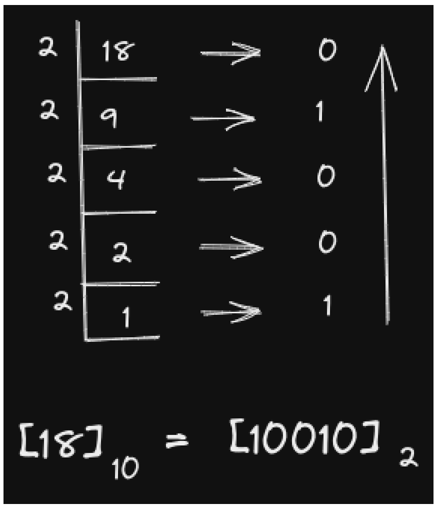
Conversions :

Decimal → any base

Steps :

1. Keep dividing by the base
2. Take remainders
3. Write the remainders in reverse

Example 1 :



Any base → Decimal

Step : Multiply & add the power of base with the digits

Example 1 :

$[10010]_2 = [1(2^4) + 0(2^3) + 0(2^2) + 1(2^1) + 0(2^0)]_{10}$
$= [2^4 + 2^1]_{10}$
$= [16 + 2]_{10}$
$= [18]_{10}$

Example 2 :

$[22]_8$
$= [2(8^1) + 2(8^0)]_{10}$
$= [16 + 2]_{10}$
$= [18]_{10}$

Q1. Check if a number is even / odd.

$(101)_2$ is odd, $(100)_2$ is even.

The last digit of binary determines the number will be even or odd.

How to get the last digit of binary ?

any binary & 00...01 = last digit of Binary (also called Least Significant Bit)

Example :

$(100101)_2$ & $(000001) = (1)_2$

```java
package com.inclass;

public class OddEven {
    public static void main(String[] args) {
        int n = 66;
        System.out.println(oddCheck(n));
    }
    static boolean oddCheck(int n) {
        return (n & 1) == 1;
    }
}

/* Output : false
```

Q2. Every number appears twice but one number appears once, Find the number.

Xor operator follows associative property.

i.e. A ^ (B ^ C) = (A ^ B) ^ C

like if arr = {-1, 3, -4, -2, 2, 4, 1},

we will find the sum of all the elements.

similarly arr = {1, 3, 4, 2, 2, 4, 1}

we will find the XOR of all the elements.

```java
package com.inclass;

public class SingleInDuplicate {
    public static void main(String[] args) {
        int[] arr = {1, 3, 2, 4, 4, 2, 1};
        System.out.println(xor(arr));
    }
    static int xor(int[] arr) {
        int xorSum = arr[0];
        for (int i = 1; i < arr.length; i++) {
            xorSum = xorSum ^ arr[i];
        }
        return xorSum;
    }
}

/* Output : 3
```

Q3. Find the i^{th} bit of a number.

$(100101)_2$ & $(000100)_2 = (1)_2$

000100 is called the mask.

Let the initial mask be 1, to find the i^{th} mask, which is mask with (i) zeroes.

we will use left shift operator i times.

```java
package com.inclass;

public class BitIndex {
    public static void main(String[] args) {
        int n = 12;
        int index = 2;
        System.out.println(bit(n, index));
    }
    static int bit(int n, int index) {
        int mask = 1 << (index);
        return (n & mask) >> (index);
    }
}

/* Output : 1
```

$[12]_{10} = [2^3 + 2^2]_{10} = [1100]_2$

Q4. Set the i^{th} bit.

Set : 0 → 1, 1 → 1

$[1010]_2 \mid [100]_2 = [1110]_2$

```java
package com.inclass;

public class SetBit {
    public static void main(String[] args) {
        int n = 10;
        System.out.println(setBit(n, 2));
        System.out.println(setBit(n, 3));
    }
    static int setBit(int n, int index) {
        int mask = 1 << index;
        return (n | mask);
    }
}

/* Output : 14 , 10
```

Q5. Reset the i^{th} bit.

Reset : 0 → 1, 1 → 0

$[1010]_2$ & $[1101]_2 = [1000]_2$

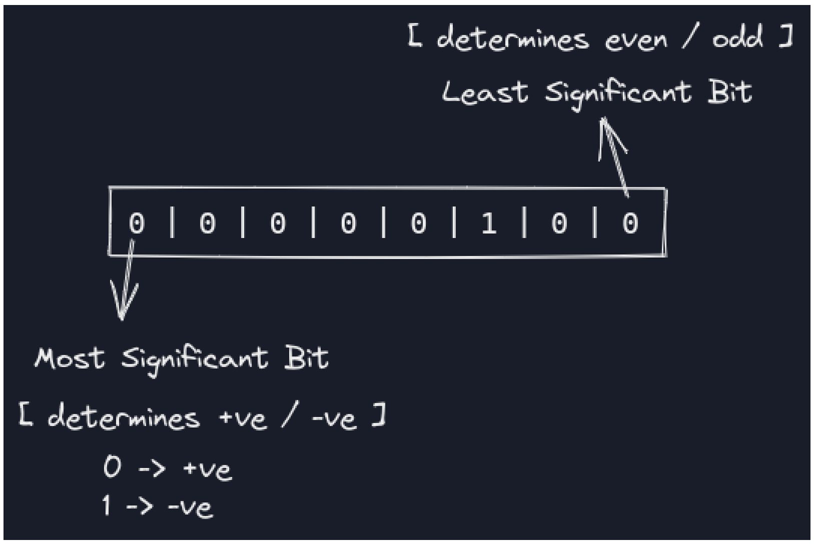
mask = ~ $[0010]_2 = [1..101]_2$

```
package com.inclass;

public class ResetBit {
    public static void main(String[] args) {
        int n = 10;
        System.out.println(resetBit(n, 2));
        System.out.println(resetBit(n, 3));
    }
    static int resetBit(int n, int index) {
        int mask = 1 << index;
        return (n & (~ mask));
    }
}

/* Output : 10 , 2
```

Negative Binary Numbers :

Steps for base-2 complement:

1. Complement of Number
2. Add 1



For Example :

$[10]_{10} = [00001010]_2$

1. $\sim[00001010]_2 = [11110101]_2$
2. $[11110101]_2 + 1 = [11110110]_2$

$[11110110]_2 = [-10]_{10}$

Why ?

Generally, to find a negative of a number we subtract the number from 0,

So, $[00000000]_2 - [00001010]_2$

We know that 1 byte = 8 bits, and if a number exceeds size of 1 byte then only 8 bits are stored and the rest becomes garbage.

$[00000000]_2 = [100000000]_2$

$= [11111111]_2 + [1]_2,$

$\therefore [11111111]_2 - [00001010]_2 + [1]_2$

$\Rightarrow [\sim 00001010]_2 + [1]_2$

Range of 1 Byte

- 1 byte contains 8 bits containing 0 & 1.

8	7	6	5	4	3	2	1
0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1

- 1 byte can contain a range of 256 numbers.
- 1 byte does not contain -128 to +128 (because it contains 257 numbers including 0)
  1 byte contain -128 to +127. Why ??

Because negative of 0 is 0.

$\sim[0000000]_2 + [1]_2$

$= [11111111]_2 + [1]_2$

$= [00000000]_2$ (number exceeds 1 byte, so 1 is discarded)

- **Range formula for n bits** :

$$[-2^{n-1}] \; to \; [2^{n-1} - 1]$$

Q6. Find the position of the rightmost set bit.

Let n be 10, $[10]_{10} = [1010]_2$

So, -n = $[\ 1010]_2 + [1]_2 = [0110]_2$

$\therefore n \ \& \ [-n] = [1010]_2 \ \& \ [0110]_2 = [0010]_2 = [2]_{10}$

```java
package com.inclass;

public class RightMostSetBit {
    public static void main(String[] args) {
        int n = 10;
        System.out.println(rightmostSetBit(n));
    }

    static int rightmostSetBit(int n) {
        return n & ((~ n) + 1);
    }
}

/* Output : 2
```

Q7. Find the unique element out of an array of triple elements.

Let arr = {2, 2, 3, 2, 7, 7, 8, 8, 7, 8},

Adding all the elements in binary,

						1	0
						1	0
						1	1
						1	0
					1	1	1
					1	1	1
				1	0	0	0
				1	0	0	0
					1	1	1
				1	0	0	0
				3	3	7	4

As the numbers are repeating thrice, $\therefore 3374 \ \% \ 3 = [0011]_2 = [3]_{10}$

```java
package com.inclass;

import java.util.Arrays;

public class UniqueInTriplets {
    public static void main(String[] args) {
        int[] arr = {10, 2, 2, 2, 2};
        System.out.println(countSetBit(arr, 4));
    }

    static int countSetBit(int[] arr, int count) {
        int[] setBit = new int[8];
        for (int i = 0; i < setBit.length; i++) {
            int temp = 0;
            for (int j = 0; j < arr.length; j++) {
                if ((arr[j] & (1 << i)) == (1 << i)) {
                    temp += 1;
                }
            }
            setBit[7 - i] = temp % count;
        }
        System.out.println(Arrays.toString(setBit));
        int result = 0;
        for (int i = 0; i < setBit.length; i++) {
            result += setBit[7 - i] << i;
        }
        return result;
    }
}

/* Output : [0, 0, 0, 0, 1, 0, 1, 0]
                                                        10
```

Q8. Find the n^{th} Magic Number.

1 → 001 → 5

2 → 010 → 25

3 → 011 → 30

4 → 100 → 125

5 → 101 → 130

```java
package com.inclass;

public class MagicNumber {
    public static void main(String[] args) {
        int n = 5;
        int base = 5;
        System.out.println(magicNo(n, base));
    }

    static int magicNo(int n, int base) {
        int magic = 0;
        for (int i = 0; i < 8; i++) {
            if ((n & (1 << i)) == (1 << i)) {
                magic += Math.pow(base, i + 1);
            }
        }
        return magic;
    }
}

/* Output : 130
```

Q9. Find the no of Digits of a Binary Number.

```
package com.inclass;

public class DigitsBinary {
    public static void main(String[] args) {
        int n = 8;
        System.out.println(leftShift(n));
        System.out.println(logBase2(n));
    }

    static int leftShift (int n) {
        for (int i = 7; i >= 0; i--) {
            if ((n & (1 << i)) == (1 << i)) {
                return i + 1;
            }
        }
        return 0;
    }

    static int logBase2 (int n) {
        return (int) (Math.log(n) / Math.log(2)) + 1;
    }
}

/* Output : 8  8
```

Q10. Find the sum of n^{th} row in Pascal's Triangle.

1						1
1	1					2
1	2	1				4
1	3	3	1			8
1	4	6	4	1		16
1	5	10	10	5	1	32

```
package com.inclass;

public class SumofRowofPascelTriangle {
    public static void main(String[] args) {
        int row = 5;
        System.out.println(power(row));
    }
    static int power(int row) {
        return 1 << (row + 1);
    }
}

/* Output : 64
```

Q11. Find the number is the power of 2 or not.

```
package com.inclass;

public class Powerof2 {
    public static void main(String[] args) {
        int n = 2;
        System.out.println(powerCheck(n));
    }

    static boolean powerCheck(int n) {
        return (n & (n - 1)) == 0;
    }
}

/* Output : true
```

Q12. Find a^b .

```
package com.inclass;

public class Power {
    public static void main(String[] args) {
        int base = 3;
        int power = 6;
        System.out.println(power(base, power));
    }

    static int power(int base, int power) {
        int ans = 1;
        while(power > 0) {
            if ( (power & 1) == 1) {
                ans = ans * base;
            }
            base *= base;
            power = power >> 1;
        }
        return ans;
    }
}

/* Output : 729
```

Q13. Find the no. of set bits.

If n = $[9]_{10} = [1001]_2$, Then Answer = 2.

If n = $[13]_{10} = [1101]_2$,

1. $-n = [0011]_2, n \ \& \ -n = [0001]_2$

   $n = n - [n \ \& \ -n] = [1100]_2$

2. $-n = [0100]_2, n \ \& \ -n = [0100]_2$

   $n = n - [n \ \& \ -n] = [1000]_2$

```java
package com.inclass;

public class NoofSetBits {
    public static void main(String[] args) {
        int n = 10;
        System.out.println(rightShift(n));
        System.out.println(and(n));
    }

    static int rightShift (int n) {
        int count = 0;
        while (n > 0) {
            if ((n & 1) == 1) {
                count += 1;
            }
            n = n >> 1;
        }
        return count;
    }

    static int and (int n) {
        int count = 0;
        while (n > 0) {
            count++;
            int temp = n & ((~n) + 1);
            n -= temp;
        }
        return count;
    }
}

/* Output : 2
                                                 2
```

Q14. Find the XOR Factorial.

```java
package com.inclass;

public class XorFactorial {
    public static void main(String[] args) {
        int low = 3;
        int high = 6;
        System.out.println(xorFactorial(low, high));
    }

    static int xorFactorial(int low, int high) {
        return (low - 1) & high;
    }
}
```

```
/* Output : 2
```

Q15. Flipping an image.

```
package com.inclass;

// https://leetcode.com/problems/flipping-an-image/

import java.util.Arrays;

public class Flipping {
    public static void main(String[] args) {
        int[][] image = {
                {1, 1, 0},
                {1, 0, 1},
                {0, 0, 0},
        };
        flip(image);
    }
/*
Runtime: 0 ms, faster than 100.00% of Java online submissions for Flipping an Image.
Memory Usage: 42.3 MB, less than 12.24% of Java online submissions for Flipping an Image.
 */
    static int[][] flip (int[][] image) {
        for (int i = 0; i < image.length; i++) {
            for (int j = 0; j < image[i].length / 2; j++) {
                int temp = image[i][image[i].length - j - 1];
                image[i][image[i].length - j - 1] = image[i][j];
                image[i][j] = temp;
            }
            System.out.println(Arrays.toString(image[i]));
            for (int j = 0; j < image[i].length; j++) {
                image[i][j] = image[i][j] ^ 1;
            }
            System.out.println(Arrays.toString(image[i]));
        }
        return image;
    }
}
/* Output:
                                                [0, 1, 1]
                                                [1, 0, 0]
                                                [1, 0, 1]
                                                [0, 1, 0]
                                                [0, 0, 0]
                                                [1, 1, 1]
```

Q16. Prime Numbers (Sieve of Eratosthenes)

```
package com.inclass;

public class Prime {
    public static void main(String[] args) {
        int n = 40;
        arrayBoolean(n);
                }
    static void arrayBoolean(int n) {
        boolean[] bool = new boolean[n - 2];
        for (int i = 2; i * i ≤ n; i++) {
            for (int j = i * 2; j < n; j++) {
                if (!bool[j - 2] && j % i == 0) {
                    bool[j - 2] = true;
                }
            }
            printArrayBoolean(bool);
        }
        printArrayBoolean(bool);
    }

    static void printArrayBoolean(boolean[] bool) {
        for (int j = 0; j < bool.length; j++) {
            if (!bool[j]) {
                System.out.print(j + 2 + ", ");
            }
        }
        System.out.println();
    }
}

/* Output :

2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
2, 3, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35, 37,
2, 3, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35, 37,
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
```

Time Complexity Analysis :

$\frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \frac{n}{7} \ldots.$

$\Rightarrow n(\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} \ldots.)$ (Harmonic progression of Primes)

$\Rightarrow n * log(log(n))$

Q17. Square Root of a Number

- By using Binary Search,

```
package com.inclass;

public class SquareRoot {
    public static void main(String[] args) {
        int n = 40;
        int p = 5;
        System.out.println(root(n, p));
    }

    static double root (int n, int p) {
        int start = 1;
        int end = n;
        double root = 0;
        while (start ≤ end) {
            int m = start + (end - start) / 2;
            if (m * m == n) {
                return m;
            } else if (m > n) {
                start = m + 1;
            } else {
                end = m - 1;
                root = m;
            }
        }

        double decimal = 0.1;
        for (int i = 0; i < p; i++) {
            while (root * root ≤ n) {
                root += decimal;
            }
            root -= decimal;
            decimal /= 10;
        }

        return root;
    }
}

/* Ouput : 6.324549999999992
```

- By using Newton Raphson Method $X \rightarrow Assumed\ square\ root N \ \rightarrow Number$

$$\sqrt{N} = \frac{1}{2}[\ X + \frac{N}{X}\ ]$$

Why ? If $X = \sqrt{N}$, then $\frac{1}{2}[\ \sqrt{N} + \frac{N}{\sqrt{N}}\ ] = \sqrt{N}$

```
package com.inclass;

public class SquareRoot {
    public static void main(String[] args) {
        int n = 40;
        System.out.println(newtonRaphson(n));
    }

    static double newtonRaphson (int n) {
        double x = n / 2;
        double error = 10;
        while (error > 1) {
            double root = 0.5 * (x + n / x);
            error = Math.abs(root - x);
            x = root;
        }
        return x;
    }
}

/* Output: 6.392010163749294
```

- Time Complexity Analysis:

$$T \ = \ O(log(n) * f(n))$$

$f(n)$ → cost of calculating $f(n)/f'(n)$ with n-digit precision.

Q18. Factors of a number.

If $n = 20, \Rightarrow 1, 2, 4, 5, 10, 20$

```java
package com.inclass;

import java.util.ArrayList;
import java.util.Arrays;

public class Factors {
    public static void main(String[] args) {
        int n = 20;
        bruteForce(n);
        checkRoot(n);
    }

    static void bruteForce (int n) {
        for (int i = 1; i <= n; i++) {
            if (n % i == 0) {
                System.out.print(i + ", ");
            }
        }
        System.out.println();
    }

    static void checkRoot (int n) {
        ArrayList<Integer> list = new ArrayList<> (0);
        for (int i = 1; i <= Math.sqrt(n); i++) {
            if (n % i == 0) {
                System.out.print(i + ", ");
                if (n / i != i) {
                    list.add(n / i);
                }
            }
        }
        for (int i = 1; i <= list.size(); i++) {
            System.out.print(list.get(list.size() - i) + ", ");
        }
        System.out.println();
    }
}
```

Modulo Properties

1. $[a + b] \ \% \ m = [[a \ \% \ m] + [b \ \% \ m]] \ \% \ m$
2. $[a - b] \ \% \ m = [[a \ \% \ m] - [b \ \% \ m]] \ \% \ m$
3. $[a * b] \ \% \ m = [[a \ \% \ m] * [b \ \% \ m]] \ \% \ m$
4. $\frac{a}{b} \ \% \ m = [[a \ \% \ m] + [b^{-1} \ \% \ m]] \ \% \ m$, where $b^{-1} \ \% \ m$ is multiplicative inverse modulo.

Q19. Highest Common Factor

$$Euclid\ Root = [a \ \% \ b] \ / \ b$$

```java
package com.inclass;
```

```java
public class EuclidHCF {
    public static void main(String[] args) {
        int a = 18;
        int b = 8;
        System.out.println(recursion(a, b));
        System.out.println(lcm(a, b));
    }

    static int euclid (int a, int b) {
        if (a < b) {
            int temp = a;
            a = b;
            b = temp;
        }
        while (a % b ≠ 0) {
            int temp = a;
            a = a % b;
            b = temp;
        }
        return b;
    }

    static int recursion (int a, int b){
        if (a == 0) {
            return b;
        }
        if (a > b) {
            return recursion(a % b, a);
        }
        return recursion(b % a, a);
    }

    static int lcm (int a, int b) {
        return a * b / recursion(a, b);
    }
}
```

Q20. Die-Hard Bucket Problem

Will two containers of 3 L and 5 L, create 4 L ?

$3x + 5y = 4 \ \Rightarrow 1(3x + 5y) = 4$

$\therefore$ 4 is divisible by 1 (which is the HCF of 3 L and 5 L buckets)

$\therefore$ Yes

```java
package com.inclass;

public class DieHard {
    public static void main(String[] args) {
        int[] arr = new int[] {8, 18};
        int k = 5;
        System.out.println(bucket(arr, k));
    }

    static boolean bucket (int[] arr, int k) {
        for (int i = 1; i < arr.length; i++) {
            if (arr[i - 1] < arr[i]) {
                int temp = arr[i - 1];
                arr[i - 1] = arr[i];
                arr[i] = temp;
            }
            while (arr[i - 1] % arr[i] ≠ 0) {
                int temp = arr[i - 1];
                arr[i - 1] = arr[i - 1] % arr[i];
                arr[i] = temp;
            }
        }
        return (k % arr[arr.length - 1] == 0);
    }
}
```