CSE 321 Homework 4

Fatih Dural

151044041

Report

Part1

a) The "only if" part is trivial -- it follows form the definition of Monge array. As for the "if" part, let's first prove that:

$$A[i,j]+A[i+1,j+1]\le A[i,j+1]+A[i+1,j]$$

$$\Downarrow$$

$$A[i,j]+A[k,j+1]\le A[i,j+1]+A[k,j]$$

Where $i<k$.

Let's prove it by induction. The base case of $k=i+1$ is given. As for the inductive step, we assume it holds for $k=i+n$ and we want to prove it for $k+1=i+n+1$. If we add the given to the assumption, we get:

$$\text{A[i,j]+A[k,j+1]}\le\text{A[i,j+1]+A[k,j] (assumption)}$$

$$\text{A[k,j]+A[k+1,j+1]}\le\text{A[k,j+1]+A[k+1,j] (given)}$$

$$\rightarrow\text{A[i,j]+A[k,j+1]+A[k,j]+A[k+1,j+1]}\le\text{A[i,j+1]+A[k,j]+A[k,j+1]+A[k+1,j]}$$

$$\rightarrow\text{A[i,j]+A[k+1,j+1]}\le\text{A[i,j+1]+A[k+1,j]}$$

c) Let ai and bj be the leftmost minimal elements on rows a and b and let's assume that $i>j$. Then we have:

$$A[j,a]+A[i,b]\le A[i,a]+A[j,b]$$

But:

$$A[j,a]\ge A[i,a] \text{ (}ai\text{ is minimal)}$$
$$A[i,b]\ge A[j,b] \text{ (}bj\text{ is minimal)}$$

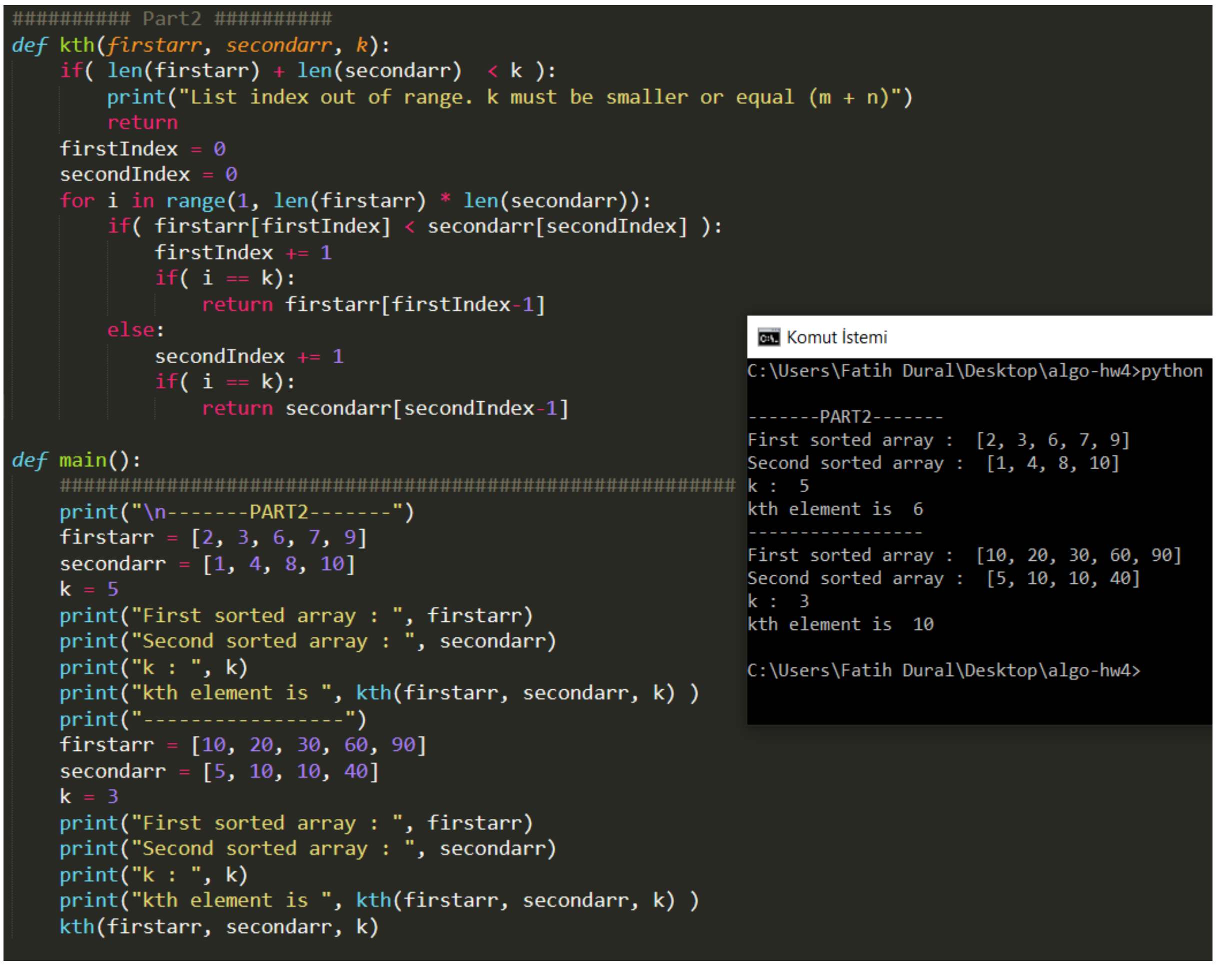
Which implies that:

$$A[j,a]+A[i,b]\ge A[i,a]+A[j,b]$$
$$\Downarrow$$
$$A[j,a]+A[i,b]=A[i,a]+A[j,b]$$

Which in turn implies that either:

$$A[j,b]<A[i,b]\Rightarrow A[i,a]>A[j,a]\Rightarrow ai \text{ is not minimal}$$
$$A[j,b]=A[i,b]\Rightarrow bj \text{ is not the leftmost minimal}$$

Part2



```python
########## Part2 ##########
def kth(firstarr, secondarr, k):
    if( len(firstarr) + len(secondarr)  < k ):
        print("List index out of range. k must be smaller or equal (m + n)")
        return
    firstIndex = 0
    secondIndex = 0
    for i in range(1, len(firstarr) * len(secondarr)):
        if( firstarr[firstIndex] < secondarr[secondIndex] ):
            firstIndex += 1
            if( i == k):
                return firstarr[firstIndex-1]
        else:
            secondIndex += 1
            if( i == k):
                return secondarr[secondIndex-1]

def main():
    ###########################################################
    print("\n-------PART2-------")
    firstarr = [2, 3, 6, 7, 9]
    secondarr = [1, 4, 8, 10]
    k = 5
    print("First sorted array : ", firstarr)
    print("Second sorted array : ", secondarr)
    print("k : ", k)
    print("kth element is ", kth(firstarr, secondarr, k) )
    print("-----------------")
    firstarr = [10, 20, 30, 60, 90]
    secondarr = [5, 10, 10, 40]
    k = 3
    print("First sorted array : ", firstarr)
    print("Second sorted array : ", secondarr)
    print("k : ", k)
    print("kth element is ", kth(firstarr, secondarr, k) )
    kth(firstarr, secondarr, k)
```

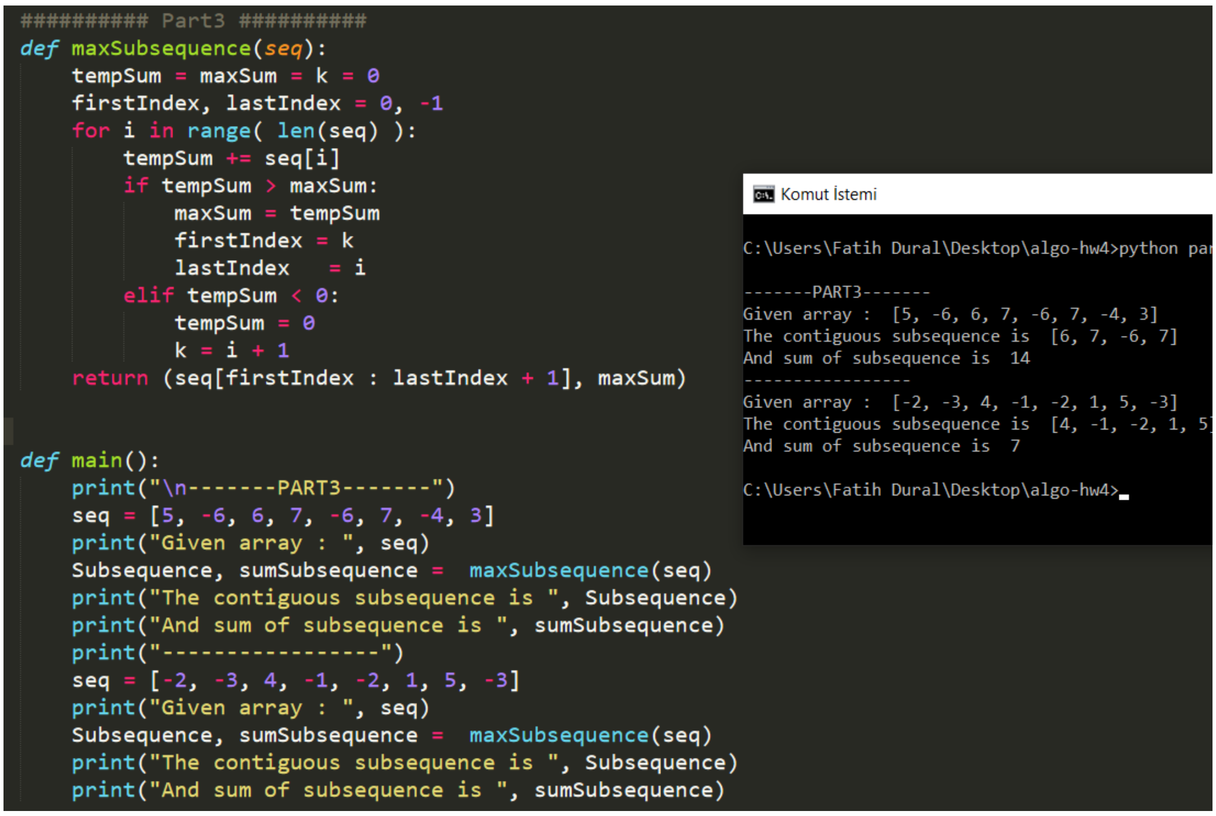
```
C:\Users\Fatih Dural\Desktop\algo-hw4>python

-------PART2-------
First sorted array :  [2, 3, 6, 7, 9]
Second sorted array :  [1, 4, 8, 10]
k :  5
kth element is  6
-----------------
First sorted array :  [10, 20, 30, 60, 90]
Second sorted array :  [5, 10, 10, 40]
k :  3
kth element is  10

C:\Users\Fatih Dural\Desktop\algo-hw4>
```

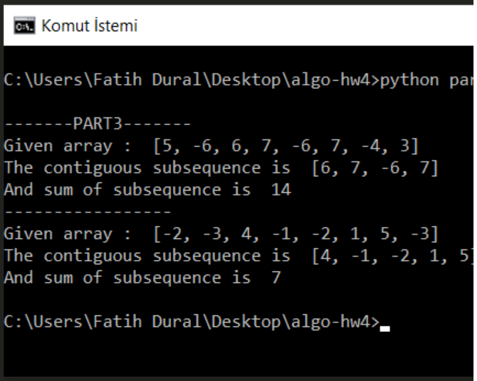
In this part, there are two sorted arrays, it will be designed a divide-and-conquer algorithm that finds the kth element of the merged array of these two sorted arrays. Two sorted arrays are created, and given to function called kth. It controls that k whether the variable exceeds the list size. It starts from the smallest element and continue with it. When it returns k times, the result is found. Test was done twice. Time complexity will be O(n) time and the worst case is the same.

Part3



```python
########## Part3 ##########
def maxSubsequence(seq):
    tempSum = maxSum = k = 0
    firstIndex, lastIndex = 0, -1
    for i in range( len(seq) ):
        tempSum += seq[i]
        if tempSum > maxSum:
            maxSum = tempSum
            firstIndex = k
            lastIndex   = i
        elif tempSum < 0:
            tempSum = 0
            k = i + 1
    return (seq[firstIndex : lastIndex + 1], maxSum)


def main():
    print("\n-------PART3-------")
    seq = [5, -6, 6, 7, -6, 7, -4, 3]
    print("Given array : ", seq)
    Subsequence, sumSubsequence =  maxSubsequence(seq)
    print("The contiguous subsequence is ", Subsequence)
    print("And sum of subsequence is ", sumSubsequence)
    print("-----------------")
    seq = [-2, -3, 4, -1, -2, 1, 5, -3]
    print("Given array : ", seq)
    Subsequence, sumSubsequence =  maxSubsequence(seq)
    print("The contiguous subsequence is ", Subsequence)
    print("And sum of subsequence is ", sumSubsequence)
```



Komut İstemi

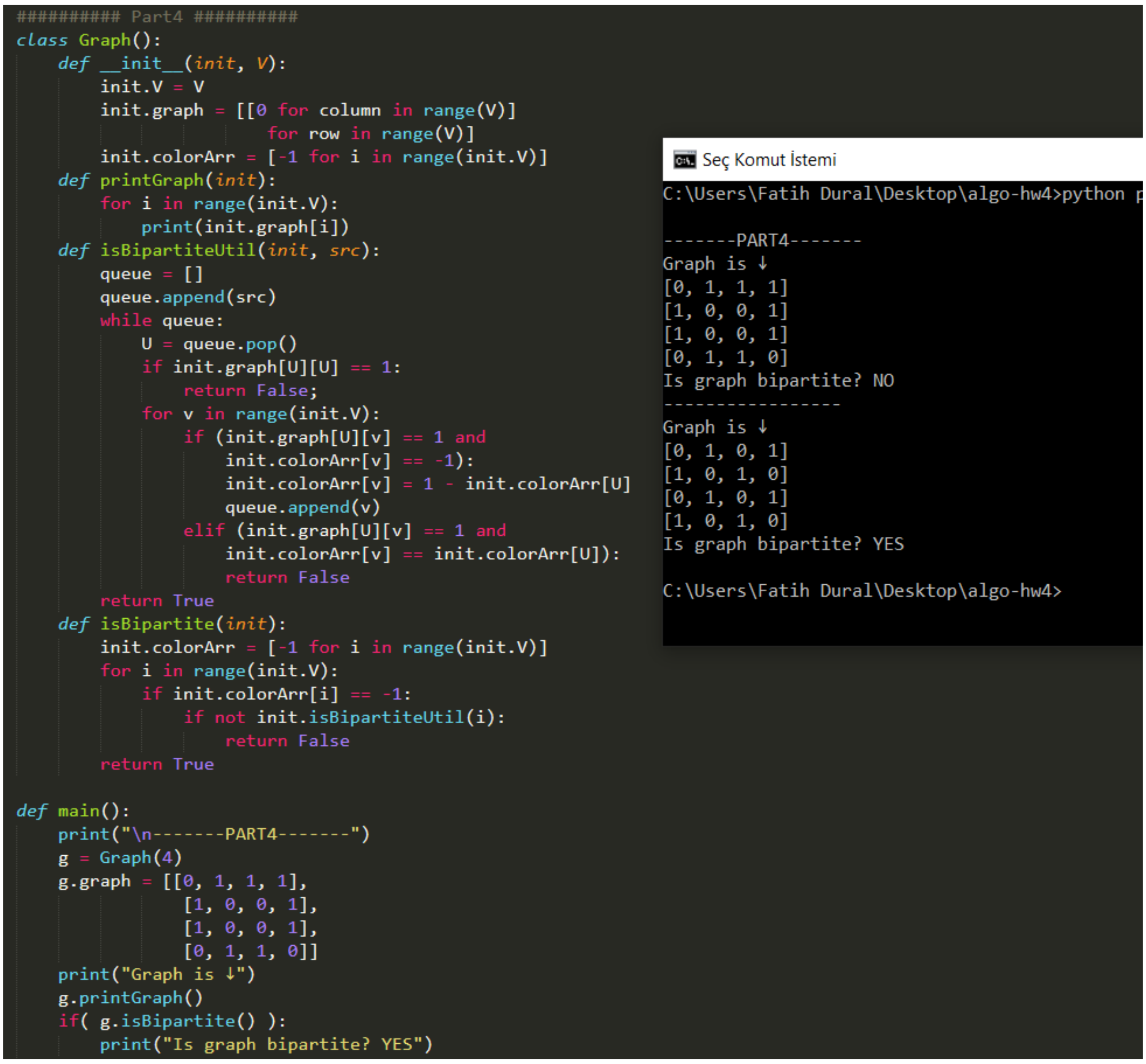
```
C:\Users\Fatih Dural\Desktop\algo-hw4>python par

-------PART3-------
Given array :  [5, -6, 6, 7, -6, 7, -4, 3]
The contiguous subsequence is  [6, 7, -6, 7]
And sum of subsequence is  14
-----------------
Given array :  [-2, -3, 4, -1, -2, 1, 5, -3]
The contiguous subsequence is  [4, -1, -2, 1, 5
And sum of subsequence is  7

C:\Users\Fatih Dural\Desktop\algo-hw4>_
```

In part3, given an array of integers, it will be proposed a divide-and-conquer algorithm that finds a contiguous subset having the largest sum. Received help from kadane’s algorithm. Assumed that the largest subarray is the first element, then we go through all elements except the first one. The elements are added individually and checked to see if they are greater than the previous total value. If it is large it will be the new maxSum. The number of rotations, i, describes the last index. If the current total falls below zero, tempSum is set to 0 and k is updated. At the end of the function, first list and last list are used to determine the subsequence and return the total and sequence. The time complexity of the function is O(n) and the worst case is the same.

Part4



```python
########## Part4 ##########
class Graph():
    def __init__(init, V):
        init.V = V
        init.graph = [[0 for column in range(V)]
                        for row in range(V)]
        init.colorArr = [-1 for i in range(init.V)]
    def printGraph(init):
        for i in range(init.V):
            print(init.graph[i])
    def isBipartiteUtil(init, src):
        queue = []
        queue.append(src)
        while queue:
            U = queue.pop()
            if init.graph[U][U] == 1:
                return False;
            for v in range(init.V):
                if (init.graph[U][v] == 1 and
                    init.colorArr[v] == -1):
                    init.colorArr[v] = 1 - init.colorArr[U]
                    queue.append(v)
                elif (init.graph[U][v] == 1 and
                    init.colorArr[v] == init.colorArr[U]):
                    return False
        return True
    def isBipartite(init):
        init.colorArr = [-1 for i in range(init.V)]
        for i in range(init.V):
            if init.colorArr[i] == -1:
                if not init.isBipartiteUtil(i):
                    return False
        return True

def main():
    print("\n-------PART4-------")
    g = Graph(4)
    g.graph = [[0, 1, 1, 1],
               [1, 0, 0, 1],
               [1, 0, 0, 1],
               [0, 1, 1, 0]]
    print("Graph is ↓")
    g.printGraph()
    if( g.isBipartite() ):
        print("Is graph bipartite? YES")
```

Seç Komut İstemi

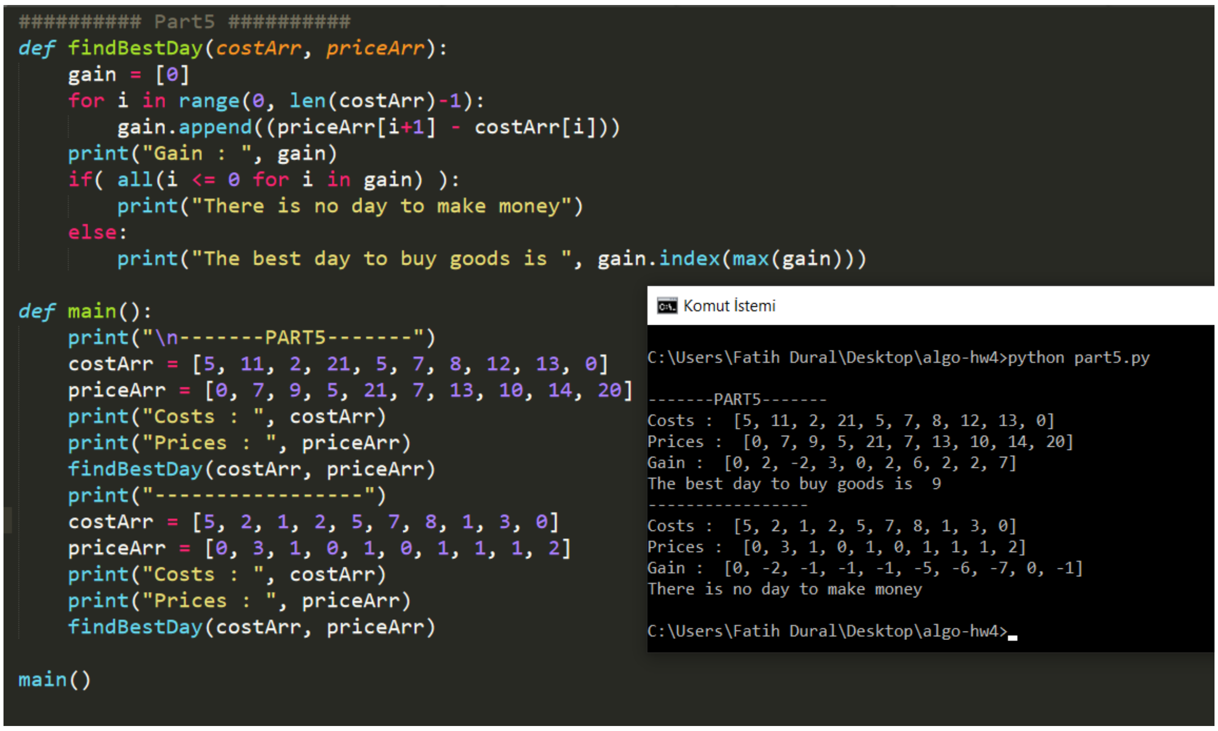
```
C:\Users\Fatih Dural\Desktop\algo-hw4>python p

-------PART4-------
Graph is ↓
[0, 1, 1, 1]
[1, 0, 0, 1]
[1, 0, 0, 1]
[0, 1, 1, 0]
Is graph bipartite? NO
-----------------
Graph is ↓
[0, 1, 0, 1]
[1, 0, 1, 0]
[0, 1, 0, 1]
[1, 0, 1, 0]
Is graph bipartite? YES

C:\Users\Fatih Dural\Desktop\algo-hw4>
```

A bipartite graph is a graph whose vertices can be divided into two disjoint and independent sets U and V such that every edge connects a vertex in U to one in V. It will be designed a decrease and conquer algorithm that checks whether a given graph is a bipartite graph or not. In the algorithm, Assign red color to the source vertex (putting into set U). Color all the neighbors with blue color (putting into set V). Color all neighbor's neighbor with red color (putting into set U). This way, assign color to all vertices such that it satisfies all the constraints of m way coloring problem where m = 2. While assigning colors, if we find a neighbor which is colored with same color as current vertex, then the graph cannot be colored with 2 vertices (or graph is not Bipartite). Time complexity of the above approach is same as that Breadth First Search and it is O(V^2) where V is number of vertices.

Part5



```
########## Part5 ##########
def findBestDay(costArr, priceArr):
    gain = [0]
    for i in range(0, len(costArr)-1):
        gain.append((priceArr[i+1] - costArr[i]))
    print("Gain : ", gain)
    if( all(i <= 0 for i in gain) ):
        print("There is no day to make money")
    else:
        print("The best day to buy goods is ", gain.index(max(gain)))

def main():
    print("\n-------PART5-------")
    costArr = [5, 11, 2, 21, 5, 7, 8, 12, 13, 0]
    priceArr = [0, 7, 9, 5, 21, 7, 13, 10, 14, 20]
    print("Costs : ", costArr)
    print("Prices : ", priceArr)
    findBestDay(costArr, priceArr)
    print("-----------------")
    costArr = [5, 2, 1, 2, 5, 7, 8, 1, 3, 0]
    priceArr = [0, 3, 1, 0, 1, 0, 1, 1, 1, 2]
    print("Costs : ", costArr)
    print("Prices : ", priceArr)
    findBestDay(costArr, priceArr)

main()
```

```
C:\Users\Fatih Dural\Desktop\algo-hw4>python part5.py

-------PART5-------
Costs :  [5, 11, 2, 21, 5, 7, 8, 12, 13, 0]
Prices :  [0, 7, 9, 5, 21, 7, 13, 10, 14, 20]
Gain :  [0, 2, -2, 3, 0, 2, 6, 2, 2, 7]
The best day to buy goods is  9
-----------------
Costs :  [5, 2, 1, 2, 5, 7, 8, 1, 3, 0]
Prices :  [0, 3, 1, 0, 1, 0, 1, 1, 1, 2]
Gain :  [0, -2, -1, -1, -1, -5, -6, -7, 0, -1]
There is no day to make money

C:\Users\Fatih Dural\Desktop\algo-hw4>_
```

In the last part, we are asked to you manage a warehouse: buy some goods, store the goods for one day and sell them the next day. The cost of storing the goods changes daily depending on the occupancy ratioin the warehouse. The selling price of the goods varies according to markets. It will be designed a divide-and-conquer algorithm that finds the best day to buy the goods. Cost and price arrays are created. According to problem, the last element of cost is zero and the first element of price is zero. The gain consists of subtracting the element in the cost sequence from the next element in the price sequence. After doing this for all elements, the gain sequence holds all the gains. The maximum number in the gain sequence is the maximum gain. If all elements are less than or equal to zero, there is no day to gain. This is indicated as an extra. The time complexity and worst case running time would be O(n).

Fatih DURAL

151044041