LDS - תת-הסדרה היורדת הארוכה ביותר (לא נלמד)

תיאור הבעיה

נתון מערך של מספרים. יש למצוא את אורך תת הסדרה היורדת הארוכה ביותר מתוך המערך.

דוגמה

עבור הסדרה במערך:

0	1	2	3	4	5
10	2	12	4	8	0

נקבל את הפתרונות הבאים:

$$12, 8, 0 \quad OR \quad 10, 4, 0 \quad OR \quad 12, 4, 0 \quad OR \quad 10, 8, 0 \quad OR \quad 10, 2, 0$$

פתרון הבעיה

- אפשרות א' - אלגוריתם חמדני
  - נקבע שהאיבר הראשון יהיה תחילת הסדרה.
  - נעבור על המערך וניקח בכל שלב את האיבר הבא שקטן מקודמו עד שנגיע לסוף המערך.
  - לדוגמא, עבור הסדרה הבאה: $4, 3, 2, 5$ אנחנו ניקח את 4, אחר כך את 3 , 2 ואז נתקל בסוף ב-5 ושעולה מ-2 ולך לא יכנס לתת הסדרה של הפתרון.

```
public static Stack<Integer> greedy(int[] arr) {
   Stack<Integer> sequence = new Stack<>();
   sequence.add(arr[0]);
   for(int i = 1 ; i < arr.length; i++) {
       if(sequence.peek() > arr[i])
           sequence.add(arr[i]);
   }
   return sequence;
}

public static void main(String[] args) {
   int[] arr = {4,3,2,5};
   System.out.println(greedy(arr)); // [4, 3, 2]
}
```

- **סיבוכיות**: אנחנו עוברים פעם אחת על כל המערך ולכן $O(n)$.
- **נכונות השיטה**: אלגוריתם זה יחזיר את הסדרה היורדת הראשונה שימצא ולא בהכרח הארוכה ביותר בסדרה ולכן זה לא אלגוריתם טוב.

❖ אפשרות ב' - אלגוריתם חמדני משופר

- נרוץ על המערך ועבור על איבר נפעיל את האלגוריתם החמדני שהצגנו באפשרות א'.
- בסוף כל הפעלה נבצע השוואה ונמצא את תת הסדרה היורדת הארוכה ביותר מבין כל האיטרציות שביצענו.
- לדוגמה, נתבונן בסדרה $\{7,\ 6,\ 5,\ 4,\ 3,\ 2,\ 101,\ 100,\ 1\}$.
  נקבל תחילה את הסדרה 7,6,5,4,3,2 כאשר 7 הוא ראש הסדרה, אחר כך נמשיך החל מראש הסדרה 6 ונקבל 6,5,4,3,2 וכו'...

```
public static Stack<Integer> greedy(int[] arr, int start) {
   Stack<Integer> sequence = new Stack<>();
   sequence.add(arr[start]);

   for(int i = start+1; i < arr.length; i++) {
       if(sequence.peek() > arr[i])
           sequence.add(arr[i]);
   }

   return sequence;
}

public static Stack<Integer> improved(int[] arr) {
   Stack<Integer> sequence = new Stack<>();

   for(int i = 0 ; i < arr.length; i++) {
       Stack<Integer> temp_seq = greedy(arr,i);
       if(temp_seq.size() > sequence.size())
           sequence = temp_seq;
   }
   return sequence;
}

public static void main(String[] args) {
   int[] arr = {7, 6, 5, 4, 3, 2, 101, 100, 1};
   System.out.println(improved(arr)); // prints [7, 6, 5, 4, 3, 2, 1]
}
```

- **סיבוכיות**: על כל איבר חוזרים ובודקים את המשך המערך החל ממנו ולכן $O(n^2)$.
- **נכונות השיטה**: האלגוריתם לא החזיר את התשובה הנכונה כי לא פתרנו את הבעיה שלפעמים כדאי לוותר על מספר איברים כדי לקחת אחרים טובים יותר.

❖ אפשרות ג' - אלגוריתם באמצעות LCS

- נשתמש באלגוריתם של LCS (מציאת תת-המחרוזת המשותפת הארוכה ביותר בין 2 מחרוזות).
- נשכתב את האלגוריתם שיתאים ל-2 מערכים של מספרים ונפעיל אותו על המערך הנתון ועל המערך הנתון לאחר מיון כלומר $LCS(arr,\ Sort(arr))$.
- בשונה מ-LIS, הפעם נמיין את המערך **בסדר יורד** ולא בסדר עולה.
- האלגוריתם עובד בשיטת תכנות דינאמי של LCS.

```
public static int[] LCS(int[] X) {
   int[] temp_Y = new int[X.length];

   for(int i = 0; i < X.length; i++)
       temp_Y[i] = X[i];

   Arrays.sort(temp_Y);

   int[] Y = new int[X.length];

   for(int i = X.length-1; i >= 0 ; i--) {
       Y[X.length-1-i] = temp_Y[i];
   }

   int[][] matrix = new int[X.length+1][Y.length+1];
   generateMatrix(matrix,X,Y,1,1);

   int i = matrix.length - 1;
   int j = matrix.length - 1;
   int end = matrix[i][j];
   int start = 0;
   int[] solution = new int[end];

   while(start < end) {
       if(X[i-1] == Y[j-1]) {
           solution[end-start-1] = X[i-1];
           i--;
           j--;
           start++;
       }
       else if(matrix[i-1][j] >= matrix[i][j-1]) {
           i--;
       }
       else {
           j--;
       }
   }
   return solution;
}
```

```
public static void generateMatrix(int[][] matrix, int[] X, int[] Y, int i, int j) {
   if(i == matrix.length)
       return;
   if(j == matrix.length) {
       System.out.println();
       generateMatrix(matrix,X,Y,i+1,1);
   } else {
       if(X[i-1] == Y[j-1]) {
           matrix[i][j] = matrix[i-1][j-1] + 1;
       }else {
           matrix[i][j] = Math.max(matrix[i-1][j], matrix[i][j-1]);
       }
       System.out.print(matrix[i][j] + " ");
       generateMatrix(matrix,X,Y,i,j+1);
   }
}
public static void main(String[] args) {
   int[] arr = {7, 6, 5, 4, 3, 2, 101, 100, 1};
   System.out.println(Arrays.toString(LCS(arr)));
   // prints [7, 6, 5, 4, 3, 2, 1]
}
```

- **סיבוכיות**: סיבוכיות ההעתקה $O(n)$, סיבוכיות המיון $O(nlogn + n)$ וסיבוכיות בניית המטריצה היא $O(n^2)$ ולכן סה"כ נקבל $O(n^2 + nlogn + 2n)$.

- **נכונות השיטה**: מכיוון שהמטרה היא למצוא תת סדרה היורדת הארוכה ביותר אז בהכרח התשובה היא תת סדרה של המערך הממוין כך ששומרים על סדר האיברים במערך ולכן תת הסדרה המשותפת בין 2 המערכים היא בדיוק התשובה.

❖ אפשרות ד' - אלגוריתם באמצעות חיפוש שלם

- נייצר את כל תתי המערכים האפשריים.
- נבדוק עבור על תת מערך אם הוא תת סדרה יורדת (מתחילתו ועד סופו).
- אם כן, נבדוק האם הוא הכי ארוך שמצאנו עד עכשיו ונשמור אותו.
- הפונקציה תחזיר בסוף מערך של תת הסדרה היורדת הגדולה ביותר.

```
public static Vector<int[]> subsets(int[] arr) {
   Vector<int[]> subsets = new Vector<>();
   int length = (int)Math.pow(2,arr.length)-1;

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

       int binary = decimal;
```

```
        int i = 0;
        Vector<Integer> subset = new Vector<>();
        while (binary != 0) {
            if(binary % 2 == 1) {
                subset.add(arr[i]);
            }
            i++;
            binary /= 2;
        }

        int[] ss = new int[subset.size()];

        for(int j = 0 ; j < ss.length; j++) {
            ss[j] = subset.get(j);
        }
        subsets.add(ss);
    }
    return subsets;
}

public static int[] bruteForce(int[] arr) {

    Vector<int[]> vector = subsets(arr);
    int[] bestSubset = new int[0];
    for(int i = 0 ; i < vector.size(); i++) {
        boolean isDecreasing = true;
        int[] subset = vector.get(i);
        for(int j = 1; j < subset.length; j++) {
            if(subset[j-1] < subset[j]){
                isDecreasing = false;
            }
        }
        if(isDecreasing && bestSubset.length < subset.length) {
            bestSubset = subset;
        }
    }

    return bestSubset;
}

public static void main(String[] args) {
    int[] arr = {7, 6, 5, 4, 3, 2, 101, 100, 1};
    System.out.println(Arrays.toString(bruteForce(arr)));
    // print [7, 6, 5, 4, 3, 2, 1]
}
```

↜ **סיבוכיות**: מספר תתי הקבוצות הוא $2^n - 1$ וגם על כל תת-מערך עוברים ובודקים האם הוא ממוין בסדר

יורד כך שלכל היותר תת מערך כזה הוא בגודל n ולכן סה"כ $O(2^n \cdot n)$.

↜ **נכונות השיטה**: בודקים את כל האפשרויות ולכן בהכרח נגיע גם לתשובה הנכונה.

שאלה

האם אנחנו רוצים למצוא את אורך המחרוזת או רק דוגמא למחרוזת המקיימת LDS?

- ❖ אפשרות ה' - מציאת אורך המחרוזת.
  - ⇚ נדרוס איברים תוך כדי שמירה על האורך הנכון.

```
public static int binarySearch(int[] sequence, int left, int right, int value) {

   while(right - left > 1) { // while we can compare min two values

       int middle = (right+left)/2;

       if(value >= sequence[middle]) {
           right = middle;
       }
       else if(value < sequence[middle]) {
           left = middle;
       }
   }
   return right;
}

public static int length(int[] arr) {
   int[] sequence = new int[arr.length];
   sequence[0] = arr[0];

   int length = 0;
   for(int i = 1 ; i < sequence.length; i++) {

       if(arr[i] > sequence[0]) {
           sequence[0] = arr[i];
       }
       else if(arr[i] < sequence[length]) {
           length++;
           sequence[length] = arr[i];
       } else {
           int index = binarySearch(sequence,0,length,arr[i]);
           sequence[index] = arr[i];
       }
   }

   System.out.println(Arrays.toString(sequence));
```

```
    return length+1;
}

public static void main(String[] args) {
    int[] arr = {11,8,7,3,6,20,4,9,5};
    System.out.println(length(arr)); // prints 5
}
```

- **סיבוכיות**: על כל איבר בסדרה נחפש איפה לשים אותו בתת הסדרה שאנו יוצרים.
  נעשה זאת בחיפוש בינארי כדי לחסוך, סה"כ $O(n \cdot logn)$.