LIS - תת-הסדרה העולה הארוכה ביותר

תיאור הבעיה

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

דוגמה

עבור הסדרה במערך:

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

נקבל את הפתרונות הבאים:

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

פתרון הבעיה

- אפשרות א' - אלגוריתם חמדני
  - נקבע שהאיבר הראשון יהיה תחילת הסדרה.
  - נעבור על המערך וניקח בכל שלב את האיבר הבא בגודלו עד שנגיע לסוף המערך.
  - לדוגמא, עבור הסדרה הבאה: $1, 3, 4, 2$ אנחנו ניקח את 1, אחר כך ניקח את 3 שגדול מהמספר הקודם ולבסוף ניקח את 4 כי אחריו אין איברים שגדולים ממנו ולכן נקבל את הפתרון הבא: $1, 3, 4$

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

public static void main(String[] args) {
   int[] arr = {1,3,4,2};
   System.out.println(greedy(arr)); // [1, 3, 4]
}
```

- **סיבוכיות**: אנחנו עוברים פעם אחת על כל המערך ולכן $O(n)$.
- **נכונות השיטה**: אלגוריתם זה יחזיר את הסדרה העולה 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 העולה הארוכה ביותר מבין כל האיטרציות שביצענו.
- לדוגמה, נתבונן בסדרה $1,\ 100,\ 101,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7$.

  נקבל תחילה את הסדרה $1, 100, 101$ כך ש-1 הוא האיבר הראשון.

  אחר כך נמשיך ונקבע את 100 להיות האיבר הראשון ואז נקבל סדרה $100\ ,\ 101$ ככה נמשיך עד שנקבל את ה-LIS שלנו שהוא במקרה זה $1, 2,\ 3,\ 4,\ 5,\ 6,\ 7$.

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

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

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

   return stack;
}

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

- **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 $LCS(arr,\ Sort(arr))$.
- האלגוריתם עובד בשיטת תכנות דינאמי של LCS.

```
public static int[] LCS(int[] X) {// O(n^2)).

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

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

   Arrays.sort(Y);// sort s_arr O(n*log(n)
   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 = {1, 100, 101, 2, 3, 4, 5, 6, 7};
    System.out.println(Arrays.toString(LCS(arr)));
    // prints [1, 2, 3, 4, 5, 6, 7]
}
```

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

- **נכונות השיטה**: מכיוון שהמטרה היא למצוא תת סדרה עולה ארוכה 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.

- **דוגמא**:

בהינתן סדרת מספרים (מערך) : $arr = \{1, 100, 101, 2, 3, 4, 5, 6, 7\}$

נמיין את המערך $s\_arr = \{1, 2, 3, 4, 5, 6, 7, 100, 101\}$

נשלח ליצירת מטריצה ונקבל: (השורה זה המערך המקורי והעמודה זה הממוין):

1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 2 2
1 1 1 1 1 1 1 2 3
1 2 2 2 2 2 2 2 3
1 2 3 3 3 3 3 3 3
1 2 3 4 4 4 4 4 4
1 2 3 4 5 5 5 5 5
1 2 3 4 5 6 6 6 6
1 2 3 4 5 6 7 7 7

לאחר מכן נשלוף את הפתרון הארוך ביותר כלומר ניצור מערך פתרון באורך 7 ובכל פעם שיש לנו התאמה בין המערך הממוין למערך המקורי נצרף למערך הפתרון את ההתאמה ההתאמה הזאת.
הפתרון יתקבל באמת בסדר עולה מכיוון שאנו מכניסים את ההתאמות החל מהמקום nxn במטריצה ונרוץ "באלכסון" עד ההתאמה האחרונה.
לבסוף נקבל את הפתרון הבא: $[1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7]$.

- ❖ אפשרות ד' - אלגוריתם באמצעות חיפוש שלם
  - ⇐ נייצר את כל תתי המערכים האפשריים.
  - ⇐ נבדוק עבור על תת מערך אם הוא תת סדרה עולה (מתחילתו ועד סופו).
  - ⇐ אם כן, נבדוק האם הוא הכי ארוך שמצאנו עד עכשיו ונשמור אותו.
  - ⇐ הפונקציה תחזיר בסוף מערך של תת הסדרה העולה הגדולה ביותר.

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

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

       int binary = decimal;
       int i = 0;
       Vector<Integer> vec_set = new Vector<>();
       while(binary > 0) {
           if(binary % 2 == 1) {
               vec_set.add(arr[i]);
           }
           binary /= 2;
           i++;
       }
       int[] arr_set = new int[vec_set.size()];
       for(int s = 0 ; s < vec_set.size(); s++) {
           arr_set[s] = vec_set.get(s);
       }
       subsets.add(arr_set);
   }

   return subsets;
}

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

   Vector<int[]> subsets = getSubsets(arr);
   int[] solution = new int[0];
```

```
    int max = 0;

    for(int i = 0 ; i < subsets.size(); i++) {
        int[] subset = subsets.get(i);
        boolean isIncreasing = true;
        for(int j = 1 ; j < subset.length; j++) {
            if(subset[j-1] > subset[j]) {
                isIncreasing = false;
                break;
            }
        }
        if(isIncreasing && subset.length > max) {
            solution = subset;
            max = solution.length;
        }
    }

    return solution;
}


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

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

נפצל את השאלה ל-2 אפשרויות:

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

דוגמה

למשל עבור $10, 12, 20, 21, 16, 17, 18$ נבחר את $10, 12, 20, 21$.

מאחר והאיבר הבא אחרי 21 הוא 16 והוא נוגד את ה-LIS במידה ונצרף אותו לסדרה הנבחרת, אז "נדרוס" את האיברים כלומר נדרוס את 20 ונצרף במקומו את 16.

כלומר נקבל את הסדרה הבאה: $10, 12, 16, 21$.
בכל פעם שאנו מוסיפים איבר אנו בודקים שהוא יותר גדול מכל האיברים הנמצאים משמאלו. אנו חייבים לשמור על אורך הסדרה הנבחרת בכל ביצוע של דריסה אלא אם צירפנו איבר שהוא "לטובתנו" כלומר שומר על סדר עולה ותקין ואז נרחיב את גודל הסדרה בהתאם.
לבסוף נקבל את הסדרה הבאה $10, 12, 16, 17, 18$.
⇐ עבור דוגמה זו האורך נכון והסדרה שהתקבלה חוקית, אבל **לא תמיד נקבל סדרה חוקית.**

דוגמה

נציג סדרה: $5, 9, 4, 20, 6, 3, 7, 8, 11$.
כאן הסדרה שתתקבל היא לא חוקית כי קיבלנו $3, 6, 7, 8, 11$ ובסדרה המקורית $6, 3, 7, 8, 11$
אך עדיין, קיימת תת סדרה שהאורך שלה הוא באורך הזה: $4, 6, 7, 8, 11$ או $5, 6, 7, 8, 11$.
כלומר, אנחנו לא תמיד נקבל את תת הסדרה הארוכה ביותר (מבחינת ערכים חוקיים) אבל זה לא משנה, כי הדרישה היא להחזיר את האורך הגדול ביותר וזה כן יהיה בידיים שלנו.

```
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];
        }
    }
    return length+1;
}

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

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

❖ אפשרות ו' - החזרת תת הסדרה בתכנות דינאמי.

- ניצור מטריצה חדשה nxn (גודל הסדרה x גודל הסדרה).
- נאתחל את ראש המטריצה $matrix[0][0] = arr[0]$ ונתחיל למלא את המטריצה בלולאה.
- אם הערך $arr[i]$ לא גדול מהאיבר האחרון שנוסף למטריצה ולא קטן מהאיבר הראשון במטריצה, אז נשלח נחפש את הערך המתאים בתוך המטריצה בעזרת חיפוש בינארי שיחזיר לנו את האינדקס הדרוש להצבת הערך החדש $arr[i]$.
- כל שורה במטריצה תגדיר לנו את הטווח החדש שנוסף והאיברים החוקיים והמתאימים ביותר לפתרון הנחוץ.
- בסוף נעתיק את השורה האחרונה לפי מספר הטווח למערך חדש בגודל הטווח שקיבלנו ונחזיר אותו.

דוגמה: עבור המערך $5, 9, 4, 20, 6, 3$:

	0	1	2	...5
0	3	0	0	0
1	4	6	0	0
2	5	9	20	0
... ... 5	0	0	0	0

ונחזיר את תת הסדרה **5,9,20**.

מימוש **אינדוקטיבי**:

```
public static int binarySearch(int[][] matrix, int left, int right, int value) {
   while(left <= right) {
       if(left == right) {
           return left;
       }
       int middle = (right+left)/2;
       if(value == matrix[middle][middle]) {
           return middle;
       }
       if(value < matrix[middle][middle]) {
           right = middle;
       }
```

```
        else {
            left = middle+1;
        }
    }
    return -1;
}

public static int[] dynamic(int[] arr) {
    int[][] matrix = new int[arr.length][arr.length];
    int length = 1;
    matrix[0][0] = arr[0];

    for(int i = 1; i < arr.length; i++) {
        int index;
        if(arr[i] > matrix[length-1][length-1]) {
            index = length;
        }
        else if(arr[i] < matrix[0][0]) {
            index = 0;
        } else {
            index = binarySearch(matrix,0,length,arr[i]);
        }
        matrix[index][index] = arr[i];

        if(index == length){
            length++;
        }

        for(int j = 0 ; j < index; j++) {
            matrix[index][j] = matrix[index-1][j];
        }// end j for

    }// end i for

    int[] solution = new int[length];
    for(int i = 0; i < length; i++) {
        solution[i] = matrix[length-1][i];
    }

    return solution;
}
public static void main(String[] args) {
    int[] arr = {5,9,4,20,6,3};
    System.out.println(Arrays.toString(dynamic(arr)));
}
```

מימוש **רקורסיבי**:

```
public static int[] dynamic(int[] arr) {
   int[][] matrix = new int[arr.length][arr.length];
   matrix[0][0] = arr[0];

   int length = recursive(matrix,arr,1,1);

   int[] solution = new int[length];
   fillSolution(matrix,solution,0);

   return solution;
}

public static int recursive(int[][] matrix, int[] arr, int length,
int i) {
   if(i == arr.length){
       return length;
   }

   int index;
   if(arr[i] > matrix[length-1][length-1]) {
       index = length;
   }
   else if(arr[i] < matrix[0][0]) {
       index = 0;
   } else {
       index = binarySearch(matrix,0,length,arr[i]);
   }

   matrix[index][index] = arr[i];
   newRow(matrix,index,0);

   if(index == length) {
       return recursive(matrix,arr,length+1,i+1);
   } else {
       return recursive(matrix,arr,length,i+1);
   }
}
```

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

   if(left > right) {
       return -1;
   }
   if(left == right){
       return left;
   }

   int middle = (right+left)/2;

   if(value < matrix[middle][middle]) {
       return binarySearch(matrix,left,middle,value);
   }
   else if(value > matrix[middle][middle]){
       return binarySearch(matrix,middle+1,right,value);
   }
   else { // value == matrix[middle][middle]
       return middle;
   }
}

public static void newRow(int[][] matrix, int index, int i) {
   if(i == index) {
       return;
   }
   matrix[index][i] = matrix[index-1][i];
   newRow(matrix,index,i+1);
}

public static void fillSolution(int[][] matrix, int[] solution, int
i) {
   if(i == solution.length) {
       return;
   }
   solution[i] = matrix[solution.length-1][i];
   fillSolution(matrix,solution,i+1);
}
```

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

סיבוכיות זמן הריצה:

בכל שלב בלולאה i נבצע: חיפוש בינארי $O(log(n))$ ונעתיק במקרה הגרוע את כל השורה שמעליו $O(n)$ מאחר והלולאה באורך n אז הסיבוכיות היא $O(n \cdot (log(n) + n)) = O(n \cdot log(n) + n^2)$.