1. Let +i denote "insert item i into a red-black tree using the book's (or class') code"; let -i denote "delete item i into a red-black tree using the book's (or class') code". Show the RB-trees that result after each operation in the following sequence of inserts and deletes: +1 +4 +7 +10 -4 +2 +5 +8 +11 -8 +3 +6 +9 +12.

Insert 1

1

Insert 4

1

4

Insert 7

1

4

7

After Rotation

4

1 7

Insert 10

4

1 7

10

After Rotation

4

1 7

10

Delete 4

1

7

10

Post Rotation

7

1 10

Insert 2

7

1 10

2

Insert 5

7

1 10

2

5

Post Rotation

7

2 10

1 5

Insert 8

7

2 10

1 5 8

Insert 11

7

2 10

1 5 8 11

Delete 8

7

2 10

1 5 11

Insert 3

7

2 10

1 5 11

3

Adjust Color

7

2 10

1 5 11

3

Insert 6

7

2 10

1 5 11

3 6

Insert 9

7

2 10

1 5 9 11

3 6

Insert 12

7

2 10

1 5 9 11

3 6 12

Adjust Color

7

2 10

1 5 9 11

3 6 12

2. Consider the two sequences: ⟨0,1,0,0,1,0,1⟩ and ⟨1,1,0,1,0,0,1,1,0⟩. Give the tables that would be constructed by the LCS-length procedure. Then show step-by-step how PRINT-LCS would use these table to build a longest common sub-sequence

NOTE : * implies arrow to the upper left cell

Table:

		0	**1**	**0**	**0**	**1**	**0**	**1**
	0	0	0	0	0	0	0	0
1	0	0	*1	1	1	*1	1	*1
1	0	0	*1	1	1	*2	2	*2
0	0	*1	1	*2	*2	2	*3	3
1	0	1	*2	2	2	*3	3	*4
0	0	*1	2	*3	*3	3	*4	4
0	0	*1	2	*3	*4	4	*4	4
1	0	1	*2	3	4	*5	5	*5
1	0	1	*2	3	4	*5	5	*6

0	0	*1	2	*3	*4	5	*6	6

There are 6 elements in the LCS

LCS: <0,1,0,0,1,0>

with size 6

Print LCS

1:

LCS will start by printing in the lower right corner of the table so that it can print the LCS in forward order using Recurisive calls.

2:

The Initial call is Print-LCS of (b,X,X.length,Y.length) then this method is recursively called the index of x by 1 or the index of y by 1, until base case is reached.

3. Consider the instance of the activity-selector problem given by the table below:

i	1	2	3	4	5	6	7	8
si	2	3	0	4	5	7	8	2
fi	5	6	7	7	8	10	11	11

Show how the greedy recursive algorithm, RECURSIVE-ACTIVITY-SELECTOR, solves this problem.

Mutually Compatible Selection

<a(1), a(5), a(7)>

3 activities.

K	S(k)	F(k)		
0		0 a(0)		
1	2	5	a(1)m=1	
2	3	6	a(1):a(2)	False

3	0	7	a(1):a(3)	False
4	4	7	a(1):a(4)	False
5	5	8	a(1):a(5)	True
6	7	10	a(1):a(5):a(6)	False
7	8	11	a(1):a(5):a(6)	True
8	2	11	a(1):a(5):a(8)	False

a(1) : 1 to 5

a(5) : 5 to 8

a(7) : 8 to 11

4. Prove that the Knapsack problem discussed in class is in NP. Here a Knapsack instance consists of a set of a table of items sizes s(1), ..., s(n); a table of values of each item v(1), ..., v(n), a knapsack size *m* and a goal value *k*. An instance is in Knapsack if this is a subset of the items such that the sum of their sizes is less than *m* and the value of these items is greater than *k*.

To prove the problem is NP, for any instance of selected items you would add up to the size of s(x) of the items, and the values v(x) of the selected items, and compare the sums with n and k. If the sum of the size is at most m and if the sum of the values is at least k, then it is an instance of knapsack.

This can be denoted as polynomial time as O(x), there is only one pass through the proposed instance is needed to determine if it would actually be an instance.

x = the number of the selected items of the set, and a potential instance of knapsack

v = is the size of the items J

y = is the value of the item in J

v = 0 , y = 0

v = v + s(x)

y = y + v(x)

if v > m || y < k

then x would be a knapsack

if not then it is not a knapsack