BIG-O NOTATION

BIG-O NOTATION

THIS EXPRESSES THE COMPLEXITY OF AN ALGORITHM

AN ALGORITHM WHOSE COMPLEXITY DOES NOT CHANGE WITH THE INPUT SIZE IS O(1)

THE ALGORITHM IS SAID TO HAVE CONSTANT TIME COMPLEXITY

IT TAKES THE SAME AMOUNT OF TIME EVEN IF THE INPUT SIZE IS DOUBLED, TRIPLED OR INCREASED TO ANY LEVEL

IF "N" IS THE SIZE OF THE INPUT....

THE COMPLEXITY OF AN ALGORITHM IS O(N) IF THE TIME TAKEN BY THE ALGORITHM INCREASES LINEARLY WHEN N INCREASES

THE COMPLEXITY OF AN ALGORITHM IS $O(N^2)$ IF THE TIME TAKEN BY THE ALGORITHM INCREASES QUADRATICALLY WHEN N INCREASES

WHAT IS THE COMPLEXITY OF COMMON OPERATIONS?

THE COMPLEXITY OF AN ALGORITHM IS O(N) IF THE TIME TAKEN BY THE ALGORITHM INCREASES LINEARLY WHEN N INCREASES

THE COMPLEXITY OF AN ALGORITHM IS $O(N^2)$ IF THE TIME TAKEN BY THE ALGORITHM INCREASES QUADRATICALLY WHEN N INCREASES

LOWER ORDER TERMS AND CONSTANTS DO NOT MATTER WHILE EXPRESSING COMPLEXITY, THE ASSUMPTION IS THAT N IS VERY LARGE

$O(N^2 + 1000)$ IS EQUIVALENT TO $O(N^2)$

$O(N^2 + N)$ IS EQUIVALENT TO $O(N^2)$

WHICH ALGORITHMS ARE FASTER?

TIME TAKEN

$O(1) < O(N) < O(N^2) < O(N^3)$

FASTEST SLOWEST

WHAT ARE THE COMPLEXITIES OF THE FOLLOWING PIECES OF CODE?

```java
public static void simpleFunction(int n) {
    int a = 9;
    int b = 3;

    int sum = a + b + n;
    int product = a * b * n;
    int quotient = a * n/ b;

    System.out.println(String.format(
            "The sum is: %s, product is: %s and quotient is: %s", sum, product, quotient));
}
```

HINT: NOTE THAT WE DON'T CARE ABOUT THE ACTUAL NUMBER OF OPERATIONS, COMPLEXITY IS BASED ON THE SIZE OF THE INPUT

THIS IS A CONSTANT TIME OPERATION, O(1). THE CODE TAKES THE SAME TIME WHATEVER THE VALUE OF N, IT USES THE VALUE OF N, RATHER THAN USE IT AS A SIZE OF INPUT

```java
public static void singleForLoop(int n) {
    for (int i = 0; i < n; i++) {
        System.out.println(String.format("Square of %s is %s", i, Math.pow(i, 2.0)));
    }
}
```

HINT: THE NUMBER OF OPERATIONS OBVIOUSLY CHANGES WITH THE SIZE OF THE INPUT

THE COMPLEXITY OF THIS OPERATION IS O(N), THE OPERATIONS CHANGE LINEARLY WITH THE SIZE OF INPUT

IF THE VALUE OF "N" DOUBLES, THE CODE WILL TAKE ROUGHLY TWICE AS LONG

```java
public static void singleWhileLoop(int n) {
    int i = 0;
    while (i < n) {
        System.out.println(String.format("Square of %s is %s", i, Math.pow(i, 2.0)));
        i++;
    }
}
```

HINT: THIS IS CONCEPTUALLY SIMILAR TO THE PREVIOUS ONE

THE COMPLEXITY OF THIS OPERATION IS O(N), THE OPERATIONS CHANGE LINEARLY WITH THE SIZE OF INPUT

IF THE VALUE OF "N" DOUBLES, THE CODE WILL TAKE ROUGHLY TWICE AS LONG

```java
public static void ifStatement(int n) {
    if (n % 2 == 0) {
        System.out.println("The input is even");
    } else {
        for (int i = 0; i < n; i++) {
            System.out.println("Printing: " + i);
        }
    }
}
```

HINT: COMPLEXITY ANALYSIS IS BASED ON THE WORST CASE SCENARIO

THE COMPLEXITY OF THIS OPERATION IS O(N), THE WORST CASE IS THAT THE INPUT IS ODD AND WE ENTER THE FOR-LOOP

IN THE FOR LOOP IF THE VALUE OF "N" DOUBLES, THE CODE WILL TAKE ROUGHLY TWICE AS LONG

```java
public static void nestedForLoop(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            System.out.println(String.format("Product of %s and %s is %s", i, j, i * j));
        }
    }
}
```

HINT: A SINGLE LOOP OF N IS O(N), TWO NESTED LOOPS WILL BE OF HIGHER COMPLEXITY

THE COMPLEXITY OF THIS OPERATION IS $O(N^2)$, AS N CHANGES THE NUMBER OF OPERATIONS CHANGE QUADRATICALLY

FOR EVER ITERATION OF THE OUTER LOOP THE INNER LOOP ITERATES N TIMES SO THE STATEMENT IS CALL N*N TIMES = N^2