Home Work 7

Ques 1. You are given an Integer array a[1], ..., a[n], find the contiguous subarray (containing at least one number) which has the largest sum and only return its sum. The optimal subarray is not required to return or compute. Taking a = [5, 4, -1, 7, 8] as an example: the subarray [5] is considered as a valid subarray with sum 5, though it only has one single element; the subarray [5, 4, -1, 7, 8] achieves the largest sum 23; on the other hand, [5, 4, 7, 8] is not a valid subarray as the number 4 and 7 are not contiguous.

(a) Define (in plain English) subproblems to be solved.
(b) Write a recurrence relation for the subproblems.
(c) Using the recurrence formula in part b, write pseudocode to find the subarray (containing at least one number) which has the largest sum.
(d) Make sure you specify
 i. base cases and their values
 ii. where the final answer can be found.
(e) What is the complexity of your solution?

Ans 1. (a) array[i] - sum of maximum contiguous subarray

(b) Recurrence relation -

```
array[i] = max(array[i-1] + a[i], a[i])
if array[i-1] + a[i] > a[i]:
        max_sum += [a[i])
else:
    max_sum = [a[i]]
```

(c) Pseudocode -

```
array = [0 for i in range(len(a))]
array[0] = nums[0]
max_sum = list()
for i in range(1, len(a)):
    array[i] = max(array[i-1] + a[i], a[i])
    if array[i-1] + a[i] > a[i]:
        max_sum += [a[i]]
    else:
        max_sum = [a[i]]
```

(d) (i) array = [0 for i in range(len(a))]
array[0] = nums[0]

(ii) max_sum list will have the final answer

(e) Time complexity is $\theta(n)$

Ques 2 You've started a hobby of retail investing into stocks using a mobile app, RogeGood. You magically gained the power to see N days into the future and you can see the prices of one particular stock. Given an array of prices of this particular stock, where prices[i] is the price of a given stock on the ith day, find the maximum profit you can achieve through various buy/sell actions. RogeGood also has a fixed fee per transaction. You may complete as many transactions as you like, but you need to pay the transaction fee for each transaction.

(a) Define (in plain English) subproblem to be solved.
(b) Write a recurrence relation for the subproblems.
(c) Using the recurrence formula in part b, write pseudocode to solve the problem.
(d) Make sure you specify
    i. base cases and their values
    ii. where the final answer can be found
(e) What is the complexity of your solution?

Ans 2. (a) buy(i) - max. profit at day i start when you do not own stock
sell(i) - max. profit to start at i when you own a stock

(b) Recurrence relation:

$$buy[i] = \max(sell[i+1] - prices[i],\ buy[i+1])$$
$$sell[i] = \max(buy[i+1] + prices[i] - fee,\ sell[i+1])$$

(c)

```
n = len(prices)
buy = [0 for i in range(n+1)]
sell = [0 for i in range(n+1)]
for i = n-1 to 0 do
    buy[i] = max(sell[i+1] - prices[i], buy[i+1])
    sell[i] = max(buy[i+1] + prices[i] - fee, sell[i+1])
end for
return buy[0]
```

(d)

```
buy = [0 for i in range(n+1)]
sell = [0 for i in range(n+1)]
```

buy[0] will have the final answer.

(e) Time complexity of the solution is $\Theta(n)$

Ques3. You are given an array of positive numbers $a[1], \ldots, a[n]$. For a sub-sequence $a[i_1], a[i_2], \ldots, a[i_t]$ of array a (that is, $i_1 < i_2 < \ldots, i_t$): if it is an increasing sequence of numbers, that is $a[i_1] < a[i_2] < \ldots < a[i_t]$, its happiness score is given by

$$\sum_{k=1}^{t} k \times a[i_k]$$

Otherwise, the happiness score of this array is zero.

For example, for the input $a = [22, 44, ~~60~~ 33, 66, 55]$, the increasing subsequence $[22, 44, 55]$ has happiness score $(1)\times(22) + 2\times(44) + (3)\times(55) = 275$; the increasing subsequence $[22, 33, 55]$ has happiness score $(1)\times(22) + (2)\times(33) + (3)\times(55) = 253$; ~~the increasing~~ the ~~subsequence~~ subsequence $[33, 66, 55]$ has happiness score 0 as this sequence is not increasing. Please design an efficient algorithm to only return the highest happiness score over all the subsequences.

(a) Define (in plain English) subproblem to be solved.
(b) Write a recurrence relation for the subproblem.
(c) Using the recurrence formula in part b, write pseudocode to find the highest happiness score over all the subsequences.
(d) Make sure you specify.
  i. base cases and their values
  ii. where the final answer can be found
(e) What is the complexity of your solution.

3. (a) happiness-score[i] has the happiness score of each increasing subsequence

(b)

happiness_score[i] = happiness_score[j] + [array[i] * (len(happiness_score[j])+1)]

max_value = sum(happiness_score[i]) if max_value < sum(happiness_score[i]) else max_value

(c)

```
max_value = 0
happiness_score = [0 for i in range(len(array))]

for i in range (len (array)):
    happiness_score [i] = [array[i]]

for i in range (1, len(array)):
    for j in range(i):
        if array[i] > array[j] and
        sum( happiness_score[i] ) < sum( happiness_score[j])
                                        + array [i]:
            happiness_score [i] = happiness_score [j] +
                [array [i] * (len( happiness_score [j])+1)]
            max_value = sum (happiness_score[i]) if
                max_value < sum( happiness_score[i])
                    else max_value
```

(d) (i)

```
max_value = 0
happiness_score = [0 for i in range(len(array))]
~~[illegible]~~
for i in range(len(array)):
    happiness_score[i] = [array[i]]
```

(ii) Final answer will be in max_value

(e) Time Complexity - $O(n^2)$

Ques 4. You are given $m \times n$ binary matrix $q \in \{0,1\}^{m \times n}$. Each cell either contains a "0" or a "1". Give an efficient algorithm that takes the binary matrix q, and return the largest side length of a square that only contain 1's. You are not required to give the optimal solution.

(a) Define (in plain English) subproblems to be solved.

(b) Write a recurrence relation for the subproblems.

(c) Using the recurrence formula in part b, write pseudocode using iteration to compute the largest side length of a square that only contain 1's to meet the objective.

(d) Make sure you specify.

i. base cases and their values

ii. where the final answer can be found

(e) What is the complexity of your solution?

Ans 4. (a) val[i][j] has the side length of the square which has only 1's

(b) val[i][j] = min(val[i-1][j], val[i-1][j-1], val[i][j-1])+1

max_side_length = max(max_side_length, val[i][j])

(c) val = [[0 for _ in range(len(g[0])+1) for _ in range(len(g) + 1)]

max_side_length = 0

```
for i in range(1, len(g)+1):
    for j in range(1, len(g[0])+1):
        if int(g[i-1][j-1]) == 1:
            val[i][j] = min(val[i-1][j],
                val[i-1][j-1], val[i][j-1])+1
            max_side_length = max(max_side_length,
                val[i][j])
```

(d) (i) val = [[0 for _ in range(len(g[0])+1) for _ in range(len(g) + 1)]

max_side_length = 0

(ii) max_side_length will have the final answer

(e) Time complexity - $O(n^2)$