Question 1 *Crystal Ball Investing*

You have somehow acquired a crystal ball that tells you the price of a particular stock at the start of each day for the next n days – the stock will be worth $A[i]$ cents on day i, with $A[i]$ always being an integer. Trying to make the most of this, you manage to get approved to trade that stock, but with a few restrictions:

[A] You only have the option of trading at the start of the day, when your crystal ball is accurate. That is, you can either buy, sell, or do nothing at the price reported by the ball

[B] You can only have up to a single unit of this stock, and can't short it (i.e. have a negative amount of the stock by selling when you don't own any)

[C] You start with zero units of the stock, and must end with zero units as well – you must sell everything on the nth day

With these constraints, provide an algorithm in pseudo-code that calculates the maximum profit you can make over the n days, that runs in $O(n)$ time?
Answer:

```
Function find-max-profit($A$, $n$):
    $i \leftarrow 0$
    $profit \leftarrow 0$
    while $i < n$ do
        $lmin \leftarrow$ find-next-min($A$, $i$, $n$)
        $lmax \leftarrow$ find-next-max($A$, $lmin + 1$, $n$)
        if $lmin = n \lor lmax = n$ then
            return $profit$
        else
            $profit \leftarrow profit + A[lmax] - A[lmin]$
            $i \leftarrow lmax + 1$
    return $profit$

Function find-next-max($A$, $i$, $n$):
    while $i < n$ do
        if $(i = 0 \lor A[i-1] < A[i]) \land (i = n-1 \lor A[i] > A[i+1])$ then
            return $i$
        $i \leftarrow i + 1$
    return $i$

Function find-next-min($A$, $i$, $n$):
    while $i < n$ do
        if $(i = 0 \lor A[i-1] > A[i]) \land (i = n-1 \lor A[i] < A[i+1])$ then
            return $i$
        $i \leftarrow i + 1$
    return $i$
```

Find-max-profit returns the maximum profit you can make over the n days. Find-next-max returns the next local/end-point maximum in A starting from i inclusively. Find-next-min returns the next local/end-point minimum in A starting from i inclusively.

The algorithm works at buying the stock at the first local/end-point minimum price, then selling at the next local maximum price, buying again at the next local minimum etc... Repeat the process until the last local/end-point maximum.

Note that the first day and the last day is either a local/end-point minimum or local/end-point maximum. We can partition these n days into k intervals. Denote m_k as the day of kth local minimum stock price and M_k as the day of kth local maximum stock price. Partition is $[m_1/M_0, M_1]...[M_{x-1}, M_x][M_x, M_{x+1}]...[M_{k-1},\ m_k/M_k]$, these partitions are inclusive, and notice that between every interval $[M_x, M_{x+1}]$, there exist an m_{x+1}. Denote these interval as $I_1, I_2, ..., I_k$, and denote profit in the kth interval from our algorithm as $p_k = A[M_k] - A[m_k]$.

Any valid transaction plan can be written as a list of tuples in chronological order, ie $[(B_1, S_1), (B_2, S_2)....(B_h, S_h)]$ for a transaction plan of h transactions, where B and S is the day of buying and selling the stock respectively. Our algorithm produce a transaction plan $[(m_1, M_1), (m_2, M_2)....(m_k, M_k)]$

Lemma 0.1. *For any valid transaction plan, sum of profit earned by all transactions within interval k must be less than or equal to p_k. That is,*

$$\sum_{[B,S]\in I_k} (A[S] - A[B]) \le p_k = A[M_k] - A[m_k] \tag{0.1}$$

Lemma 0.2. *For a transaction (B, S), where B and S occurs at different interval I_n and I_k respectively, it can be broken down into $k - n + 1$ intervals where the sum of those profits has the same profit as (B, S). In addition, the buy and sells fall on the same interval. In particular, it can be broken down as*

$$(B, S) = [(B, M_n), (M_n, M_{n+1}), (M_{n+1}, M_{n+2})..., (M_{k-1}, S)] \tag{0.2}$$

For any valid transaction plan, we write it as the list of tuple shown above. For any (B, S) that spans two interval, break it down by lemma 0.2. Now we contain a list where all transaction tuples don't cross intervals. We than replace all (B, S) within the same interval I_k with (m_k, M_k). By lemma 0.1, we know this new transaction plan must have profit $\ge$ the original plan. Repeat this argument for all intervals, and discard all transactions occurred in the last interval if the last interval is $[M_{k-1},\ m_k]$ (all transaction here result in a loss), then every replacement will result in higher or same profit, and we will end up with the same transaction plan produced by our algorithm. By the exchange argument, our profit is maximized.

For every element, the algorithm check whether it is a local minimum/maximum, which takes $O(1)$. Every element is only checked once, so checking all elements is $O(n)$. The cost of calculating total profit is less than checking all elements, so the algorithm is $O(n)$

Question 2 *Plane*

There are n planes at an airport waiting to take off. For safety reasons, only one aircraft is allowed to take off at any given time and it takes one minute for each aircraft to take off.

For each aircraft i, there is a time limit t_i which is a non-negative integer number of minutes. There is no fee if the aircraft takes off at or before t_i, but there is a c_i dollar delaying fee for every minute after.

Design an algorithm in plain English that calculates a schedule of when planes take off that minimises the total cost in $O(n \lg n)$ time. If there are multiple solutions which have the same cost, your algorithm may output any of them.
Answer:

We represent the time schedule as an array where the array index represents time in minutes. Notice that since all loose schedule will always preform worse than or equal to the compact version of that schedule, our algorithm shall only consider a compact schedule. (loose schedule means there exist a time slot where there isn't any planes taking off, and a compact schedule is a schedule where a plane will start to take immediately after the previous plane has take off)

Group flights that have the same starting time t_i together, then sort this list by descending delaying fee using merge sort or quick sort, this creates a group of list where each list contains all flight that has the same t_i but sorted in descending c_i, the total cost of this operation will be is O($nlg(n)$).

If a solution is optimal, for all flights that have the same time limit, the flights must be ordered in descending cost. Since if a solution has flights that have the same starting time but isn't in a descending delaying fee order, we can always swap to make it ordered in descending order without worsening the solution.

Suppose we now have l lists, where the flight in those lists must be in that particular order in the optimal flight schedule. We can merge these l lists into the flight schedule to produce an optimal schedule.

To merge these lists, we start from the first time slot. To decide which flight goes into time slot t, we calculate the minimum cost of that list if it's flight isn't chosen. For example, when deciding each flight takes off at $t = 0$,if list [5,4,2] is $t_i = 0$ (the list elements represents the delaying fee planes), then the minimum cost is occurs when 5 is at $t = 1$, 4 is at $t = 2$, and 2 is at $t = 3$, which is $1 \times 5 + 2 \times 4 + 2 \times 3 = 19$. If list [3, 2, 1] has $t_i = 2$, then the minimum cost occurs when 3 is at $t = 1$, 2 is at $t = 2$ and 1 is at $t = 3$, which is $1 \times 1 = 1$. Pick the list which has the largest minimum cost, then remove the flight in that list, then repeat the process for all other time slots.

Question 3 *Colour a Tree*

You are given a rooted tree with n nodes, and each node i has an integer weight a_i. This tree is given to you as an array of node weights W and an array of parents P. I.e. the ith node has weight $W[i]$ and parent $P[i]$.

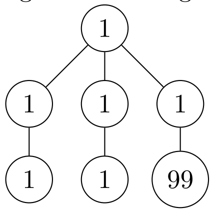
We want to colour all the nodes of the tree as follows:

[A] The root node must be coloured first.

[B] For all other nodes, its parent node must be coloured before it is coloured.

However, colouring nodes has a cost: the kth node that is coloured will cost k times its weight. For example, the cost of colouring the root node r is $1 \times a_r$ since it must be the first node that gets coloured.

3.1 A basic Greedy approach is to focus on the current node, and colour all its children in descending order of weight. Provide a counter example to this approach.



answer:
coloring cost by basic greedy = 420.
A counter example is to prioritize node with weight 99 first cost = $1+2+3\times 99+4+5+6+7 = 322$

3.2 Design an algorithm in plain English to find the minimum total cost of colouring the whole tree in time $O(n^3)$.

3.3 Design an algorithm in plain English that solves the problem in time $O(n \lg n)$. If you are able to correctly solve this subproblem, you do not need to provide a separate solution to Question 3.2.
Answer:

We sort with a array numbered [1, 2, 3,..., n], where the values is the node number. We then sort this array based on the node's weight in descending order, that is, if m precedes n but $W[m] < W[n]$, then swap m and n. We now obtain an array S, where $W[S[i]] \geq W[S[i+1]]$. This is O($nlog(n)$).

Starting from the node with a highest weight, backtrack using the array P, and stop until we reach a parent node we already colored (for the node with highest weight, it must backtrack all the way to the root, as nothing has been colored). Then calculate the coloring cost of this path, and discard all the nodes that have been colored for this path in S. If we have more nodes that have the same weight, then the path that costs more will be colored first. Repeat the process until S is empty, which means all nodes has been colored. This operation is $O(n)$ as each vertex and edge is only traversed once.

Question 4 *Max Grid*

You are given a $n \times n$ grid where each cell has a positive number a_{ij} in it. Your goal is to select a subset of cells with the maximum sum, given that adjacent cells cannot be selected at the same time.

4.1 Provide the optimal solution for the following 4×4 example.

5	3	4	9
4	10	3	6
1	3	7	4
6	2	9	1

Answer: 5+10+9(r1, c4)+4(r3, c4)+9(r4, c3)+6(r4, c1)=43

4.2 Design an efficient algorithm in plain English that achieves the goal. Answer: We can model is problem with a rooted tree. The root node has a weight of 0. This tree will have 16 subtrees, where each root of the subtree is the values of the $n \times n$ grid. The child of each tree are the values it can choose. Then the optimal solution will be a path from root to an edge where the sum of those nodes of the path is maximized. This is clearly not an efficient algorithm (this is simply brute force), but i can't think of a better solution.