Homework-7

(1) Reading Assignment

(2) To maximize profit, the investor will have to sell the share on the day where the price is maximum. (and if the minimum price is on the day before the day of maximum price, then it will be the most favourable condition for the investor to buy on that day.)

Let x_j be the maximum price the investor can get on the day j.

To sell the stock on day j, the investor should have it on day $j-1$. —— Case(1)

If the investor did not have the stock on day $j-1$ then x_j becomes 0. —— Case(2)

So x_j is the total of the return investor can get on day $(j-1)$ and the difference in the prices of both the days.

$$x_j = x_{j-1} + p(j) - p(j-1)$$

Hence, final x_j is the maximum value of case(1) and case(2). And this max function for n values take linear time $O(n)$.

(3) Let us consider that a copy of file is at server j.

So let $OPT(j)$ be the optimal (minimum) cost. (for servers $1, 2, 3, \ldots j$).

In this list, let us consider the optimal solution is at position i. So the optimal cost here will be $OPT(i)$ [This is for servers $1, 2, 3, \ldots i$

Hence optimal cost for server $i+1, i+2, \ldots j$ will be ${}^{j-1}C_2$. The cost at server j will be c_j.

$$\therefore OPT(j) = c_j + \min_{0 \le i < j} \left(OPT(i) + {}^{j-1}C_2 \right)$$

The total running time here is $O(n^2)$

(4) The sequence in which the ranking officer calls his subordinates will be followed recursively by every node. Every node should call its child node in the decreasing order of time it takes for their child nodes to receive message.
Let A be our tree and A' be the subtree (every possible subtree). Let n(A') be the number of rounds it takes for everyone in subtree A' to call. So we can generalize it to: $n(A') = \min_{j} (j + n(A_j))$
The time complexity for n steps with i log n height of tree becomes O(n log n).

(5) To maximize grades, let M[i, h] be the maximum grade that can be achieved for this particular subproblem (ith course, h hours)
In optimal solution, lets say the student spends k hours on ith course (h being the max. hours for getting max grades), then → grade obtained by working for k hours

$$M[i, h] = \max \text{ of } (f_i(k) + M[i-1, h-k])$$
$$\text{from } 0 \le k \le h$$

Time complexity: $O(nH^2)$

(6) To maximize product, let N[i, p] be the rope of length R units, divided into i parts, and the max. product of each unit be p. Our goal is to maximize p. So let this be the subproblem for rope with length n-units. In optimal solution, the max. product can be found by dividing the length i into 1, 2, 3, ... $\frac{i+1}{2}$ (if the length is odd, else i/2 only) $(i-1)$ parts. (Since all subproblems are going to be integers, dividing into i parts will give each subpart of length 1 unit, and product will remain 1. So for every part we check the max product possible with what number of subparts by: $N[i, p] = \max N[i-k, p-s]$

Here, k is the no. of subparts for one particular iteration, and s is the product of values for that particular iteration.
Running time for n-iterations is $O(n^2)$

(7) Since the currency notes are arranged in a fixed but arbitrary sequence, the order can be as follows:

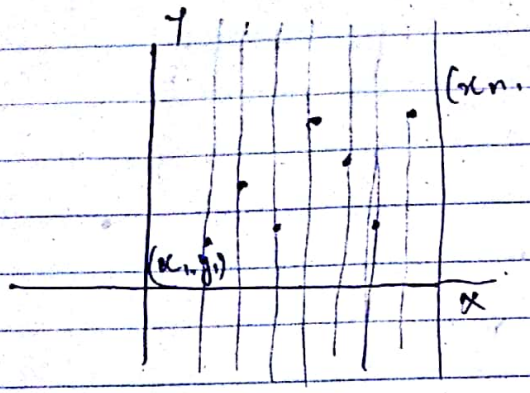
10 5 1 10 5 1 . . . .
1 5 10 1 5 10 . . . .
5 10 1 5 10 1 . . . .
10 $ 5 10 $ 5 . . . . and so on.

Assuming Alice takes the first turn.
If there is $10 on any side, Alice must pick that note
If there is $1 or $5 note, then let the Arrangement be as follows:

A B . . . C D

If A and D are not $10 notes, then compare B with D and C with A.
If AC forms a pair with smaller numbers then pick D, else pick A.
If both A and D are $10, then pick the one whose next element after removing is bigger. So here even if Bob picks $10 in his turn, the bigger number remains open for Alice to pick in her next turn.
This way, Alice will always get a bigger value, or in some cases, Alice might get a smaller value keeping the chance of getting bigger value in her next turn and making Bob get smaller values.
The method gets easier, as Alice knows that Bob uses greedy strategy and successful.

(8)



Keep going right:
$y_i < y_j < y_k < y_l \cdots < y_n$

```
for (x-coordinate = 1 to n)
    for (y-coordinate = 1 to n)
        if yi < yi+1
            pick yi+1 as next co-ordinate with its respective x-coordinate
        else
            go to check for yi+2
```

[If $y_n > y_i$, then there is no path from (x_1, y_1) to (x_n, y_n) that picks points only on the right of y_1 to y_n.]

The same loops are to be followed for (x_n, y_n) to (x_1, y_1), just the minor condition change is:

```
if yn > yn-1
    pick yn-1 as next co-ordinate with its respective yx coordinate
else
    check for yn-2
```

This condition goes for all the values from n to 1.

Time complexity : $O(n^2) + O(n^2)$

$= \boxed{O(n^2)}$