PDP ONLINE

DSA – Data Structures

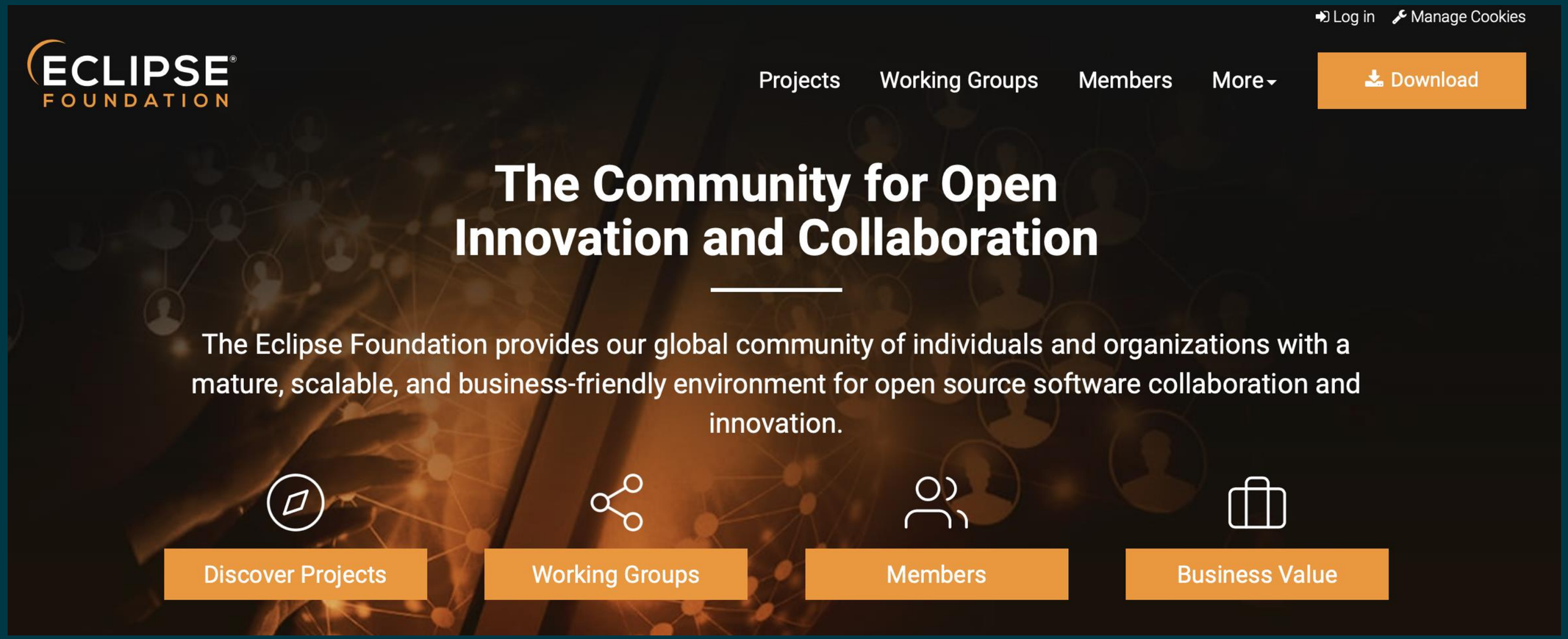
Asymptotic Analysis

Course Planning

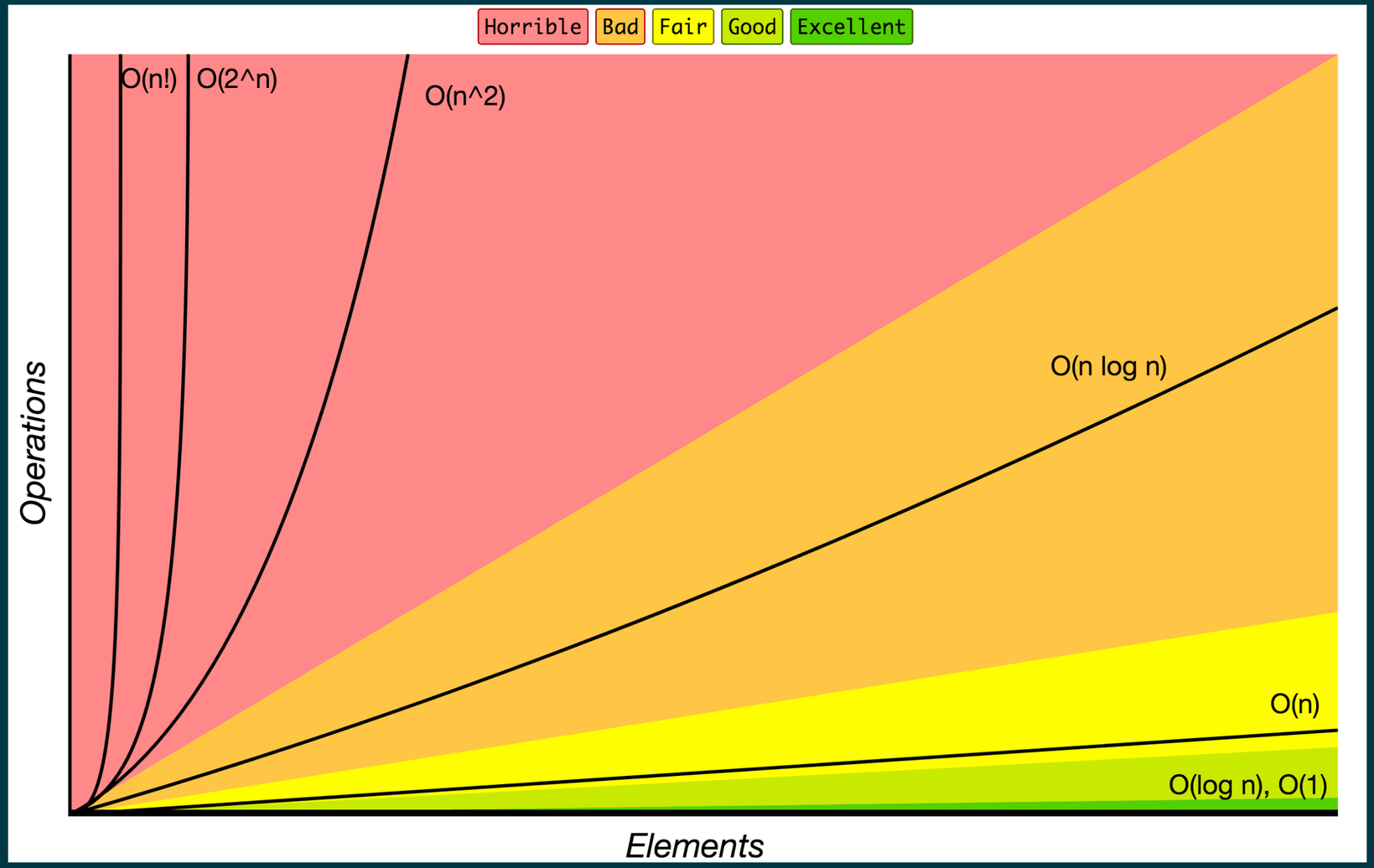
Algorithms	Data Structures	Algorithmic Approaches	Interview Practices
1.Introduction	1.Asymptotic Analysis	1.Search Algorithms	1.In-place Reversal
2.Number 1	2.Dynamic Array	2.Sort Algorithms	2.Two Heaps
3.Number 2	3.LinkedList	3.Dac Algorithms	3.Subsets
4.String 1	4.Stack	4.Recursion	4.Modified BS
5.String 2	5.Queue	5.Sliding Window	5.Bitwise XOR
6.Array 1	6.Tree	6.Two Pointers	6.Top 'K' Elements
7.Array 2	7.Heap	7.Fast & Slow	7.K-way Merge
8.Matrix	8.Trie	8.Cyclic Sort	8.Knapsack Problem
9.DP 1	9.Graph	9.Breadth First Search	9.Topological Sort
10.DP 2	10.Undirected Graph	10.Depth First Search	10.Mock Interview

https://kurbanovxurshid.medium.com/data-structures-and-algorithms-course-plan-1ab912ec1e17

Eclipse IDE


Log in
Manage Cookies
ECLIPSE FOUNDATION
Projects
Working Groups
Members
More
Download
The Community for Open Innovation and Collaboration
The Eclipse Foundation provides our global community of individuals and organizations with a mature, scalable, and business-friendly environment for open source software collaboration and innovation.
Discover Projects
Working Groups
Members
Business Value

Big O Notation



Time Complexity

```java
public static void main(String[] args) {


    int n=100;

    // n-2
    for(int i=2; i<n; i++) {
        System.out.println("Hey - I'm busy looking at: " + i);
    }

    // √n-2
    for(int i=2; i<Math.sqrt(n); i++) {
        System.out.println("Hey - I'm busy looking at: " + i);
    }

}
```

O(1)

```java
public class BigO {

    public static void main(String[] args) {

        int[] numbers = new int[100];

        // O(1)
        System.out.println(numbers[0]);

        // O(2) => O(1)
        System.out.println(numbers[0]);
        System.out.println(numbers[0]);
    }

}
```

O(n)

```java
public static void main(String[] args) {

    int[] numbers = new int[100];

    // O(n)
    for(int i=0; i<numbers.length; i++)
        System.out.println(numbers[i]);


    // O(1+n+1)=O(n+2)=O(n)
    System.out.println(); // O(1)
    for(int i=0; i<numbers.length; i++)
        System.out.println(numbers[i]);
    System.out.println(); // O(1)


    // O(n+n)=O(2n)=O(n)
    for(int i=0; i<numbers.length; i++)
        System.out.println(numbers[i]);

    for(int i=0; i<numbers.length; i++)
        System.out.println(numbers[i]);

}
```

```java
public static void main(String[] args) {

    int[] numbers = new int[100];
    String[] names = new String[60];

    // O(n+m)=O(n)
    for(int i=0; i<numbers.length; i++) //O(n)
        System.out.println(numbers[i]);

    for(int i=0; i<numbers.length; i++) //O(m)
        System.out.println(numbers[i]);

}
```

O(n^2)

```java
public static void main(String[] args) {

    int[] numbers = new int[100];

    // O(n*n)=O(n^2)
    for(int i=0; i<numbers.length; i++) { //O(n)
        for(int j=0; j<numbers.length; j++) { //O(n)
            System.out.println(numbers[i] + numbers[j]);
        }
    }

}
```

```java
public static void main(String[] args) {

    int[] numbers = new int[100];

    // O(n^2+n)=O(n^2)
    for(int i=0; i<numbers.length; i++) { //O(n)
        for(int j=0; j<numbers.length; j++) { //O(n)
            System.out.println(numbers[i] + numbers[j]);
        }
    }

    for(int i=0; i<numbers.length; i++) { //O(n)
        System.out.println(numbers[i]);
    }
}
```

```java
public static void main(String[] args) {

    int[] numbers = new int[100];

    // O(n*n*n)=O(n^3)
    for(int i=0; i<numbers.length; i++) { //O(n)
        for(int j=0; j<numbers.length; j++) { //O(n)
            for(int k=0; k<numbers.length; k++) { //O(n)
                System.out.println(numbers[i] + numbers[j]+ numbers[k]);
            }
        }
    }
}
```

O(log n)

```java
public static void main(String[] args) {

    int[] numbers = { 2, 3, 4, 10, 40 };
    int number = 10;

    // Linear Search - O(n)
    for(int i=0; i<numbers.length; i++) {
        if(number == numbers[i]) {
            System.out.println(i);
        }
    }
}
```

```java
public static void main(String[] args) {

    int[] numbers = { 2, 3, 4, 10, 40 };
    int number = 10;

    // Binary Search - O(log n)
    int left = 0, right = numbers.length - 1;

    while(left < right) {
        int middle = left + (right-left)/2;

        if(numbers[middle] == number)
            System.out.println(middle);

        if(numbers[middle] < number)
            left = middle + 1; // If number greater, ignore left half
        else
            right = middle - 1; // If number is smaller, ignore right half
    }
}
```

Intuition of BigO

1. Overview

In this tutorial, we'll talk about what **Big O Notation means. We'll go through a few examples to investigate its effect on the running time of your code.**

2. The Intuition of Big O Notation

We often hear the **performance of an algorithm described using Big O Notation.**

The study of the performance of algorithms – or algorithmic complexity – falls into the field of algorithm analysis. Algorithm analysis answers the question of how many resources, such as disk space or time, an algorithm consumes.

We'll be looking at time as a resource. Typically, the less time an algorithm takes to complete, the better.

https://www.baeldung.com/java-algorithm-complexity

Space Complexity

```java
public class BigO {

    public static void main(String[] args) {


        int n = 100; // O(1)
        int[] numbers = new int[n]; // O(n)

        //O(n)
        for(int i=0; i<n; i++) {
            System.out.println(numbers[i]);
        }

    }
}
```

Calculate Complexity

```java
public static void main(String[] args) {

    // 1 1 2 3 5 8 13 ...
    System.out.println(fibonacci(3));
}

// Time Complexity O(n)
// Space Complexity O(1)
public static int fibonacci(int n) {
    int x=1, y=1,z = 1;
    if(n == 1 || n==2) return z;

    for(int i=2; i<n; i++) {
        z = x+y;
        x = y;
        y = z;
    }
    return z;
}
```

```java
public static void main(String[] args) {

    // 1 1 2 3 5 8 13 ...
    System.out.println(fibonacci(3));
}

// Time Complexity O(n)
// Space Complexity O(n)
public static int fibonacci(int n) {
    int[] arr = new int[n];
    arr[0] = 1;
    arr[1] = 1;

    for(int i=2; i<n; i++) {
        arr[i] = arr[i-1] + arr[i-2];
    }
    return arr[n-1];
}
```

```java
public static void main(String[] args) {

    // 1 1 2 3 5 8 13 ...
    System.out.print(fibonacci(3));
}

// Time Complexity O(2^n)
// Space Complexity
public static int fibonacci(int n) {
    if(n<=1) return 1;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

Task 1

Quyidagi dasturning Time va Space Complexity larini aniqlang.

```
public static void main(String[] args) {

    int a=0, b=0;
    int n = 100;

    for(int i=0; i<n; i++) {
        a = a + i;
    }
    for(int j=0; j<n; j++) {
        b = b + j;
    }
}
```

Task 2

Quyidagi dasturning Time va Space Complexity larini aniqlang.

```
public static void main(String[] args) {

    int a=0, b=0;
    int n = 100;

    for(int i=0; i<n; i++) {
        for(int j=0; j<n; j++) {
            a = a + j;
        }
    }
    for(int k=0; k<n; k++) {
        b = b + k;
    }
}
```

Task 3

Quyidagi dasturning Time va Space Complexity larini aniqlang.

```
public static void main(String[] args) {

    int a = 0;
    int n = 100;

    for(int i=0; i<n; i++) {
        for(int j=n; j>i; j--) {
            a = a + i + j;
        }
    }
}
```

Task 4

Quyidagi dasturning Time va Space Complexity larini aniqlang.

```
public static void main(String[] args) {

    int a = 0;
    int n = 100;

    for(int i=0; i<n; i = i*2) {
        a = a + i;
    }
}
```

Task 5
Quyidagi dasturning Time va Space Complexity larini aniqlang.

```java
public static void main(String[] args) {

    int a = 0;
    int n = 100;

    for(int i=0; i<n; i++) {
        for(int j=0; j<n; j = j*2) {
            a = a + j;
        }
    }
}
```

Task 6
Sonning tub yoki tub emasligini topadigan dastur yozing va bu dasturning Time va Space Complexity larini aniqlang.