Homework #3
Student: Alice Xiong
Course: Foundations of Algorithms
Date: Dec 8, 2015
Cellphone: 312-646-9280

1.

(a) Algorithm 1 doesn't provide an optimal path, see one example below:

	week 1	week 2	week 3
L	10	10	10
h	5	50	100

The optimal revenue and path achieved by algorithm 1 will be 0 + 50 + 10 = **60** with path None→h2→L3.

- Since h2 > L1+L2, so "none" in **week 1**, a high stress job in **week 2**,
- Continue with week 3, h4=L4=0 and h4 < L3+L4, so pick up a low stress job in **week 3**.

However, the correct optimal path should achieve maximum revenue of 10 + 0 + 100 = 110 with path L1 → None → h3.

(b) we will utilize dynamic programing to solve it with steps described below:

step1: characterize an optimal substructure of job plans

let **c(j)** be the maximum revenue received from week 1 until week j.

step2: a recursive solution

The optimal substructure of the job plans gives the recursive formula:

initialization: c(1) = max (L1, h1),

c(2) = max(L1+L2, h2)

c(j) = max { c(j-1) + Lj, c(j-2)+hj } for $2 < j \leq n$

b[j] = L or h, dependent on which job selected on week j for $1 \leq j \leq n$

Table b is to help us construct an optimal job plans. See the pseudo code below how table b has been updated to trace back the path.

step3: compute the cost of an optimal solution

It takes constant time to update each c[j] and b[j]. So the running time will be O(2n)=O(n).

```
def job_plan(L, h):
        n = len(h)
        c = np.zeros(n)  #empty array c store the numerical revenue
  #empthy array b store the string "L" and "h"; help us construct an optimal job plans later
        b = map(str, np.zeros(n))

        if L[0] >= h[0]:
            c[0] = L[0]; b[0] = "L"
        else: c[0] = h[0]; b[0] = "h"

        if h[1] > L[0] + L[1]:
            c[1] = h[1]; b[1] = "h"; b[0] = None
        else: c[1] = L[0] + L[1]; b[1] = "L"; b[0] = "L"

        for j in range(2,n):      #O(n)
            if c[j-1] + L[j] >= c[j-2]+h[j]:
                    c[j] = c[j-1] + L[j]
                    b[j] = "L"
            else:
                    c[j] = c[j-2] + h[j]
                    b[j] = "h"
                    b[j-1] = None
                    b[j-2] = "L"
# array has range between 0 and n-1, so c[n-1] means the maximum revenue in week n
        return c[n-1], b
```

step4: print out the optimal job solution from the maintained table b. It takes time O(n).

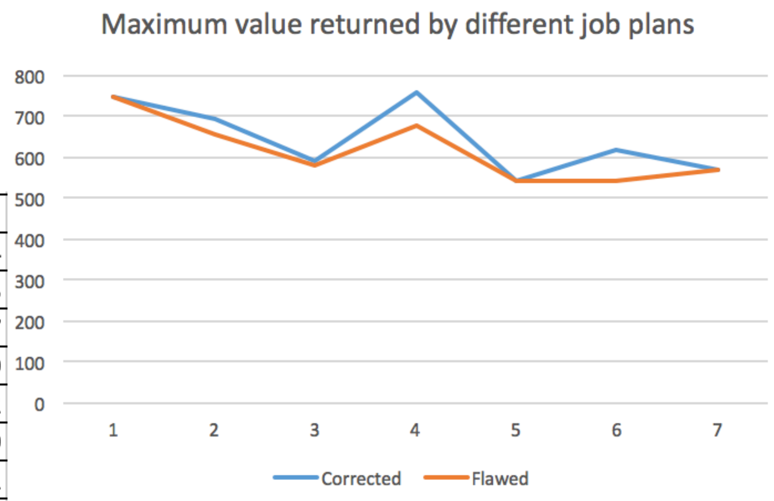
```
def print_job_plan(b):
        n = len(b)
        out = []
        for i in range(n):
            if b[i] != None:
                  out.append(b[i]+str(i+1))
            else:
                  out.append("None")
        print ' --> '.join(out)
```

2. in jobPlan.log within the attached zip files, you can find 7 simulated examples for L and h with the update revenue in each step.

```
    for i in range(7):
          weeks = 10
          L = random.sample(range(1,100),weeks)
          h = random.sample(range(10, 150), weeks)
```

Below is the summarized table with the final revenue achieved in week10 by both the corrected algorithm and flawed algorithm. You can see their detail job plan path in the trace log.

Corrected	Flawed
744	744
691	653
592	577
755	679
542	542
619	539
571	571


Maximum value returned by different job plans
Corrected
Flawed