Problem 1

Q1

There are two conditions that Fishy will *always* be sad:

- $s < f$, so that the life time of the fishes on the stand can't withhold that long to satisfy the amount required.
- $t_i <= f$, so that Fishy doesn't have enough time to clean enough fishes (the equal sign is due to the fact that each fish needs 1 second to cleanup).

Q2

Since we know that $f = 1$ means only 1 fish is required on the demanded time-points, and it takes 1 second to clean a fish, so the maximum coverage of a fish in the time span, *is* the minimum amount of fish that Fishy has to clean.

We scan through each time-point to search for the one that's appointed in t_i.

```
$counter \leftarrow 0$                                        ▷ Total fishes required.
$timeRemained \leftarrow 0$                 ▷ Remaining time before the fish is bought.
for $i \leftarrow 1, n$ do
    if $t_i \in$ the list then
        if $timeRemained = 0$ then
            $timeRemained \leftarrow s$
            $counter \leftarrow 1$
        end if
    else if $timeRemained > 0$ then
        $timeRemained \leftarrow timeRemained - 1$
    end if
end for
```

Let the minimum fishes required is c, to prove the algorithm works, we have to assume that we can find $c' < c$.

Q3

In previous question, we've pinned f to 1, now f can be greater then 1, meaning that we have to clean the fishes earlier, in order to reach the number required.

Meaning that the *timeRemained* in previous algorithm will have to decrease earlier, in order to compensate the multiple cleansing processes.

```
$counter \leftarrow 0$                              ▷ Total fishes required.
$timeRemained \leftarrow 0$          ▷ Remaining time before the fish is bought.
for $i \leftarrow 1, n$ do
    if $t_i \in$ the list then
        if $timeRemained = 0$ then
            $timeRemained \leftarrow s - f$
            $counter \leftarrow 1$
        end if
    else if $timeRemained > 0$ then
        $timeRemained \leftarrow timeRemained - 1$
    end if
end for
```

Problem 2

Q1

The items we have to buy range from 1 to N. Assume the total price needed is P

$$P = \sum_{i=1}^{N} p_i, \tag{1}$$

total price needed is directly equal to total price needed since the player can earn 1 dollar every second. So

$$T = P = \sum_{i=1}^{N} p_i \tag{2}$$

The goal is to calculate the average influence, which means the influence along the time span 1 to T, where T is the total time required to buy every equipments and E_t is the equipments that the player has at t.

$$\bar{I} = \frac{\sum_{t=1}^{T} \sum_{i \in E_t} b_i}{T} \tag{3}$$

Due to the fact that we are buying the equipments in sequential, we can expand the summation equation into

$$\bar{I} = 0 \times \frac{p_1}{T} + b_1 \times \frac{p_2}{T} + (b_1 + b_2) \times \frac{p_3}{T} + \cdots + \sum_{i=1}^{N-1} b_i \times \frac{p_N}{T}, \tag{4}$$

at last, simplify it as

$$\bar{I} = \frac{\sum_{m=2}^{N} p_m \sum_{n=1}^{m-1} b_n}{T} = \frac{\sum_{m=2}^{N} p_m \sum_{n=1}^{m-1} b_n}{\sum_{m=1}^{N} p_m}. \tag{5}$$

We can see that the last equipment doesn't have the time to show its power, aka, contribute to the average influence.

Q2

From previous question, we know that a sequential buying strategy leads to an average influence of

$$\begin{aligned}\bar{I} = &\frac{\sum_{m=2}^{i} p_m \sum_{n=1}^{m-1} b_n}{T} \\ &+ \frac{p_{i+1}(b_i + \sum_{n=1}^{i-1} b_n) + p_{i+2}(b_{i+1} + b_i + \sum_{n=1}^{i-1} b_n)}{T} \\ &+ \frac{\sum_{m=i+3}^{N} p_m \sum_{n=1}^{m-1} b_n}{T}.\end{aligned} \tag{6}$$

When we try to swap the item i and $i+1$, we have to rewrite the equation as

$$\begin{aligned}\bar{I}' = &\frac{\sum_{m=2}^{i} p_m \sum_{n=1}^{m-1} b_n}{T} \\ &+ \frac{p_{i+2}(b_{i+1} + \sum_{n=1}^{i-1} b_n) + p_{i+1}(b_i + b_{i+1} + \sum_{n=1}^{i-1} b_n)}{T} \\ &+ \frac{\sum_{m=i+3}^{N} p_m \sum_{n=1}^{m-1} b_n}{T}.\end{aligned} \tag{7}$$

The difference between $\bar{I}$ and $\bar{I}'$ can now write as

$$\begin{aligned}\Delta I = &\frac{p_{i+1}(b_i + \sum_{n=1}^{i-1} b_n) + p_{i+2}(b_{i+1} + b_i + \sum_{n=1}^{i-1} b_n)}{T} \\ &- \frac{p_{i+2}(b_{i+1} + \sum_{n=1}^{i-1} b_n) + p_{i+1}(b_i + b_{i+1} + \sum_{n=1}^{i-1} b_n)}{T},\end{aligned} \tag{8}$$

by distribute p_{i+1}, p_{i+2} and eliminate the same elements, we can further simplify ΔI as

$$\Delta I = \frac{p_{i+2}b_i - p_{i+1}b_{i+1}}{T} \tag{9}$$

Q3

If A *isn not* an optimal solution, it means that whatever A is, after it performs a swap, the average influence is higher than current one. Assume the swap is performed at k and $k+1$, this means that $f(S_{swap(k,k+1)}) > f(S)$.

We first focus on equipment k and $k+1$, which means the buying sequence from 1 to $k-1$ and $k+2$ to N are both the optimal solution. And we also know that after a swap, A will no longer be the optimal solution, which means there exists a sequence S, such that $S_k = A_i$, $S_{k+1} = A_j$, where $f(S_{swap(k,k+1)}) > f(S)$. This contradicts to the original assumption that the sequence doesn't exist, therefore, A must be an optimal solution.

Q4

Since we can only compare the nearest two elements, merge sort is the only option for us that can complete the task in $O(nlgn)$.

We first directly divide the equipment array into halves, and then combine them, during the combination, we compare the elements themselves, and then merge it. Due to the fact that after each combination, the elements will become consecutive, the ΔI can use here directly.

```
function MERGESORT($A$)
    $left \leftarrow [\ ]$
    $right \leftarrow [\ ]$
    if $A.length \leq 1$ then
        return $A$
    else
        $middle \leftarrow \frac{A.length}{2}$
        for all $x \in A[1] \cdots A[middle-1]$ do
            add $x$ to $left$
        end for
        for all $x \in A[middle+1] \cdots A[end]$ do
            add $x$ to $right$
        end for
        $left \leftarrow$ MERGESORT($left$)
        $right \leftarrow$ MERGESORT($right$)
        if COMPARE($left[end]$, $right[1]$) then
            append $right$ to $left$
            return $left$
        end if
        $result \leftarrow$ MERGE($left$, $right$)
        return $result$
    end if
end function

function MERGE($left$, $right$)
    $result \leftarrow [\,]$
    while $left.length > 0 \wedge right.length > 0$ do
        if COMPARE($left[1]$, $right[1]$) then
            $append\ left[1]\ to\ result$
            $left \leftarrow left[2 \cdots end]$
        else
            $append\ right[1]\ to\ result$
            $right \leftarrow right[2 \cdots end]$
        end if
    end while
    if $left.length > 0$ then
        $append\ left\ to\ result$
    end if
    if $right.length > 0$ then
        $append\ right\ to\ result$
    end if
    return $result$
end function
```

Note: The algorithm is based on the pseudocode enlisted in the rosettacode.org

The COMPARE function call calculates the difference of influence using the result from question 2. Which only requires the elements that we'd like to swap. Since we know the indices, we can easily locate the value p_i and b_i in $O(1)$.

Using the recurrence relationship, we can know that a single call of the quick sort involves $O(n)$ work and 2 recursive calls on two lists of size $n/2$ (on average), therefore,

$$T(n) = 2T(\frac{n}{2}) + O(n), \tag{10}$$

through the master theorem, we know that $T(n) = O(nlgn)$.

Q5

From the result in question 2, we can see that T is always the same through out the calculation, due to the fact that we only care about the relative relationship, aka larger or smaller, no division is needed here.

```
struct equip {
    int price;
    int influence;
}

// n is the size of the array
void merge_sort(struct equip* A, int n) {
    if(n < 2)
        return;
    int m = n/2;
    merge_sort(A, m);
    merge_sort(A+m, n-m);
    merge(A, n, m);
}

void merge(struct equip* A, int n, int m) {
    int i, j, k;
    int *x = malloc(n * sizeof(struct equip));
    for(i = 0, j = m, k = 0; k < n; k++) {
        if(j == n)
            x[k] = A[i++];
        else if(i == m)
            x[k] = A[j++];
        else if(compare(a[j] < A[i])
            x[k] = A[j++];
        else
            x[k] = A[i++];
    }
    for(i = 0; i < n; i++) {
        A[i] = x[i];
    }
    free(x);
}

int compare(struct equip* A, int a, int b) {
    int pi1 = (A+a+1)->price, pi2 = (A+b+1)->price;
    int bi = (A+a)->influence, bi1 = (A+b)->influence;
    // Reverse the order since this is compare for '<'
    return (pi1*bi1 - pi2*bi);
}
```

Problem 3

Q1

The problem states that Paul is the first one to make a move, therefore, he will definitely get the largest card on the table.

When it's John's turn, assume that both of them are familiar with this game, aka, they go for the best strategy (play optimally). John must chooses the largest card on the table as well. However, since the largest one is picked, he can only targeted the second largest one.

This continues, until all the cards are taken, which leads to the odd-even sum on the respective party.

Q2

When the number of cards in each pile is larger than 2 (and even), normally, this means that Paul will take the upper pile of the piles while John will take the lower portion of the piles.

The following scenario assume that John would like to take the other portion of the cards from Paul, which is marked as red here.

$$\begin{array}{c} a_{1,1} \cdots {\color{red}a_{1,\frac{n}{2}}}\ a_{1,\frac{n}{2}+1} \cdots a_{1,n_1} \\ a_{2,1} \cdots a_{2,\frac{2}{2}}\ a_{2,\frac{n}{2}+1} \cdots a_{2,n_2} \end{array}$$

During the first round, assume that there are cards that is more attractive than $a_{1,\frac{n}{2}}$ to Paul, so he takes $a_{2,1}$ first.

$$\begin{array}{c} a_{1,1}\ a_{1,2} \cdots {\color{red}a_{1,\frac{n}{2}}}\ a_{1,\frac{n}{2}+1} \cdots a_{1,n_1} \\ a_{2,2}\ a_{2,3} \cdots a_{2,\frac{2}{2}}\ a_{2,\frac{n}{2}+1} \cdots a_{2,n_2} \end{array}$$

John can now make his move and begin to take the card on the first pile.

$$\begin{array}{c} a_{1,1}\ a_{1,2} \cdots {\color{red}a_{1,\frac{n}{2}}}\ a_{1,\frac{n}{2}+1} \cdots a_{1,n-2}\ a_{1,n_1-1} \\ a_{2,2}\ a_{2,3} \cdots a_{2,\frac{2}{2}}\ a_{2,\frac{n}{2}+1} \cdots a_{2,n_2-1}\ a_{2,n_2} \end{array}$$

If Paul wants to block John's attack, he can simply choose to take the card $a_{1,1}$, after $\frac{n}{2}$ rounds, when they all focus on that pile of cards,

$$a_{1,\frac{n}{2}-1} \; a_{1,\frac{n}{2}} \; a_{1,\frac{n}{2}+1} a_{1,\frac{n}{2}+2}$$

John will still be 1 card away from his target. Leading to the result that *who ever wants to guard their portion of cards, they must can.* Under the optimal condition, both of them will get at least $\sum_{i=1}^{N}\sum_{j=1}^{\frac{n_i}{2}} a_{ij}$ and $\sum_{i=1}^{N}\sum_{j=\frac{n_i}{2}+1}^{n_i} a_{ij}$ as the problem stated.

Q3

When every single pile of the cards are odd in their amounts. We can view the card piles as

$$\begin{array}{c} a_{1,1} \cdots a_{1,\frac{n-1}{2}} \; a_{1,\frac{n-1}{2}+1} \; a_{1,\frac{n+1}{2}} \cdots a_{1,n_1} \\ a_{2,1} \cdots a_{2,\frac{n-1}{2}} \; a_{2,\frac{n-1}{2}+1} \; a_{2,\frac{n+1}{2}} \cdots a_{2,n_2}, \end{array}$$

the marked cards are the *center* pieces, whereas card counts below and above them are all the same: $\frac{N-1}{2}$.

The center pieces are the final decision scores that Paul and John are fighting for. However, Paul is the one who gets to pick the cards first, he certainly will target for the maximum *center piece.* Therefore, we can degrade this scenario back to question 1, whereas we only consider each *pile* of cards has only 1 card.

When we consider the center pieces, than the score $\sum_{k \ is \ odd} a_{[k]\frac{n_i+1}{2}}$ and $\sum_{k \ is \ even} a_{[k]\frac{n_i+1}{2}}$ is reasonable here. Since Paul and John can both guard for their respective portion of cards, aka upper pile or lower pile, their total scores are respectively

$$\sum_{i=1}^{N}\sum_{j=1}^{\frac{n_i-1}{2}} a_{ij} + \sum_{k \ is \ odd} a_{[k]\frac{n_i+1}{2}} \tag{11}$$

and

$$\sum_{i=1}^{N}\sum_{j=\frac{n_i+2}{2}}^{n_i} a_{ij} + \sum_{k \ is \ even} a_{[k]\frac{n_i+1}{2}}, \tag{12}$$

since we just have to add back their left, right portion of the scores.

Q4

We now no longer have the constrain on the number of cards for each pile, therefore, we simply just divide the piles of cards into odd and even cases, and add up the results. Assuming the odd piles of cards denotes as N_{odd} while the even piles denotes as N_{even},

$$N_{odd} = \{i : n_i \in odd\} \tag{13}$$
$$N_{even} = \{i : n_i \in even\}. \tag{14}$$

For Paul,

$$\left(\sum_{i=1}^{N_{odd}} \sum_{j=1}^{\frac{n_i-1}{2}} a_{ij} + \sum_{k\ is\ odd\ in\ N_{odd}} a_{[k]\frac{n_i+1}{2}}\right) + \sum_{i=1}^{N_{even}} \sum_{j=1}^{\frac{n_i}{2}} a_{ij}. \tag{15}$$

For John,

$$\left(\sum_{i=1}^{N_{odd}} \sum_{j=\frac{n_i+2}{2}}^{n_i} a_{ij} + \sum_{k\ is\ even\ in\ N_{odd}} a_{[k]\frac{n_i+1}{2}}\right) + \sum_{i=1}^{N_{even}} \sum_{j=\frac{n_i}{2}+1}^{n_i} a_{ij}. \tag{16}$$

Note 1: From the hint, we know that as long as we find the least boundary for Paul and John, when they play the game optimally, the boundary limits will be their scores.
Note 2: Taking *k is even in* N_{odd} as an example, it means that in the group of numbers in N_{odd}, such as $1, 5, 7, 11, 27, 33 \cdots$, we takes the even ones: $5, 11, 33 \cdots$