Q1. **Recursive Algorithm**

public class SentenceSegmenter {

public static String findSegment(String sentence, int start, int end) {

if (start >= end) {

return "";

} else {

int spaceIndex = sentence.indexOf(' ', start);

if (spaceIndex != -1 && spaceIndex < end) {

return sentence.substring(start, spaceIndex) + " " + findSegment(sentence, spaceIndex + 1, end);

} else {

return sentence.substring(start, end);

}

}

}

public static void main(String[] args) {

String sentence = "This is an example sentence for the recursive algorithm.";

int startIndex = 5;

int endIndex = 25;

String segment = findSegment(sentence, startIndex, endIndex);

System.out.println(segment);

}

}

Recurrence Relation:

T(n)={1T(n−1)+c​if start≥endotherwise​

**Tracing Tree Method**

findSegment(sentence, 1, 4)

|

|--- findSegment(sentence, 2, 4)

|

|--- findSegment(sentence, 3, 4)

|

|--- findSegment(sentence, 4, 4) -> Base case reached

**Time Complexity: ( O(n) )**

**Q2**. **Algorithm For Circular Shift in a Sorted array**

public class ArrayShifter {

public static int[] circularShift(int[] array, int shift) {

int length = array.length;

int[] shiftedArray = new int[length];

shift = shift % length;

for (int i = 0; i < length; i++) {

int newPosition = (i + shift) % length;

shiftedArray[newPosition] = array[i];

}

return shiftedArray;

}

public static void main(String[] args) {

int[] array = {5, 15, 29, 35, 42};

int shift = 2;

int[] shiftedArray = circularShift(array, shift);

for (int num : shiftedArray) {

System.out.print(num + " ");

}

}

}

Time Complexity: **( O(n) )**