Lecture 13

Time and Space Complexity

What did we discuss in last class?

- Programs don’t take the same time in all the systems to run - Program A could run in 1.5s in System A and can take 1 hr on System B.
- So we can’t measure the runtime of a program in seconds/minutes/hours.
- So what do we do?
- We measure the maximum number of instructions a program is going to execute to measure its runtime complexity.
- We assume that each instruction takes 1 unit of time to execute.
- We try to identify the worst case time complexity - Why? —- Because we are always measuring the maximum number of instructions executed.

Constant Time Complexity

```
// 1 instruction execution take 1 unit of time

// Big O of 1 --- O(1) -- Constant Time
public static void main(String[] args) {
    System.out.println("Hello World"); // 1
}
```

Constant Time Complexity

```
// 1 instruction execution take 1 unit of time

// 1+1+1+1+1 = 5 -- O(5) -- Constant Time Complexity -- O(1)
public static void main(String[] args) {
    System.out.println("Hello World"); // 1
    System.out.println("Hello World"); // 1
    System.out.println("Hello World"); // 1
    System.out.println("Hello World"); // 1
    System.out.println("Hello World"); // 1

}
```

Linear Time Complexity

```
//Time Complexity = O(N)
public static void main(String[] args) {
        int N = 1000000;
        for(int i = 1; i <= N; i++) {
            System.out.println("Hello World"); // Will be executed N times .
        }
    }
```

Linear Time Complexity

```
  // 1+1+1+1+1 = 5 + N === O(5) + O(N) == O(N) - Linear Time Complexity
public static void main(String[] args) {
      System.out.println( "Hello World"); // 1
      System.out.println( "Hello World"); // 1
      System.out.println( "Hello World"); // 1
      System.out.println( "Hello World"); // 1
      System.out.println( "Hello World"); // 1


      int N = 1000000;
      for(int i = 1; i <= N; i++) {
          System.out.println( "Hello World"); // Will be executed N times . Time Complexity = O(N)
      }
  }
```

Linear Time Complexity

```
// O(1) + O(N) + O(N) == O(1) + O(2N) == O(1) + 2*O(N) == O(N)
public static void main(String[] args) {
    System.out.println("Hello World"); // 1
    System.out.println("Hello World"); // 1
    System.out.println("Hello World"); // 1
    System.out.println("Hello World"); // 1
    System.out.println("Hello World"); // 1
    // 1 + 1 + 1 + 1 + 1 = 5 == O(5) -- O(1)

    int N = 1000000000;
    for(int i = 1; i <= N; i++) {
        System.out.println("Hello World"); // Will be executed N times . Time Complexity = O(N)
    }

    for(int i = 1; i <= N; i++) {
        System.out.println("Hello World"); // Will be executed N times . Time Complexity = O(N)
    }
}
```

Quadratic Time Complexity

```
for(int i = 1; i <= N; i++) { // N times
    for(int j = 1; j <= N; j++) { // N times
        System.out.println("Hello World"); //  N * N times == N^2 (N squared) == O(N^2)
    }
}
```

Quadratic Time Complexity

```
// O(1) + O(N) + O(N^2) == O(N) + O(N^2) == O(N^2) -- Quadratic Time COmplexity
public static void main(String[] args) {
    System.out.println("Hello World"); // 1
    System.out.println("Hello World"); // 1
    System.out.println("Hello World"); // 1
    System.out.println("Hello World"); // 1
    System.out.println("Hello World"); // 1

    int N = 1000000000;
    for(int i = 1; i <= N; i++) {
        System.out.println("Hello World"); // Will be executed N times . Time Complexity = O(N)
    }

    for(int i = 1; i <= N; i++) { // N times
        for(int j = 1; j <= N; j++) { // N times
            System.out.println("Hello World"); //  N * N times == N^2 (N squared) == O(N^2)
        }
    }
}
```

Cubic Time Complexity

```
        for(int i = 1; i <= N; i++) { // N times
            for(int j = 1; j <= N; j++) { // N times
                for(int k = 1; k <= N; k++) { // N times
                    System.out.println("Hello World"); //  N * N * N == O(N^3)  Cubic Time
COmplexity
                }
            }
        }
```

Cubic Time Complexity

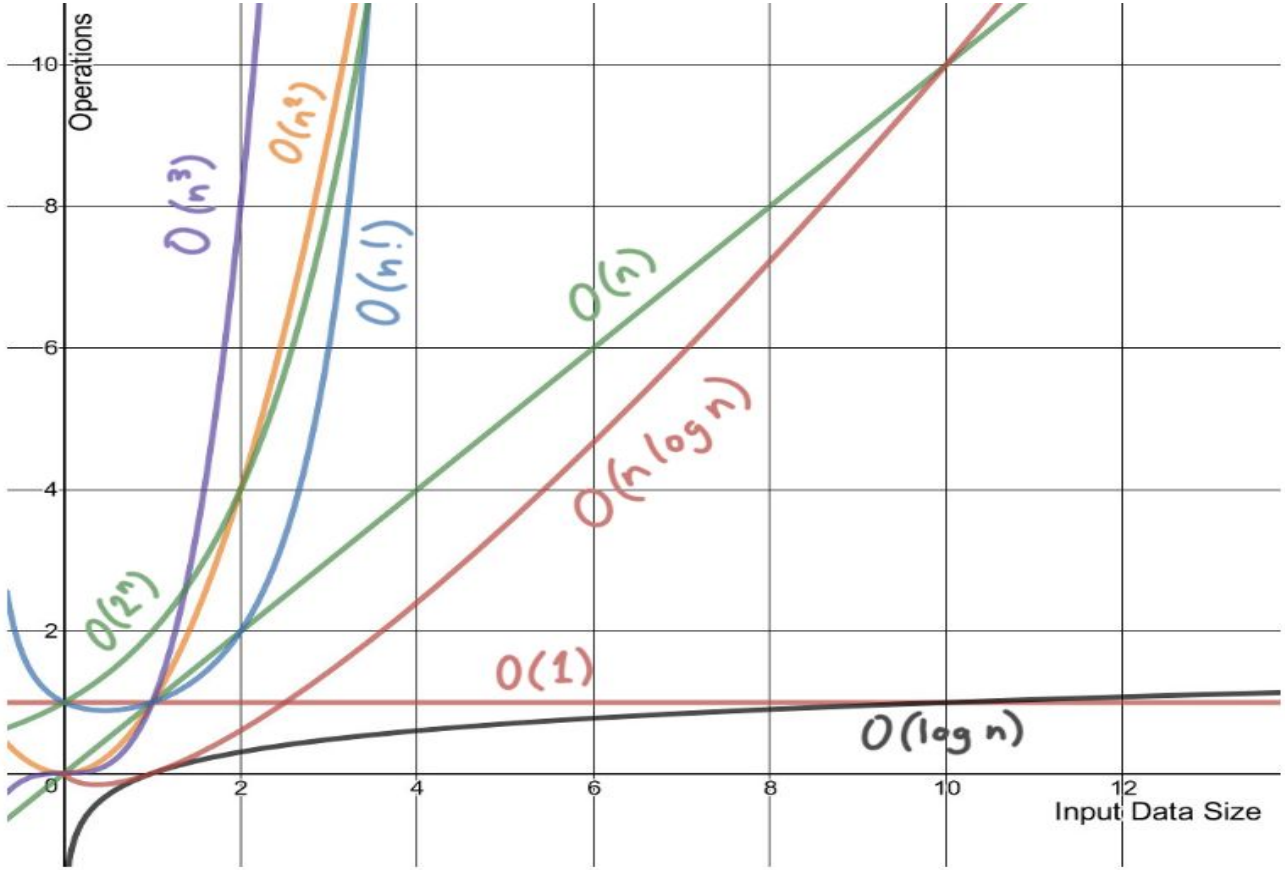
```
public static void main(String[] args) {
    System.out.println( "Hello World" ); // 1
    System.out.println( "Hello World" ); // 1
    System.out.println( "Hello World" ); // 1
    System.out.println( "Hello World" ); // 1
    System.out.println( "Hello World" ); // 1

    int N = 1000000000 ;
    for(int i = 1; i <= N; i++) {
        System.out.println( "Hello World" ); // Will be executed N times . Time Complexity = O(N)
    }

    for(int i = 1; i <= N; i++) {  // N times
        for(int j = 1; j <= N; j++) {  // N times
            System.out.println( "Hello World" ); //  N * N times == N^2 (N squared) == O(N^2)
        }
    }

    for(int i = 1; i <= N; i++) {  // N times
        for(int j = 1; j <= N; j++) {  // N times
            for(int k = 1; k <= N; k++) {  // N times
                System.out.println( "Hello World" ); //  N * N * N == O(N^3)  Cubic Time COmplexity
            }
        }
    }
}
```

Time Complexity Graphs



O(MN) Time Complexity

```
int P = 1000000000;
int Q = 1000;
int R = 1000000;

for(int i = 1; i <= P; i++) { // P times
    for(int j = 1; j <= Q; j++) { // Q times == P * Q
        for(int k = 1; k <= R; k++) { // R times = P * Q * R
            System.out.println("Hello World"); //  P * Q * R == O(PQR) == O(MNP)
        }
    }
}
```

O(M * N) Time Complexity

```
public static void calculateTimeComplexity(int[] array1, int[] array2) {
    for(int i = 0; i < array1.length; i++) { // array1.length times
        for(int j = 0; j < array2.length; j++) { // array1.length * array2.length
            System.out.println("Hello World"); // array1.length * array2.length == O(M *
N)
            // where M == array1.length
            // N == array2.length
        }
    }
}
```

Linear Time Complexity

```
public static void calculateTimeComplexity(int[] intArray) {
    for(int i = 0; i < intArray.length; i++) { // intArray.length times
        System.out.println("Hello World"); // O(N) where N = size of intArray
    }
}
```

Quadratic Time Complexity

```
   public static void calculateTimeComplexity(int[] intArray) {
       for(int i = 0; i < intArray.length; i++) { // intArray.length times
           for(int j = 0; j < intArray.length; j++) { // intArray.length * //
intArray.length times
               System.out.println("Hello World"); // O(N * N) = O(N^2) where N = size of
intArray
           }
       }
   }
```

Space Complexity

How many units of storage/memory are we using for a given input in our program

Constant Complexity

```
// 1 + 1 = O(2) == O(1) == Constant Space Complexity
public static void calculateSpaceComplexity(int[] intArray) {
    for(int i = 0; i < intArray.length; i++) { // 1 unit for variable 1
        for(int j = 0; j < intArray.length; j++) { // 1 unit for variable j
            System.out.println("Hello World");
        }
    }
}
```

Linear Space Complexity

```
    // 1 + 1 + intArray.length == 1 + 1 + N == O(2) + O(N) == O(N) == Linear Space
Complexity
    // where N is the size of intArray
    public static void calculateSpaceComplexity(int[] intArray) {
        int[] intArray2 = Arrays.copyOf(intArray); // Size of intArray == intArray.length

        for(int i = 0; i < intArray2.length; i++) { // 1 unit for variable 1
            for(int j = 0; j < intArray2.length; j++) { // 1 unit for variable j
                System.out.println("Hello World");
            }
        }
    }
```

Linear Space Complexity

```
   // 1 + 1 + intArray.length + intArray.length == O(2) + O(2N) == O(N) == Linear Space
Complexity
   // where N is the size of intArray
   public static void calculateSpaceComplexity(int[] intArray) {
       int[] intArray2 = Arrays.copyOf(intArray, intArray.length);  // Size of intArray ==
intArray.length
       int[] intArray3 = Arrays.copyOf(intArray, intArray.length);  // Size of intArray ==
intArray.length
       for(int i = 0; i < intArray2.length; i++) {  // 1 unit for variable 1
           for(int j = 0; j < intArray2.length; j++) {  // 1 unit for variable j
               System.out.println("Hello World");
           }
       }
   }
```

Logarithms

2 * 2 * 2 * 2 = 16

I can write it as 2^4 = 16

Here 4 is called the logarithm of 16 when base is 2

log(16) base2 = 4

—----------

3 * 3 * 3 = 27 —---------- 3 ^3 = 27

log(27)base3 = 3

Suppose this cake has 16 slices… and every time someone asks for cake I give them half the slices

- Person 1 = 8 slices … so I have 16 - 8 = 8 slices remaining
- Person 2 = 4 slices … so I have 8 - 4 = 4 slices remaining
- Person 3 = 2 slices … so I have 4 - 2= 2 slices remaining
- Person 4 = 1 slice … so I have 2 - 1 = 1 slice remaining
- Person 5 = 1 slice … so I have 1 - 1 = 0 slice remaining

16 = 8 * 2 == (4 * 2) * 2 == 2 * 2 * 2 * 2 = 1 * 2 * 2 * 2 * 2 = 2^4

2^4 = 16

log16 base2

Suppose I have 1 orange and for every friend I make I buy double the oranges that I have

- 1 orange
- Friend 1 == 1 * 2 = 2 oranges
- Friend 2 == 2 * 2 = 4 oranges
- Friend 3 == 4 * 2 = 8 oranges
- Friend 4 == 8 * 2 = 16 oranges

By the time I had 16 oranges how many friends did I make?

2 ^ 4 = 16

log(16)base2 = 4

Logarithm time complexity

```
public static void calculateTimeComplexity(int[] intArray) {
        int sizeOfArrayToProcess = intArray.length;// 64 - 32 - 16 - 8 - 4 - 2 - 1

        while(sizeOfArrayToProcess >1) {
            System.out.println("Hello World"); // 1 + 1 + 1 + 1 + 1 + 1

            sizeOfArrayToProcess = sizeOfArrayToProcess/2; // 1 + 1 + 1 + 1 + 1 + 1
        }

       // 2 * 2 * 2 * 2 * 2 = 32  ==== 2^5 = 32
       // 2 * 2 * 2 * 2 * 2 * 2 = 64 ==== 2^6 = 64
       // 2 * 2 * 2 * 2 * 2 * 2 * 2 = 128 === 2^7 = 128
       // log(intArray.length) base 2
       // logN where N is the size of intArray and base is 2
}
```

```java
public static void main(String[] args) {
    int[] intArray = {1,2,3,4,5,6,7,8,9};
    int value = 17;
    System.out.println(findNumber(intArray, value));
}


// Given an array and a value, identify if the value is present in the array or not.
// Examples
// intArray = {1,2,3,4,5,6,7}
// value = 4 then you return true
// if value = 15 then you return false
public static boolean findNumber(int[] intArray, int value) {
    for(int i = 0; i < intArray.length; i++) {
        if(intArray[i] == value) { // intArray.length
            return true;
        }
    }
    return false;
}
// Time Complexity = O(N) where N is the size of the array
// Space Complexity = O(1)
```

```java
// Given an array and a value, create a new array which contains the square of the original array elements
// and then check if the value exists in the new array
// Examples
// intArray = {1,2,3,4,5,6,7}
// intArray2 = {1, 4, 9, 16, 25, 36, 49};
// value = 25 then you return true
// if value = 15 then you return false
public static boolean findNumber(int[] intArray, int value) {
    // int[] arrayOfSquares = {?,?,?,?}; we don't know the values
    // N space where N is the size of input array
    int[] arrayOfSquares = new int[intArray.length]; // create a new array with same size as input array

    // We populate/initialize arrayOfSquares
    for(int i = 0; i < intArray.length; i++) {  // 1 size
        // This statement executes N time where N is the size of input array
        arrayOfSquares[i] = intArray[i] * intArray[i];
    }

    for(int i = 0; i < arrayOfSquares.length; i++) {  // 1 size
        // This statement executes N time where N is the size of input array
        if(arrayOfSquares[i] == value) {
            return true;
        }
    }

    return else; // 1
}
// Time Complexity = 1 + N + N = O(1) + O(N) + O(N) = O(N);
// Space Complexity = 1 + 1 + N = O(N)
```