Corey Kipp (Student ID: 57723335)

kippc@uci.edu

CS 161

Homework 6

R-12.4

(9,1,7,3,2,8,9,3)

A will choose 9 and B will choose 1

A will choose 7 and B will choose 3

A will choose 3 and B will choose 9

A will choose 8 and B will choose 2

A = 9 + 7 + 3+ 8 = 27

B = 1 + 3 + 9 + 2 = 15

So Alice can win a maximum of 27 compared to Bob's 15, for difference of 12.

R-12.8

	0	L	U	L	L	A	B	Y	B	A	B	I	E	S
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	1
K	0	0	0	0	0	0	0	0	0	0	0	0	0	1
U	0	0	**1**	1	1	1	1	1	1	1	1	1	1	1
L	0	1	1	**2**	2	2	2	2	2	2	2	2	2	2
L	0	1	1	2	**3**	3	3	3	3	3	3	3	3	3
A	0	1	1	2	3	**4**	4	4	4	4	4	4	4	4
N	0	1	1	2	3	4	4	4	4	4	4	4	4	4
D	0	1	1	2	3	4	4	4	4	4	4	4	4	4

B	0	1	1	2	3	4	**5**	5	5	5	5	5	5	5
O	0	1	1	2	3	4	5	5	5	5	5	5	5	5
N	0	1	1	2	3	4	5	5	5	5	5	5	5	5
E	0	1	1	2	3	4	5	5	5	5	5	5	**6**	6
S	0	1	1	2	3	4	5	5	5	5	5	5	6	**7**

The longest common sequence is “ullabes” which has a length of 7.

C-12.3

In the game (20,50,40,5) if Alice first plays only in such a way that she minimizes the max move for Bob it would play like:

A will choose 5 and B will choose 40

A will choose 50 and B will choose 20

A = 5 + 50 = 55

B = 40 + 20 = 60

So in this case Bob wins.

However if she chose not to adopt this strategy but instead did something like minimax then the following game would result:

A will choose 20 and B will choose 50

A will choose 40 and B will choose 5

A = 20 + 40 = 60

B = 50 + 5 = 55

So in the case Alice wins.

A-12.9

```
L = [-3, 10, 2, -5, 7, -1, -2, -1, -3]
n = len(L)
T = [[None]*n for k in range(n)]

def houses(i, j, s):

    if T[i][j] is not None and (s==1):
        return T[i][j]
    if i==j and s==1:
        T[i][j] = L[i]
        return L[i]
    if i==j and s==0:
        take = L[i]
        dont = -L[i]
        return max(take,dont)
    else:
        takeLeft = L[i] + houses(i+1, j, 0)
        takeRight = L[j] + houses(i, j-1, 0)
        takeNone = None
        if (s == 0):
            takeNone = 0 - houses(i, j, 1)
        if (takeNone != None):
            if ((takeNone >= takeRight) and (takeNone >= takeLeft)):
                T[i][j] = takeNone
                return T[i][j]
            elif ((takeLeft >= takeRight) and (takeLeft >= takeNone)):
                T[i][j] = takeLeft
                return T[i][j]
            elif ((takeRight >= takeLeft) and (takeRight >= takeNone)):
                T[i][j] = takeRight
                return T[i][j]
        elif (takeNone == None):
            if (takeLeft >= takeRight):
                T[i][j] = takeLeft
                return T[i][j]
            else:
                T[i][j] = takeRight
                return T[i][j]


        return T[i][j]

houses(0,n-1,1)

for row in T:
    for elem in row:
        print(elem,end='\t')
    print()
print()
```

The idea behind the algorithm is that in order to properly compute the best move for the player they must recurse through every possibility in order to figure out when to stop taking houses, since there might be the possibility that taking every house in one turn is the best outcome. In order to be optimal you have to check that there isn't some amazing house value a few down the line that could allow you to win, therefore it is necessary to check every possible move you can make in every turn. This will result in a running time of **O(n^(number of houses))**.