Longest Increasing Subsequence - LIS בעיית

בעיית תת-סדרה עולה ארוכה ביותר

ניסוח הבעיה: בהינתן סדרה X=x1, x2, …, xn , יש למצוא את תת-סדרה עולה ארוכה ביותר.

דוגמה:
נתבונן בסדרה :
X = 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15
דוגמה לתת-סדרה עולה ארוכה ביותר היא: 0, 2, 6, 9, 11, 15.
דוגמה נוספת לתת-סדרות ארוכות ביותר היא: 0, 4, 6, 9, 13, 15.

פתרון 1: אלגוריתם חמדני:

אלגוריתם חמדני אומר לנו בכל שלב לקחת איבר ראשון שמתאים ללא התחשבות במה שיקרה בשלב הבא:

```
greedy(arr[])
    n = arr.length
    temp[n], k = 0, temp[0] = arr[0]
    for i=1 to n-1
        if (arr[i] > temp[k])  temp[++k] = arr[i]
    end-for
    ans[k+1]
    for i=1 to k
        ans[i] = temp[i]
    end-for
    return ans
end-greedy
```

סיבוכיות האלגוריתם: O(n)

דוגמה 1:
קלט: Z= 0, 8, 4, 12, 2, 10
פלט: 0, 8, 12 – האלגוריתם עובד.

דוגמה 2:
קלט: נחליף בסדרת Z 8 ב 15:
Z'= 0, 15, 4, 12, 2, 10
פלט: 0, 15

בדוגמה זו רואים כי התת-מחרוזת שקיבלנו היא לא ארוכה ביותר.
נכונות האלגוריתם: אלגוריתם חמדני לא נותן פתרון אופטימאלי. השיטה לא מחזירה את התשובה הנכונה תמיד כי לא תמיד האיבר הראשון שנמצא הוא חלק מהסדרה. בנוסף, לפעמים כדאי לוותר על מספר איברים כדי לקחת אחרים טובים יותר.

פתרון 2: חיפוש שלם:

כמו ב-LCS מחפשים את כל תתי-סדרות של סדרה נתונה ובוחרים בתת-מחרוזת עולה ארוכה ביותר. יוצרים כל תתי המחרוזות האפשריים בעזרת מספרים בינאריים.
פסדו-קוד של חיפוש שלם:

```
int[] exhaustiveSearch (int arr[]) //O(2^m*m)
      m = arr.length, maxLen = 0, n = 2^m-1, ans[]
      int c[] = allCombinations(arr)
      maxLen = 0
      for i=0 to n-1 //O(2^m)
            len = c[i].length
            if (isSorted(c[i]) && len > maxLen) //O(m)
                  maxLen = len
                  ans = c[i]
            end-if
      end-for
      return ans
end-exhaustiveSearch

boolean isSorted(arr[])
n = arr.length
for i=1 to n-1
      if (arr[i-1] > arr[i]) return false
end-for
return true
end-isSorted
```

פסדו-קוד של בניית כל הצירופים של n תווים:

```
int[] allCombinations(int arr[]) //O(2^n)
      n = arr.length, count = 2^n - 1
      int list[count]
      int bin[n]
      for i=0 to count-1
            plus1(bin)
            int k=0
            int temp[n]
            for j=0; to n-1
                  if (bin[j] == 1) temp[k++] = arr[j]
            end-for
            list[i] = temp
      end-for
      return list
end-allCombinations
```

פסדו-קוד של בניית המערך שמכיל מספר בינארי סידורי מ 0- עד 1111...1:

```
plus1(arr[])
```

```
      i=arr.length-1
      while( i>=0 && arr[i]==1)
            arr[i--] = 0
      end-while
      if (i>=0) arr[i] = 1
end-plus1
```

פתרון 3: נשתמש באלגוריתם ידוע - LCS:

נמיין את הסדרה המקורית arr ונמצא תת-סדרה משותפת ארוכה ביותר בין הסדרה המקורית לבין הסדרה ממוינת:
LIS(X) = LCS(arr, sort(arr))

סיבוכיות האלגוריתם: O(n^2)+O(n*log n).

דוגמה: ניקח אותה סדרה arr ונמיין אותה:

arr = **0**, 8, **4**, 12, 2, 10, **6**, 14, 1, **9**, 5, **13**, 3, 11, 7, **15**

sort(arr) = **0**, 1, 2, 3, **4**, 5, **6**, 8, **9**, 10, 11, 12, **13**, 14,**15**

הוכחת תקינות האלגוריתם נובעת מעובדה שלכל תת-סדרה עולה יש התאמה בסדרה ממוינת.

```
LIS_with_LCS(arr[]) //O(n²)
      c[] = copyOf(arr)
      sort(c)
      retun LCS(c, arr)
end_LIS_with_LCS
```

פתרון 4: פתרון באמצעות תכנון/תכנות דינאמי:

4.1 חישוב אורך של תת-סדרה עולה ארוכה ביותר
אלגוריתם:
נבנה מערך עזר בגודל המערך.
מגדירים משתנה lis=0 שיכיל את אורך של תת-סידרה עולה ארוכה ביותר.
בשלב האתחול האיבר הראשון במערך העזר הוא האיבר הראשון במערך הנתון.
כעת נעבור על כל האיבר הבא במערך המקור:

- אם הוא קטן מהאיבר הראשון במערך העזר, הוא יחליף את האיבר הראשון במערך העזר, lis לא משתנה,
- אם הוא גדול מהאיבר האחרון במערך העזר, הוא יוסף לסוף מערך העזר ויוקדם lis: ++lis,
- אם הוא בין לבין נשתמש בחיפוש בינארי למציאת אינדקס/מיקום שלו במערך העזר.

בסיום מקדמים את ה-lis ב-1 כי לא נחשב איבר ראשון בסדרה.

```
int LISLength(int [] arr) // O(n*logn)
      int size = arr.length
      int t[size] // help array
      t[0] = arr[0]
      int lis = 0
      for i=1 to size
            int index = binarySearchBetween(t, lis, arr[i])
            t[index] = arr[i]
            if (index>lis) lis++
      end-for
      return lis+1 //add a first element
end-LISLength
```

<u>הערה:</u> מערך עזר t לא מהווה את התת- סדרה העולה הארוכה ביותר.

```
int binarySearchBetween(int []arr, int end, int value) // O(logn)
      /**
      if value<arr[0] the function returns zero
      if value>arr[end] the function returns end+1
      if arr[index-1] < value < arr[index]
      the function returns index
      */
      int low = 0, high = end
      if (value<arr[0]) return 0
      if (value>arr[end])  return end+1
      while (low <= high)
            int middle = (low + high)/2
            if (low == high) return low // stop search
            else
                  if (arr[middle] == value) return middle //value was found
                  else if (value < arr[middle]) high = middle // value supposes in the left
                  else low = middle+1 // value supposes in the right
                  end-if
            end-if
      end-while
      return -1
end-binarySearchBetween
```

דוגמא:

```
int[]arr = {8, 2, 9, 3, 1, 10, 4, 6, 5, 7}
Initialization: i = 0, t[0] = 8, lis=0, [8, 0,  0, 0, 0, 0, 0, 0, 0, 0]

i = 1, index = 0, element = 2,  lis = 0, [2, 0,  0, 0, 0, 0, 0, 0, 0, 0]
i = 2, index = 1, element = 9,  lis = 1, [2, 9,  0, 0, 0, 0, 0, 0, 0, 0]
i = 3, index = 1, element = 3,  lis = 1, [2, 3,  0, 0, 0, 0, 0, 0, 0, 0]
i = 4, index = 0, element = 1,  lis = 1, [1, 3,  0, 0, 0, 0, 0, 0, 0, 0]
i = 5, index = 2, element = 10, lis = 2, [1, 3, 10, 0, 0, 0, 0, 0, 0, 0]
i = 6, index = 2, element = 4,  lis = 2, [1, 3,  4, 0, 0, 0, 0, 0, 0, 0]
i = 7, index = 3, element = 6,  lis = 3, [1, 3,  4, 6, 0, 0, 0, 0, 0, 0]
i = 8, index = 3, element = 5,  lis = 3, [1, 3,  4, 5, 0, 0, 0, 0, 0, 0]
i = 9, index = 4, element = 7,  lis = 4, [1, 3,  4, 5, 7, 0, 0, 0, 0, 0]
DP: Length of LIS = 5
```

4.2 מציאת תת -סדרה עולה ארוכה ביותר

אלגוריתם:

האלגוריתם למציאת תת סדרה עולה ארוכה ביותר דומה לאלגוריתם של מציאת `lis` - אבל כאן אנו צריכים לזכור את התת סדרות הארוכות ביותר שמתקבלות במהלך החישובים .למטרה זו נצטרך להשתמש במטריצה - במערך דו ממידי בגודל n*n.
במטריצה שנבנה נשתמש במשולש מיושר לשמאל, כאשר האלכסון יכיל את הסדרה שהתקבלה בחישוב lis והשורה שמספרה ה-lis תכיל את הסדרה העולה הארוכה ביותר.
בשלב האתחול האיבר הראשון במטריצה הוא האיבר הראשון במערך הנתון.
כעת נעבור על כל האיבר הבא במערך המקור וימצא מיקומו באלכסון:

- אם הוא קטן מהאיבר הראשון, הוא יחליף את האיבר הראשון במטריצה,
- אם הוא גדול מהאיבר האחרון באלכסון באותה שורה, הוא יוסף בהמשך האלכסון
- באותה שורה ויוקדם `lis`: `lis++`,
- אם הוא בין לבין נשתמש בחיפוש בינארי למציאת מיקום האיבר באלכסון: אם האיבר ימצא, לא נעשה דבר, אם הוא לא ימצא יוחזר האינדקס המתאים לו.

בכל איטרציה נעתיק את האבירים מהשורה שמעל לשורה של מיקום האיבר הנבדק ועד לאלכסון.
בסיום מקדמים את ה-lis ב-1 כי לא נחשב איבר ראשון בסדרה.

```
int[] longestIncrSubseq (int [] arr) // O(n*logn+n^2)=O(n^2)
      int size = arr.length
```

```
        int mat [][] = new int[size][size]
        mat [0][0] = arr[0]
        int lis = 0
        for (int i=1; i<size; i++)
                int index = binarySearchBetween(mat, lis, arr[i])
                for(int j=0; j<index; j++)  mat[index][j]=mat[index-1][j]
                mat[index][index] = arr[i]
                if (index>lis) lis++
        end-for
        int ans[] = new int[lis+1];
        for(int j=0; j<=lis; j++)  ans[j]=mat[lis][j]
        return ans
end-longestIncrSubseq

int binarySearchBetween(int [][]arr, int end, int value)
        /**
        if value<arr[0] the function returns zero
        if value>arr[end] the function returns  end+1
        if arr[index-1] < value < arr[index]
        the function returns index,
        */
        int low = 0, high = end
        if (value<arr[0][0]) return 0
        if (value>arr[end][end])  return end+1
        while (low <=high)
                int middle = (low + high)/2
                if (low == high) return low
                else
                        if (arr[middle][middle] == value) return middle
                        else if (value < arr[middle][middle]) high = middle
                        else low = middle+1
                        end-if
                end-if
        end-while
        return -1
end-binarySearchBetween

int[]arr = {8, 2, 9, 3, 1, 10, 4, 6, 5, 7}
Initialization: i = 0, arr[0][0] = 8, lis=0

i = 1, element = 2, 2<8 → index = 0 → arr[0][0]=2
2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

i = 2, element = 9,
9>2 → index = 1 → arr[1][1] = 9, arr[1][0] = arr[0][0]=2
2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
2, 9, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

i = 3, element = 3,
middle = (0+1)/2 = 0 → arr[middle][middle]<3 → low=0+1=1 → low==start → index = 1 → arr[1][1]=3,
arr[1][0] = arr[0][0]=2
2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
2, 3, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

i = 4, element = 1, 1<2 → index = 0 → arr[0][0]=1
1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
2, 3, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

i = 5, element = 10,
10>3 → index = 2, arr[2][2] = 10, arr[2][0] = arr[1][0]=2, arr[2][1] = arr[1][1]=3
1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
2, 3, 0, 0, 0, 0, 0, 0, 0, 0,
2, 3, 10, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

i = 6, element = 4
middle = (0+2)/2=1 → arr[middle][middle]<4 → low=1+1=2 → low==start → index = 2 → arr[2][2]=4,
arr[2][0] = arr[1][0]=2, arr[2][1] = arr[1][1]=3
1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
2, 3, 0, 0, 0, 0, 0, 0, 0, 0,
2, 3, 4, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
i = 7, index = 3, element = 6
1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
2, 3, 0, 0, 0, 0, 0, 0, 0, 0,
2, 3, 4, 0, 0, 0, 0, 0, 0, 0,
2, 3, 4, 6, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

i = 8, index = 3, element = 5
1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
2, 3, 0, 0, 0, 0, 0, 0, 0, 0,
2, 3, 4, 0, 0, 0, 0, 0, 0, 0,
2, 3, 4, 5, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

i = 9, index = 4, element = 7
1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
2, 3, 0, 0, 0, 0, 0, 0, 0, 0,
2, 3, 4, 0, 0, 0, 0, 0, 0, 0,
2, 3, 4, 5, 0, 0, 0, 0, 0, 0,
2, 3, 4, 5, 7, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

Longest increasing subsequence: [2, 3, 4, 5, 7]
```

חישוב תת-סדרה יורדת ארוכה ביותר:

```
longestDecrSubseq (arr[])//O(n²)
      n = arr.length
      t[n]
      for i = 0 to n-1
            t[i] = - arr[i]
      end-for
      ans[] = longestIncrSubseq(t)
      for i = 0 to n-1
            ans[i] = -ans[i]
      end-for
      return ans
end-longestDecrSubseq
```