بسمه تعالی

دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر

پاسخنامه تمرین سری چهارم درس طراحی الگوریتم‌ها

1. ⟨1,0,0,1,1,0⟩

2. Given a list of numbers L, make a copy of L called L' and then sort L'.

Algorithm 1 PRINT-LCS(c,X,Y)

```
$n = c[X.length, Y.length]$
Initialize an array $s$ of length $n$
$i = X.length$ and $j = Y.length$
while $i > 0$ and $j > 0$ do
    if $x_i == y_j$ then
        $s[n] = x_i$
        $n = n - 1$
        $i = i - 1$
        $j = j - 1$
    else if $c[i-1, j] \geq c[i, j-1]$ then
        $i = i - 1$
    else
        $j = j - 1$
    end if
end while
for $k = 1$ to $s.length$ do
    Print $s[k]$
end for
```

Algorithm 2 MEMO-LCS-LENGTH-AUX(X,Y,c,b)

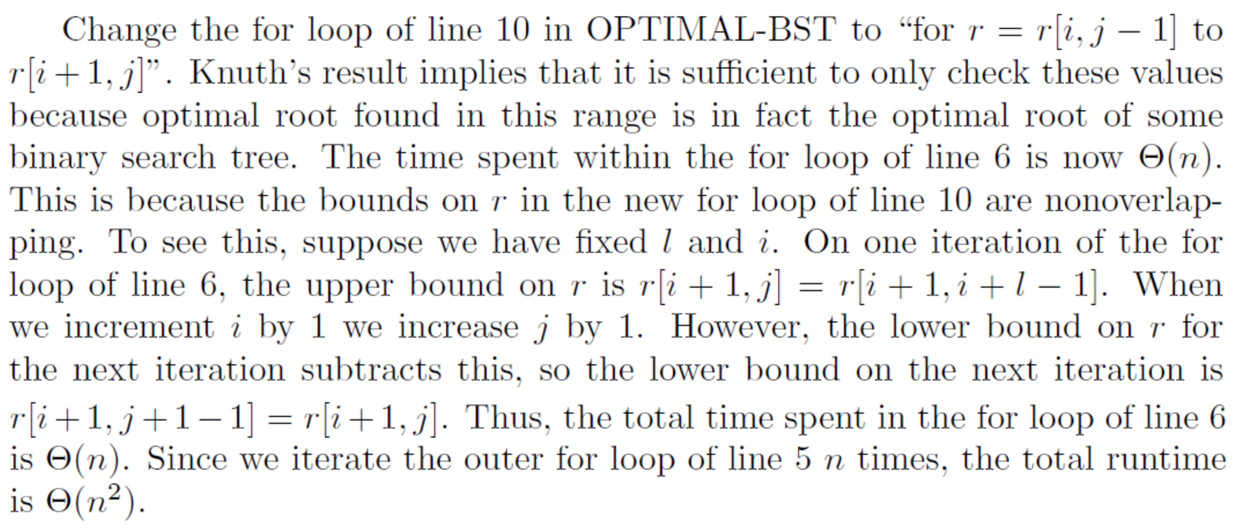
```
m = $|X|$
n = $|Y|$
**if** $c[m,n] != 0$ or $m == 0$ or $n == 0$ **then**
    **return**
**end if**
**if** $x_m == y_n$ **then**
    $b[m,n] = \nwarrow$
    c[m,n] =MEMO-LCS-LENGTH-AUX(X[1,. . . , m-1],Y[1,. . . ,n-1],c,b) +1
**else if** $MEMO - LCS - LENGTH - AUX(X[1, \ldots, m-1], Y, c, b) \geq MEMO - LCS - LENGTH - AUX(X, Y[1, \ldots, n-1], c, b)$ **then**
    $b[m,n] = \uparrow$
    c[m,n] =MEMO-LCS-LENGTH-AUX(X[1,. . . , m-1],Y,c,b)
**else**
    $b[m,n] = \leftarrow$
    c[m,n] =MEMO-LCS-LENGTH-AUX(X,Y[1,. . . ,n-1],c,b)
**end if**
```

Algorithm 3 MEMO-LCS-LENGTH(X,Y)

```
let c be a (passed by reference) $|X|$ by $|Y|$ array initiallized to 0
let b be a (passed by reference) $|X|$ by $|Y|$ array
MEMO-LCS-LENGTH-AUX(X,Y,c,b)
**return** c and b
```

Then, just run the LCS algorithm on these two lists. The longest common subsequence must be monotone increasing because it is a subsequence of L' which is sorted. It is also the longest monotone increasing subsequence because being a subsequence of L' only adds the restriction that the subsequence must be monotone increasing. Since $|L| = |L'| = n$, and sorting L can be done in $o(n^2)$ time, the final running time will be $O(|L||L'|) = O(n^2)$.

3.



Change the for loop of line 10 in OPTIMAL-BST to "for $r = r[i, j-1]$ to $r[i+1, j]$". Knuth's result implies that it is sufficient to only check these values because optimal root found in this range is in fact the optimal root of some binary search tree. The time spent within the for loop of line 6 is now $\Theta(n)$. This is because the bounds on r in the new for loop of line 10 are nonoverlapping. To see this, suppose we have fixed l and i. On one iteration of the for loop of line 6, the upper bound on r is $r[i+1, j] = r[i+1, i+l-1]$. When we increment i by 1 we increase j by 1. However, the lower bound on r for the next iteration subtracts this, so the lower bound on the next iteration is $r[i+1, j+1-1] = r[i+1, j]$. Thus, the total time spent in the for loop of line 6 is $\Theta(n)$. Since we iterate the outer for loop of line 5 n times, the total runtime is $\Theta(n^2)$.

4. For Greedy algorithm to be optimal, the following relation between coins and the given value must exist:

$$V = n_1 c_1 + n_2 c_2 + C$$

Where V is given value, $c_1 \, and \; c_2$ are the coin values, $n_1 \; and \; n_2$ are non negative integers and C is zero.