Section 03: Recurrences and Trees

1. Modeling Code with Recurrences

Given the following code snippet, determine the recurrence relation that represents its behavior:

```java
public static int f(int N) {
    if (N <= 1) {
        return 0;
    }
    int result = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < i; j++) {
            result++;
        }
    }
    return 5 * f(N / 2) + 3 * result + 2 * f(N / 2) + f(N / 2) + f (N / 2);
}
```

2. Recurrence Diagrams

Given the following code snippet:

```
static void triple(int N) {
    if (N > 1) {
        for (int i = 1; i < N / 2; i += 1) {
            System.out.println(i);
        }

        triple(N / 3);
        triple(N / 3);
        triple(N / 3);

        for (int i = 1; i < N / 2; i += 1) {
            System.out.println(i);
        }
    }
}
```

1) Determine the recurrence relation that represents its behavior.

2) Draw the first 3 levels and the final level of a **recurrence diagram** representing the recurrence relation you determined. Include the non-recursive work next to each node.

3. LinkedDeque Trace

Given the following deque (with sentinel nodes):

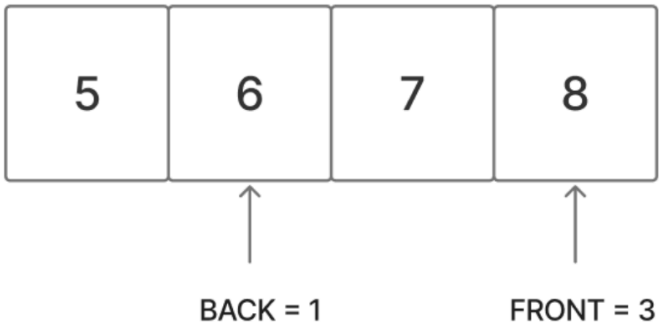
Front Sentinel <-> 9 <-> 8 <-> 7 <-> 6 <-> 5 <-> 4 <-> 3 <-> 2 <-> 1 <-> **Back Sentinel**
Note: -> indicates .next, <- indicates .prev

Now consider the following operation:
`deque.removeFirst();`

Which pointers are updated during this operation? Name the nodes involved and explain how their .next and .prev references change.

4. ArrayDeque Resize Trace

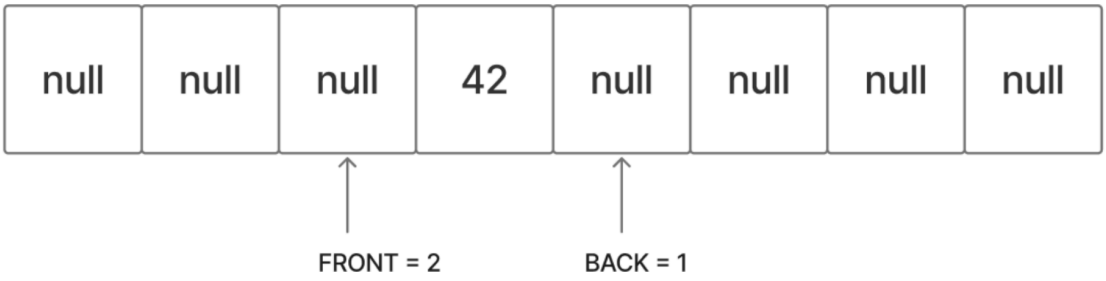
Trace through what happens when an ArrayDeque with elements [5, 6, 7, 8] (front=3, back=1, size=4) needs to **resize to capacity 14.**



What are the size, index values & capacity of the final array? Explain why the circular array approach makes this operation efficient. What should get(2) return after resizing?

5. Edge Case Handling

For an ArrayDeque with a single element (value 42, front=2, back=4, in an array of capacity 8), trace through what happens during a removeFirst() operation.



What special cases need to be considered to maintain correctness?

6. Bug Identification:

Review the following buggy implementation of addFirst() for LinkedDeque.

```
public void addFirst(E element) {

        // Bring in new node, link it up to the front sentinel node

        // and the old first node

        Node<E> newNode = new Node<>(element, front, front.next);

        front.next = newNode;

        back.prev.prev = newNode;

        size += 1;

    }
```

In what situations does this LinkedDeque addFirst() code appear to work correctly? In what situations does this code fail? Why? Identify the bug and explain how to fix the method.