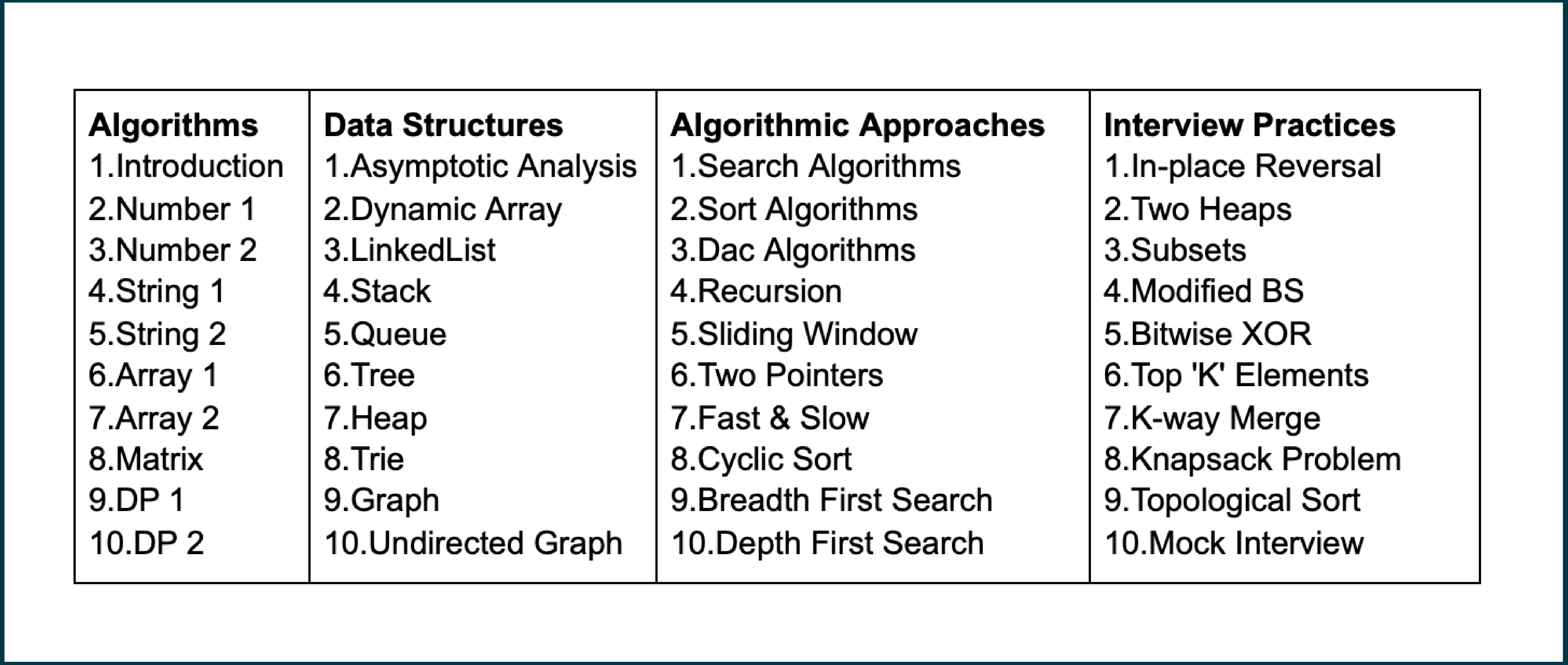
DSA – Data Structures

LinkedList

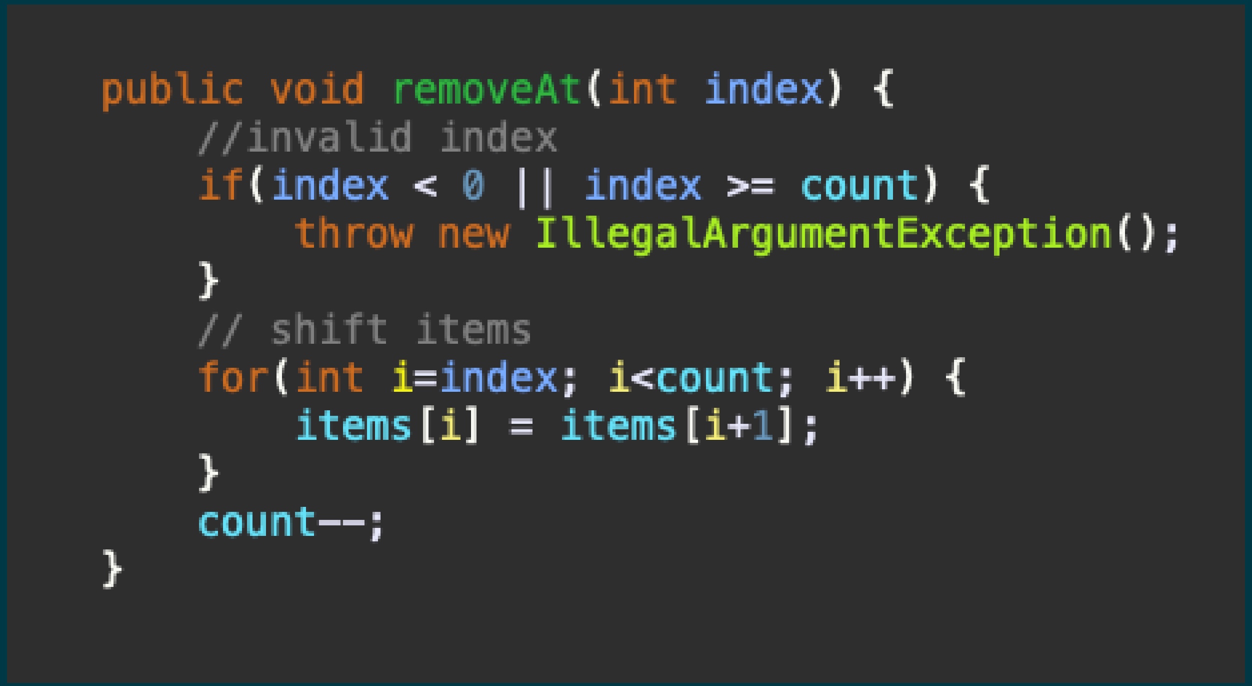
Course Planning



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

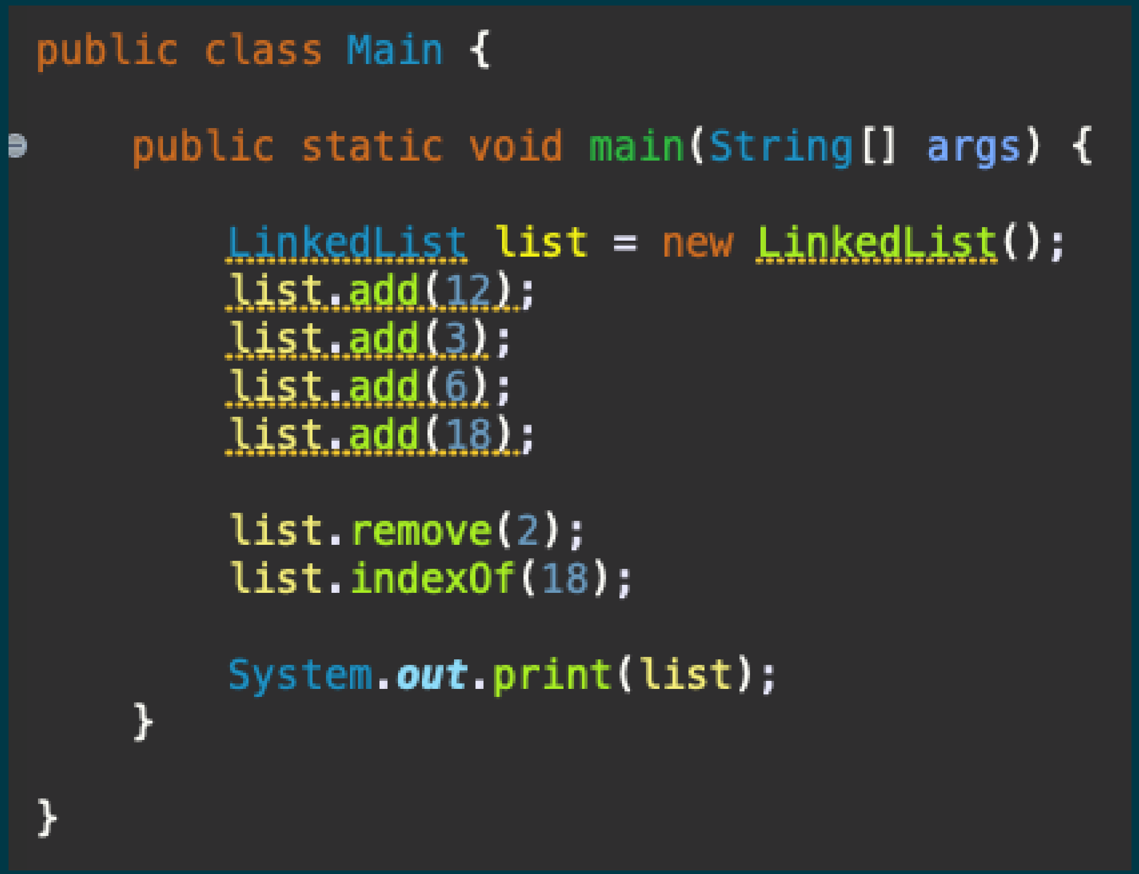
https://kurbanovxurshid.medium.com/data-structures-and-algorithms-course-plan-1ab912ec1e17

Deficiency of ArrayList

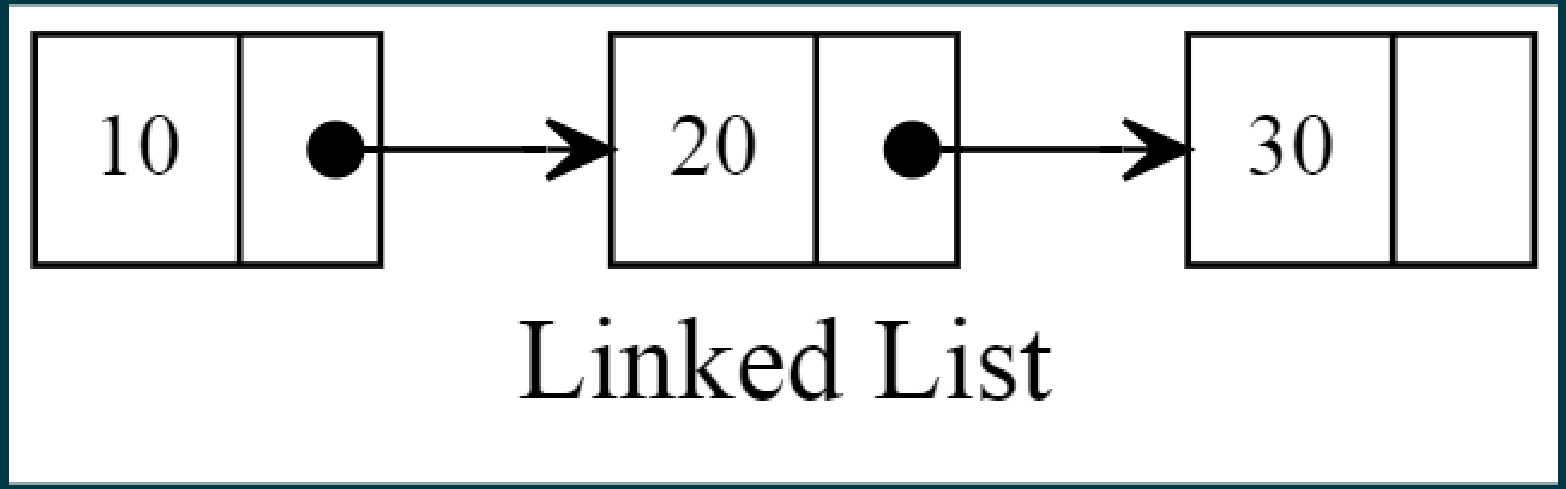


```java
public void removeAt(int index) {
    //invalid index
    if(index < 0 || index >= count) {
        throw new IllegalArgumentException();
    }
    // shift items
    for(int i=index; i<count; i++) {
        items[i] = items[i+1];
    }
    count--;
}
```

java.util.LinkedList

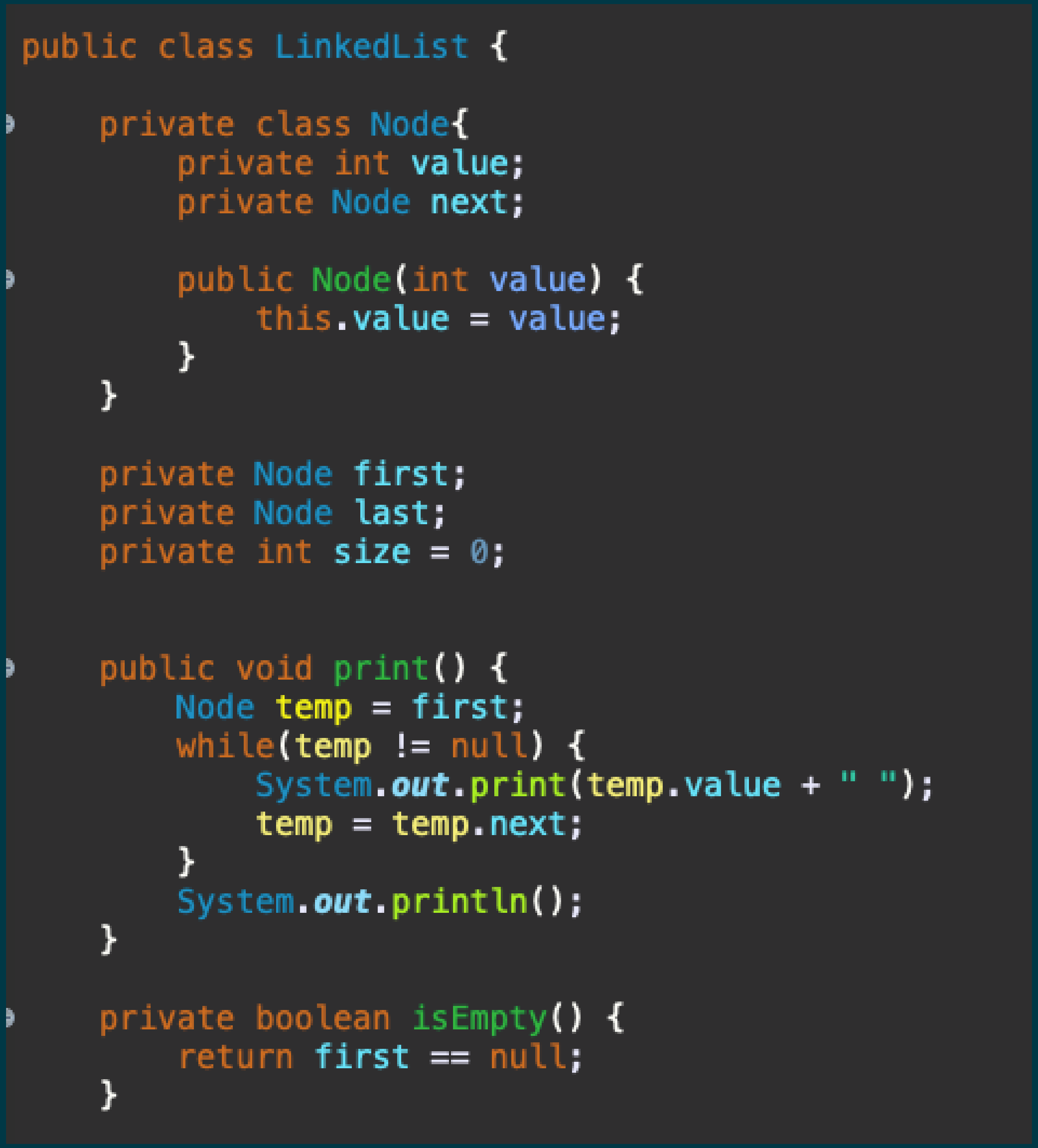


```
public class Main {

    public static void main(String[] args) {

        LinkedList list = new LinkedList();
        list.add(12);
        list.add(3);
        list.add(6);
        list.add(18);

        list.remove(2);
        list.indexOf(18);

        System.out.print(list);
    }

}
```



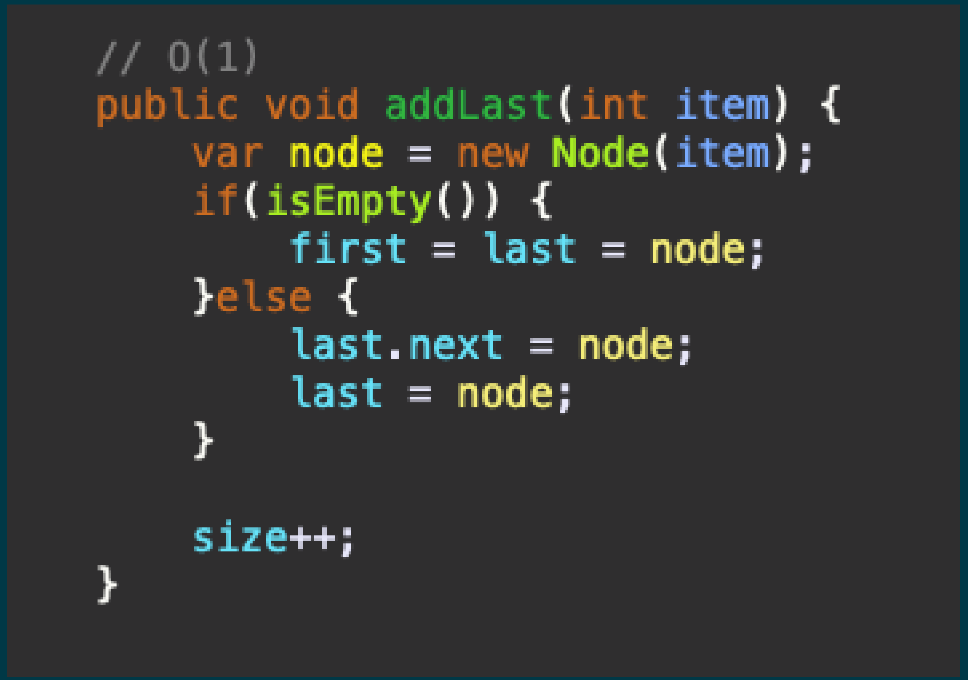
Linked List

Create LinkedList

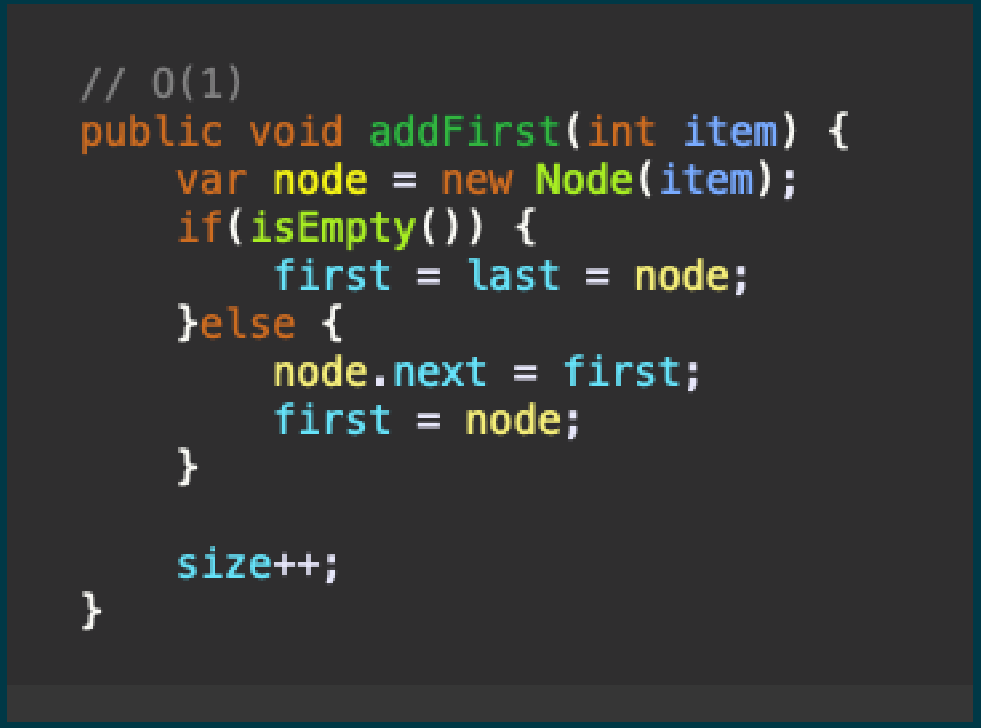


```java
public class LinkedList {

    private class Node{
        private int value;
        private Node next;

        public Node(int value) {
            this.value = value;
        }
    }

    private Node first;
    private Node last;
    private int size = 0;


    public void print() {
        Node temp = first;
        while(temp != null) {
            System.out.print(temp.value + " ");
            temp = temp.next;
        }
        System.out.println();
    }

    private boolean isEmpty() {
        return first == null;
    }
```

addLast

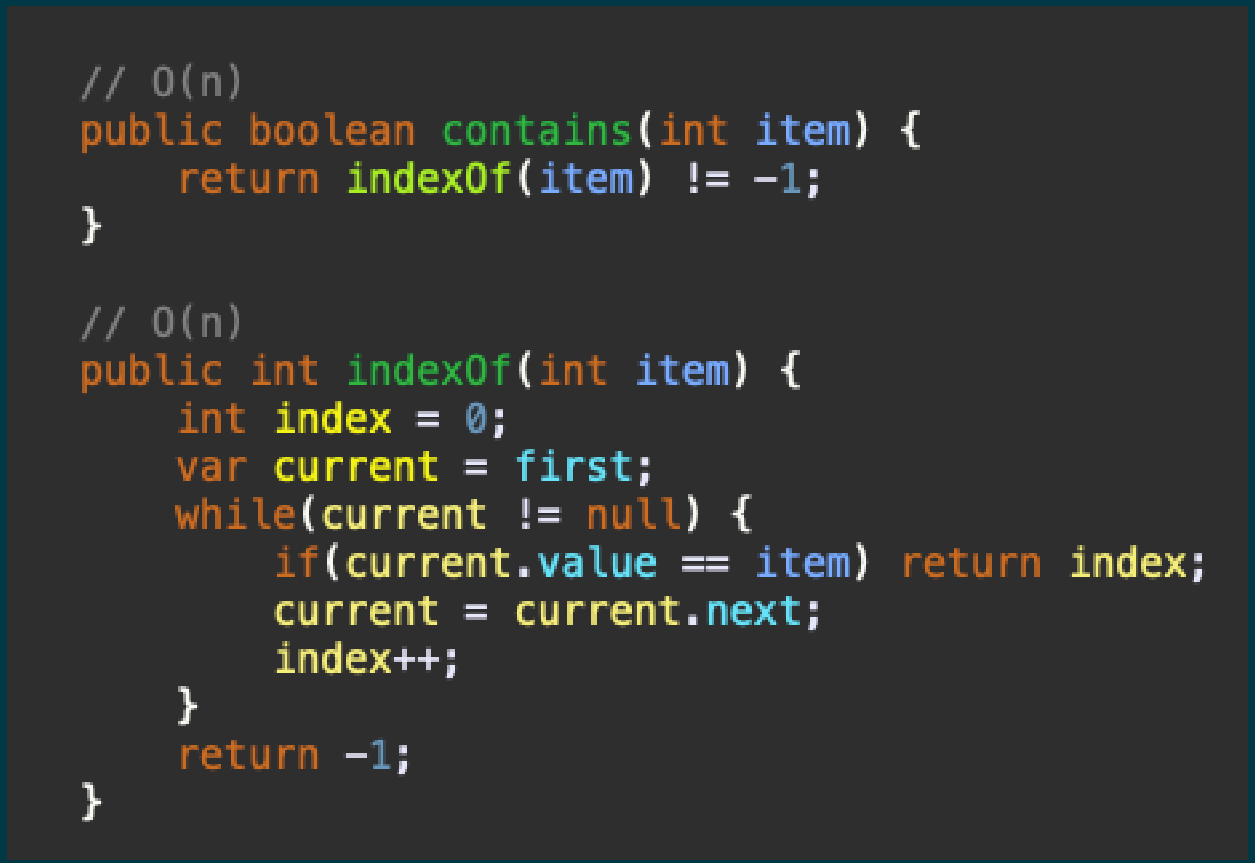

```
// O(1)
public void addLast(int item) {
    var node = new Node(item);
    if(isEmpty()) {
        first = last = node;
    }else {
        last.next = node;
        last = node;
    }

    size++;
}
```

addFirst



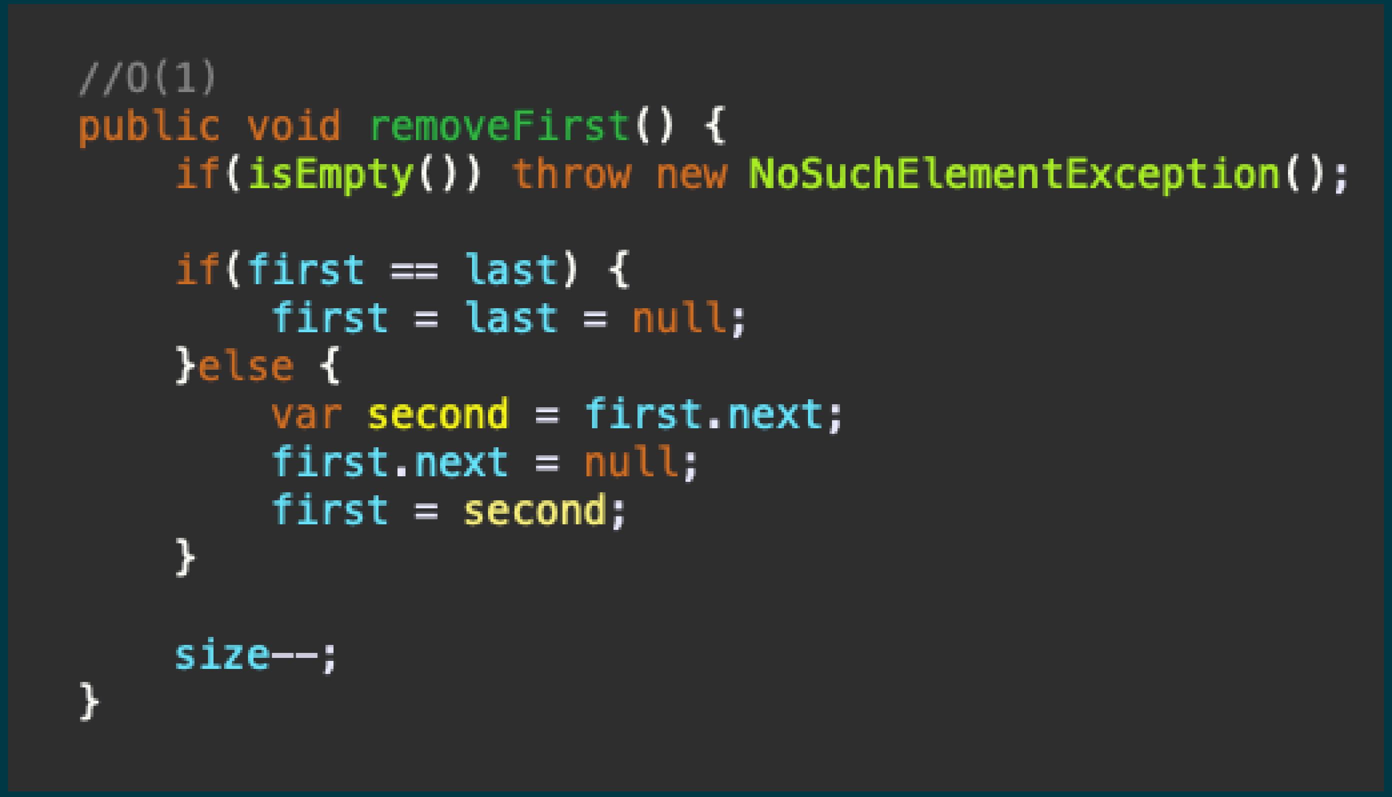
```
// O(1)
public void addFirst(int item) {
    var node = new Node(item);
    if(isEmpty()) {
        first = last = node;
    }else {
        node.next = first;
        first = node;
    }

    size++;
}
```

contains, indexOf



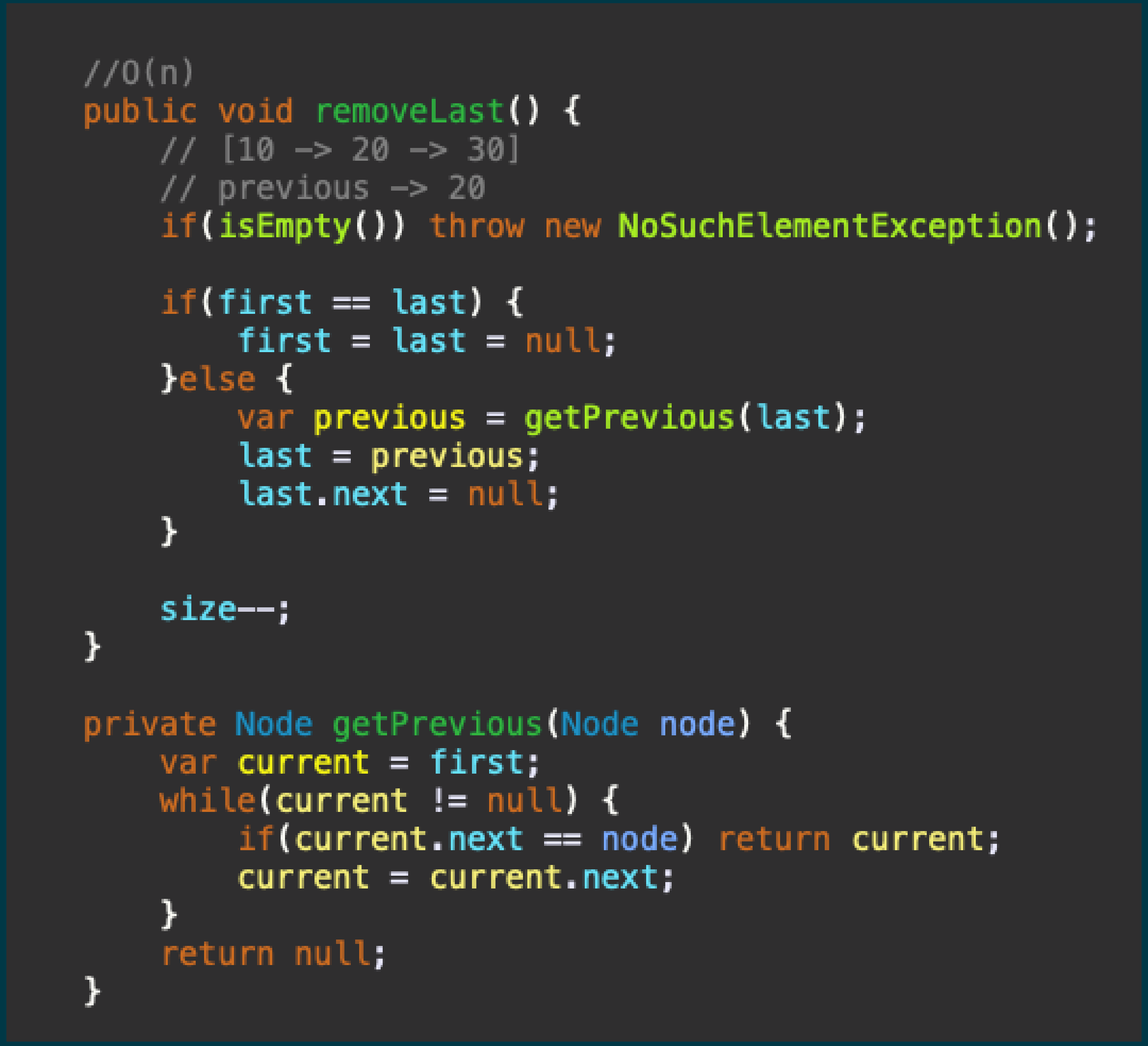
```
// O(n)
public boolean contains(int item) {
    return indexOf(item) != -1;
}

// O(n)
public int indexOf(int item) {
    int index = 0;
    var current = first;
    while(current != null) {
        if(current.value == item) return index;
        current = current.next;
        index++;
    }
    return -1;
}
```

removeFirst



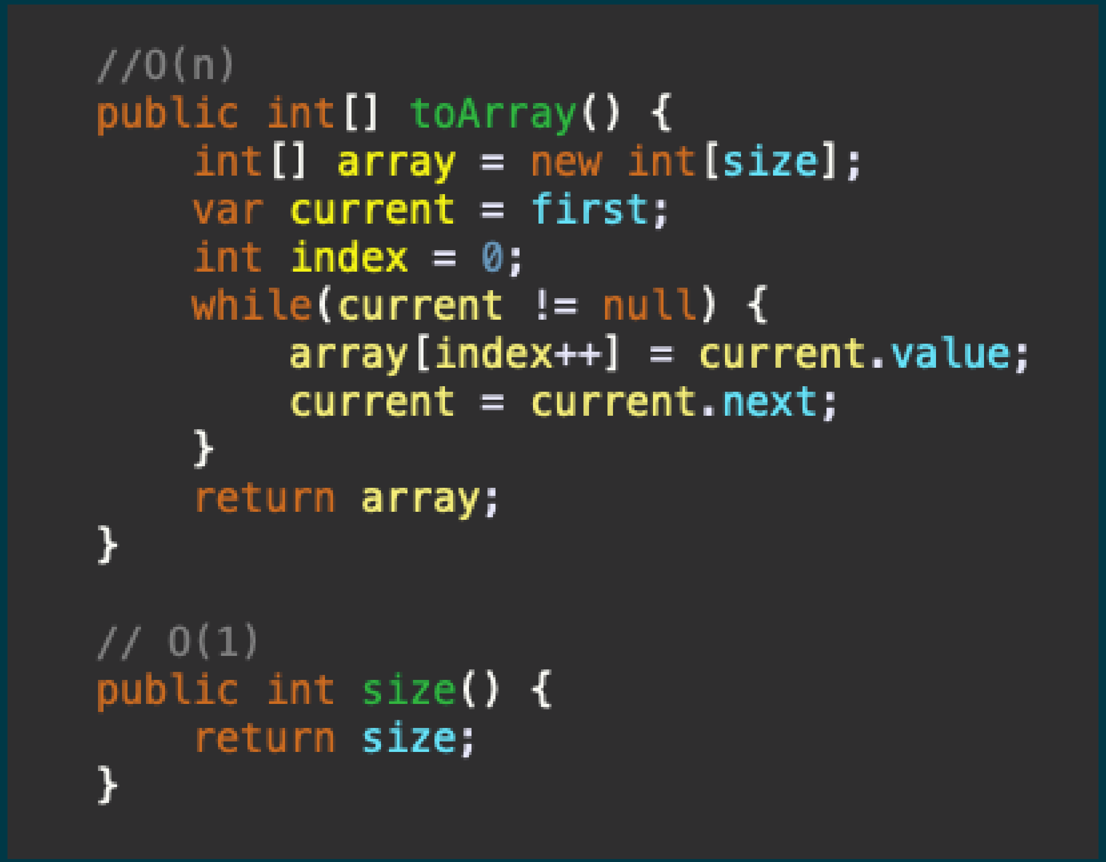
```java
//O(1)
public void removeFirst() {
    if(isEmpty()) throw new NoSuchElementException();

    if(first == last) {
        first = last = null;
    }else {
        var second = first.next;
        first.next = null;
        first = second;
    }

    size--;
}
```

removeLast



```java
//O(n)
public void removeLast() {
    // [10 -> 20 -> 30]
    // previous -> 20
    if(isEmpty()) throw new NoSuchElementException();

    if(first == last) {
        first = last = null;
    }else {
        var previous = getPrevious(last);
        last = previous;
        last.next = null;
    }

    size--;
}

private Node getPrevious(Node node) {
    var current = first;
    while(current != null) {
        if(current.next == node) return current;
        current = current.next;
    }
    return null;
}
```

size, toArray



```java
//O(n)
public int[] toArray() {
    int[] array = new int[size];
    var current = first;
    int index = 0;
    while(current != null) {
        array[index++] = current.value;
        current = current.next;
    }
    return array;
}

// O(1)
public int size() {
    return size;
}
```

Task 1
Darsda o`tilgan LinkedList dan berilgan elementni qidiradigan Search funksiya yarating hamda uning Time Complexity sini aniqlang.

public boolean search(int item)

Task 2
Darsda o`tilgan LinkedList dan qidirilayotgan elementning ohirgi index ni topadigan funksiya yarating hamda uning Time Complexity sini aniqlang.

public void lastIndexOf(int item)

Task 3
Darsda o`tilgan LinkedList ning o`rtadagi qiymatini topadigan funksiya yarating hamda uning Time Complexity sini aniqlang.

public void printMiddle()

Task 4
Darsda o`tilgan LinkedList ning ohiridan K index da turgan elementni topadigan funksiya yarating hamda uning Time Complexity sini aniqlang.

public int getKthFromEnd(int k)

Task 5
Darsda o`tilgan LinkedList ning elementlarini teskarisiga o`zgartiradigan funksiya yarating hamda uning Time Complexity sini aniqlang.

public void reverse()