Seyfullah Becerikli
171044028

CSE 321 - Homework 4

2)
```
myAlgorithm(A[1, ....,n])
  if n==1
    return A[n]==n
  x = n/2
  if A[x] = x
    return true
  else if A[x]>x
    return myAlgorithm(A[1,.....,x-1])
  else
    return myAlgorithm(A[x+1,.....n]-m)
```

4)
```
minmax(A[s,..b],min,max)
  if s==b
    min = A[s]
    max = A[s]
  else if s-b =1
    if A[s] < A[1]
      min = A[s]
      max = A[b]
    else
      min = A[b]
      max = A[s]
  else
    minMax(A[s..⌊(s+b)/2⌋],min,max)
    minmax(A[⌊(s+b)/2⌋+1...b],min2,max2)
    if min2<min
      min = min2
    if max2>max
      max = max2
```

5) Bir problemi çözen algoritmanın operasyonlarını full binary tree ile gösterebiliriz. Bizim algoritmamızda parent nodlar kırılan parçaları ve leafler orjinal çikolata barın birer parçasını göstermektedir. Sonucun sayısı nm dir ve sonucun numarası kırılan çikolata parçalarına eşittir ve nm-1 dir

Bir seferde sadece bir parça çikolata kırılabildiğinden her çikolata kırma işlemi parça sayısını bir arttırır. Buyüzden nm-1 kırma işlemi tek bir nxm parçadan elde edilmelidir. Buyüzden herhangi bir sırada nm-1 kırma işlemi kabul edilebilir

6) a) Dynamic programming büyük problemleri küçük parçalara böldüğü zamanlarda divide and conquer ile benzer.

b) Dyn.Prog. büyük problemleri küçük parçalara böler fakat bu küçük parçalar sadece bir kez çözülür ve cevapları bir tablada saklanır. Divide and conqure da ise küçük parçalar recursive olarak çözülür ve daha sonra birleştirilir.

7) a) A takımının kazanması için i oyun ve B takımının kazanması için j oyun kazanması gerekşin ve bunun olasılığı $P(i,j)$ olsun Eğer A takımı kazanırsa (p olasılıkla) A takımı i-1 oyun daha fazla kazanması lazım kazanabilmesi için fakat B hala j oyun kazanması gerek. Eğer A takımı kaybederse ($q=1-p$ olasılıkla) A i oyun kazanması gerekirken B nin j-1 oyun kazanması gerekmekte. Bu recursiona sebep olur.

$$P(i,j) = pP(i-1,j) + qP(i,j-1) \quad , i,j > 0$$

Başlangıç koşullarında sonra

$$P(0,j) = 1 \quad j>0 \quad , \quad P(i,0) = 0 \quad i>0$$

b)

i/j	0	1	2	3	4
0		1	1	1	1
1	0	0.4	0.64	0.78	0.87
2	0	0.16	0.35	0.52	0.66
3	0	0.06	0.18	0.32	0.46
4	0	0.03	0.09	0.18	0.29

$P(1,1) = p \cdot P(0,1) + q \cdot P(1,0)$

$0,4 = 0,4 \cdot 1 + 0$

$P(1,2) = p \cdot P(0,2) + q \cdot P(1,1)$

$0,64 = 0,4 \cdot 0,6 + 0,4$

c)
```
worldSeries(n,p)
  q = 1 - p
  for j = 1 to n do
    P[0,j] = 1.0
  for i = 1 to n do
    P[i,0] = 0.0
    for j = 1 to n do
      P[i,j] = p * P[i-1,j] + q * P[i,j-1]
  return P[n,n]
```

3)
```
def tower(height, from, to, with)
    if height >= 1 :
        tower(height-1, from, with, to)
        printX(from, to)
        tower(height-1, with, to, from)

def printX(from, to)
    print("move from", from, "to", to)
```

9) $A_i A_{i+1} \cdots A_j$ tane matrisimiz olsun. Bunlar için $A_i A_{i+1} \cdots A_k$ ve $A_{k+1} \cdots A_j$, $1 \le i \le k < j \le n$ tane paranteze yapabiliriz

Bu matrislerin çarpımının en az maliyetli parantezlenmesinin recursive tanımı

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j \end{cases}$$

şeklindedir.

```
def matrixChain(p, n)
    m=[[0 for x in range(n)] for x in range(n)]
    for i in range(1,n):
        m[i][i]=0
    for L in range(2,n): # L is chain length.
      for i in range(1, n-L+1):
          J = i+L-1
          m[i][j] = sys.maxint
          for k in range(i,j):
              t=m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j]
              if t<m[i][j]:
                  m[i][j]=q
    return m[i][n-1]

arr=[3,5,7,9]
size= 4
print(str(matrixChain(arr,size)))
```