Homework #2

Due Time: 2020/11/10 14:20
Contact TAs: `ada-ta@csie.ntu.edu.tw`

Instructions and Announcements

- There are **four programming problems** and **two hand-written problems**.
- **Programming.** (55 pt + 5 bonus) The judge system is located at `https://ada-judge.csie.ntu.edu.tw`. Please login and submit your code for the programming problems (i.e., those containing "Programming" in the problem title) by the deadline. **Note that if you got more than 55 points in programming part, the score would be reduce to 55 points.** NO LATE SUBMISSION IS ALLOWED.
- **Hand-written.** (50 pt) For other problems (also known as the "hand-written problems"), you should upload your answer to **Gradescope** as demonstrated in class. Please **briefly** explain your solution in this part. NO LATE SUBMISSION IS ALLOWED.
- **Collaboration policy.** Discussions with others are strongly encouraged. However, you should write down your solutions **in your own words**. In addition, for **each and every** problem you have to specify the references (e.g., the Internet URL you consulted with or the people you discussed with) on the first page or comment in code of your solution to that problem. You may get zero point due to the lack of references.

Problem 1 - Tower Defense (Programming) (10 points)

> Don't hold on to the past; it won't help in moving forward.
>
> Rajeev Suri

Problem Description

It's game time again! This time you have to build defense towers to protect your castle from foreign attacks.

The playfield can, again, be seen as a one-dimensional grid containing N cells numbered from 1 to N. Since the terrain of each cell varies, some of them may be well suited to build a defense tower on, while at the same time, towers built on some other cells can be useless. We denote the *suitability* of the i^{th} cell by s_i. **Note that s_i may be negative.**

For each cell, you can decide whether to build a defense tower on it or not, but the budget you have is only enough to build K towers. Suppose you build t towers on some t cells, where $0 \leq t \leq K$. We denote the cells you choose by a sequence of t indices $1 \leq x_1 < x_2 < \cdots < x_t \leq N$. The defense level of this arrangement is

$$\sum_{i=1}^{t} s_{x_i},$$

which is the sum of the suitability of the chosen cells.

Moreover, it will be harder for your enemy to sneak into your castle if you build the towers closer to each other. More precisely, each valid index i has a bonus defense level of

$$A \cdot \max\{0, B - (x_{i+1} - x_i)\},$$

where both A and B are positive integers. In other words, if you build two towers close enough, you receive a bonus that is negative linear to the distance between these two towers.

Now, your task is to plan out an arrangement wisely to maximize the total defense level.

Input

The first line of the input contains four integers N, K, A and B, indicating the length of the field, the budget, and the constants used in the bonus formula, respectively. The second line of the input contains N space-separated integers $s_1, s_2, \ldots, s_N$, where the i^{th} integer denotes the suitability of the i^{th} cell.

- $1 \leq B \leq N \leq 10^5$
- $1 \leq K \leq 400$
- $1 \leq A \leq 10^9$
- $-10^9 \leq s_i \leq 10^9$

Test Group 0 (0 %)

- Sample Input

Test Group 1 (10 %)

- $N \leq 1000$
- $K \leq 2$

Test Group 2 (30 %)

- $N \leq 1000$
- $K \leq 25$

Test Group 3 (10 %)

- $B = 1$

Test Group 4 (10 %)

- $B = N$

Test Group 5 (20 %)

- $K \leq 25$

Test Group 6 (20 %)

- No additional constraints

Output

Output a single integer representing the maximum total defense level.

Sample Input 1

```
5 3 9 3
4 -1 -11 6 5
```

Sample Output 1

```
37
```

Sample Input 2

```
3 2 10000 1
1 2 7
```

Sample Output 2

```
9
```

Sample Input 3

```
2 2 6 2
10 -30
```

Sample Output 3

```
10
```

Sample Input 4

```
8 4 4 4
2 15 -12 14 9 14 11 -10
```

Sample Output 4

```
84
```

Sample Input 5

```
12 4 6 2
11 13 6 -6 -8 6 -6 14 8 6 1 -10
```

Sample Output 5

```
58
```

Hint

- Try to figure out what the constraints of each test group mean.
- Oops! We're running out of space. Maybe the quote at the beginning can help you.
  - Note that *Runtime Error* may be an indication of *Memory Limit Exceeded*.
- You can solve this problem without using anything too fancy.

Problem 2 - ADA Sequence (Programming) (15 points)

Problem Description

YP and BB love sequences very much. They define a sequence $b_1, b_2, \ldots, b_Y$ of a positive length Y as an ADA sequence if for all $i \in [1, Y]$, the following condition holds:

$$|b_i - b_{Y-i+1}| \leq D,$$

where D is a given parameter.

Now, BB gives YP three N-length sequences $a_1, a_2, \ldots, a_N$; $c_1, c_2, \ldots, c_N$, and $s_1, s_2, \ldots, s_N$. Also, BB will give YP K dollars and decides to play a game with YP!

YP can do the following operations at most 1000 times (with zero times being possible):

- Select an integer $i \in [1, N]$.
- Spend c_i dollars to change a_i into an arbitrary value.

Note that the total amount of money YP spends on the operations should not exceed K.

After finishing the operations, YP needs to choose some indices $b_1, b_2, \ldots, b_M$. BB defines the indices **valid** if the following conditions are satisfied:

- $M > 0$
- $1 \leq b_1 < b_2 < \cdots < b_{M-1} < b_M \leq N$
- $a_{b_1}, a_{b_2}, \ldots, a_{b_M}$ is an ADA sequence.

Also, BB defines the **score** of the ADA sequence as $s_{b_1} + s_{b_2} + \cdots + s_{b_M}$.

Please help YP find out the maximum possible score, and tell YP how to do the operations and select $b_1, b_2, \ldots, b_M$ for forming the ADA sequence.

Input

The first line of the input contains three integers N, K, and D, as defined in the problem description.

The second line of the input contains N integers $a_1, a_2, \ldots, a_N$.

The third line of the input contains N integers $c_1, c_2, \ldots, c_N$.

The fourth line of the input contains N integers $s_1, s_2, \ldots, s_N$.

Test Group 0 (0 %)

- Sample Input

Test Group 1 (10 %)

- $1 \leq N \leq 200$
- $K = 0$
- $D = 0$
- $1 \leq a_i \leq 10^9$
- $1 \leq c_i \leq 600$
- $s_i = 1$

Test Group 2 (30 %)

- $1 \leq N \leq 200$
- $K = 0$
- $D = 0$
- $1 \leq a_i \leq 10^9$
- $1 \leq c_i \leq 600$
- $1 \leq s_i \leq 3$

Test Group 3 (30 %)

- $1 \leq N \leq 200$
- $0 \leq K \leq 600$
- $0 \leq D \leq 10^9$
- $1 \leq a_i \leq 10^9$
- $1 \leq c_i \leq 600$
- $1 \leq s_i \leq 3$

Test Group 4 (30 %)

- $1 \leq N \leq 200$
- $0 \leq K \leq 10^9$
- $0 \leq D \leq 10^9$
- $1 \leq a_i \leq 10^9$
- $1 \leq c_i \leq 10^9$
- $1 \leq s_i \leq 3$

Output

Please output $X + 4$ lines.

The first line contains one integer S denoting the maximum possible score.

The second line contains one integer X denoting the number of operations you want to do.

In the following X lines, the i-th line contains two integers pos_i, val_i. This indicates that in the i-th operation, YP changes a_{pos_i} to val_i.

In the next line, output an integer M denoting the length of the sequence b.

In the next line, output M integers denoting the indices YP chooses.

The output must satisfy the following restrictions:

- $0 \leq X \leq 1000$
- $1 \leq pos_i \leq N$
- $1 \leq val_i \leq 10^9$
- $\sum_{i=1}^{X} c_{pos_i} \leq K$
- $1 \leq M \leq N$
- $1 \leq b_i \leq N$
- $b_i < b_{i+1} \forall i \in [1, M-1]$
- After doing the K operations, $a_{b_1}, a_{b_2}, \ldots, a_{b_M}$ becomes an ADA sequence.
- $s_{b_1} + s_{b_2} + \ldots s_{b_M} = S$ is maximized.

If there are multiple solutions, you can output any of them.

It is guaranteed that there exists a solution satisfying the above restrictions.

Sample Input 1

```
6 0 0
4 2 3 2 3 4
1 1 1 1 1 1
1 1 1 1 1 1
```

Sample Output 1

```
5
0
5
1 3 4 5 6
```

Sample Input 2

```
8 0 0
8 3 1 3 4 4 1 8
1 1 1 1 1 1 1 1
1 3 1 3 1 1 1 1
```

Sample Output 2

```
9
0
5
1 2 3 4 8
```

Sample Input 3

```
6 3 1
4 8 17 1 6 12
1 2 2 3 1 2
1 3 1 2 2 1
```

Sample Output 3

```
9
2
1 12
5 8
5
1 2 4 5 6
```

Sample Input 4

```
10 1258 3
417 118 413 213 116 817 200 154 177 465
765 872 548 874 254 654 966 553 398 698
1 3 2 3 1 3 3 1 1 3
```

Sample Output 4

```
16
3
3 465
9 213
5 200
7
3 4 5 6 7 9 10
```

Hint

1. Solving the test groups in order (from 1 to 3) is helpful for solving Test Group 4.
2. *Longest Common Subsequence* (LCS), *Longest Increaing Subsequence* (LIS), and *Longest Palindromic Subsequence* (LPS) are classic dynamic programming problems.
3. GL & HF (Good Luck and Have Fun).

Problem 3 - Boook Arrangement (Programming) (15 points)

Problem Description

Boook is a librarian, and arranging books to make them look beautiful is his hobby. Boook thinks that a sequence of books is beautiful if the thickness of adjacent books differs by 1. Formally, if the thickness of the books in the sequence are $x_1, x_2, \ldots, x_k$, respectively, then it is beautiful if and only if $|x_i - x_{i+1}| = 1$ for all $i = 1, 2, \ldots, k-1$.

One day, Boook receives some books. More specifically, there are c_i books with the thickness i, for all $i = 1, 2, \ldots, n$. Boook would like to make them beautiful by arranging them into a beautiful sequence. However, he finds that sometimes doing so is impossible, so he wants to insert the least number of additional books with thickness also in the range $[1, n]$ so that he can make these books beautiful. Please help Boook complete the mission.

Input

The first line contains a positive integer n ($2 \leq n \leq 10^5$), which is mentioned in the description.

The second line contains n non-negative integers. The i-th integer c_i ($0 \leq c_i \leq 10^6$) represents the number of books with thickness i Boook initially has. It is guaranteed that at least one c_i is non-zero.

The third contains only one integer $flag$ ($flag \in \{0, 1\}$), indicating whether you should print an beautiful arrangement of the books or not.

It is guaranteed that the size of the output is less than 10^7 bytes.

Output

In the first line of the output, print an integer indicating the least number of additional books Boook needs to insert. Furthermore, if $flag = 1$ in the input, print a beautiful arrangement of the books using the least number of additional books in the second line. If there are multiple ways to arrange the books, you may print any of them.

Test Group 0 (10 %)

- Sample Input

Test Group 1 (10 %)

- $flag = 0$
- $c_i = 0, \forall\ 2 \mid i$, or $c_i = 0, \forall\ 2 \nmid i$

Test Group 2 (10 %)

- $c_i = 0, \forall\ 2 \mid i$, or $c_i = 0, \forall\ 2 \nmid i$

Test Group 3 (10 %)

- $flag = 0$
- $n \leq 4$

Test Group 4 (10 %)

- $n \leq 4$

Test Group 5 (20 %)

- Boook does not need to insert the additional books, that is, the first line of the output is 0

Test Group 6 (20 %)

- $flag = 0$

Test Group 7 (20 %)

- No other constraints

Sample Input 1

```
4
1 1 2 1
1
```

Sample Output 1

```
0
3 4 3 2 1
```

Sample Input 2

```
4
2 4 3 1
1
```

Sample Output 2

```
0
1 2 3 2 1 2 3 4 3 2
```

Sample Input 3

```
5
1 2 0 2 0
1
```

Sample Output 3

```
2
2 1 2 3 4 5 4
```

Sample Input 4

```
7
2 2 4 2 5 3 1
1
```

Sample Output 4

```
4
3 2 3 2 1 2 1 2 3 2 3 4 5 6 5 6 7 6 5 4 5 4 5
```

Problem 4 - Segments (Programming) (15 points + 5 points bonus)

Problem Description

Given N segment sets, each of them is located between $[0, M]$. The i-th segment set can be represented by (L_i, R_i, W_i), indicating that there are W_i segments covering the range $[L_i, R_i]$.

For any chosen subset of the segments and a real number x, we define the function $f(x)$ as the number of chosen segments that cover the position x. Your goal is to pick a subset S containing at most K segments that maximizes

$$V = \min_{0 \le x \le M} f(x).$$

Input

The first line contains three space-separated integers N, M and K, which represent the number of segment sets, the range of the segments, and the number of segments you can select respectively.

Each of the following N lines contains three space-separated integers L_i, R_i, and W_i, indicating the location of the segments and the number of segments in the i-th segment set.

- $0 \le K \le \sum W_i$
- $1 \le N, M \le 2 \times 10^5$
- $0 \le L_i < R_i \le M$
- $1 \le W_i \le 3 \times 10^{13}$

Test Group 0 (0 %)

- Sample Input

Test Group 1 (15 %)

- $K = 2$
- $W_i = 1$

Test Group 2 (25 %)

- $K = N$
- $W_i = 1$

Test Group 3 (25 %)

- $1 \le N, M \le 1000$
- $W_i = 1$

Test Group 4 (25 %)

- $1 \le N, M \le 10^5$

Test Group 5 (10 %)

- No other constraints

Output

In the first line of the output, print an integer indicating the maximum V.

In the second line, print N integers, where the i-th integer represents the number of segments you selected in the i-th segment set. If there are many possible answers, you can print any of them.

Sample Input 1

```
3 9 2
0 4 1
4 9 1
1 8 1
```

Sample Output 1

```
1
1 1 0
```

Sample Input 2

```
3 9 3
0 4 1
4 9 1
0 9 1
```

Sample Output 2

```
2
1 1 1
```

Sample Input 3

```
5 9 3
0 4 1
2 6 1
7 8 1
5 9 1
1 7 1
```

Sample Output 3

```
1
1 1 0 1 0
```

Sample Input 4

```
5 9 40
0 4 30
2 6 18
7 8 99
5 9 70
1 7 12
```

Sample Output 4

```
13
13 1 0 13 12
```

Origin & Bonus (5 points)

In homework 1 bridge, we discovered that a wrong solution is able to pass all test data generated randomly. The cool but **wrong** method is mainly comprised of the following two steps:

1. Rotate every point $P_i \in P$ by an angle randomly.
2. Sort all points $P_i \in P$ according to the X-axis, and find the closest pair of points from all pairs of points if the distance of their positions is less than $Magic$. Formally,

$$\min_{i=1}^{N} \left(\min_{j=i+1}^{\min(j+Magic,N)} (Dis(P_i, P_j)) \right)$$

Now please generate a test data to make the following solution (the source code below) wrong.

Source Code: `https://www.csie.ntu.edu.tw/~b07902133/wa-bridge.cpp`

Hint

This bonus can be reduced to the **Segments** problem.

Problem 5 - Trick-or-Treating (Hand-Written) (30 points)

*In problem 5, please **briefly** explain your solution in text. **Do not** use pseudo code, or you will receive penalty. Note that if you use Greedy or Dynamic Programming in any subproblem of this problem, you should prove their properties (optimal substructure, greedy-choice property).*

Halloween is coming! As a starving student writing ADA homework 2 in the dorm all day long, you decided to go Trick-or-Treating with your friend Ada and ask for some candy from professors at midnight.

There are N professors participating in the event. After knocking on the i-th professor's door, he/she will chat with you by the door for t_i unit time and give you c_i candy.

However, to show respect to the professor, students should not do anything except chatting with the professor during the t_i unit time. Furthermore, the i-th professor needs to rest after r_i unit time from midnight. Suppose any student knocks on the professor's door or chats with the professor during his/her resting time, or visits the same professor more than once. In that case, the professor will be furious and ask the ADA instructor to give the student an F.

You are both great cyclists and can move from one door to another with your bicycle immediately (no time consuming). Please maximize the candies you can receive without getting an F on ADA.

Lastly, there are some assumptions shown as follows, which are used in the following problems.

Assumption 1. *Every professor can only give* 1 *candy, i.e., for all* $1 \leq i \leq N$, $c_i = 1$.

Assumption 2. *The longer a professor chats, the longer he/she needs to rest, i.e., for all* $1 \leq i, j \leq N$, *if* $t_i < t_j$, *then* $r_i < r_j$.

Assumption 3. *The professors' chatting time sequence is strictly increasing, i.e., for all* $i < j$, $t_i < t_j$.

Assumption 4. *The total amount of candy received is smaller than* M, *i.e.,* $\sum_{i=1}^{N} c_i < M$.

Please answer the following problems. **Note that if you answer (3) correctly, you will automatically get the score of (2).**

(1) (3%) Given the following information, please calculate the maximum number of candies you can get.

- $N = 5$
- $t = [2, 2, 3, 6, 10]$
- $r = [3, 4, 15, 16, 10]$
- $c = [8, 5, 1, 9, 8]$

23. Visits the 1st, 2nd, 3rd, and 4th professor.

(2) (5%) Please design an algorithm with time complexity $O(N)$ under all **Assumption 1, Assumption 2**, and **Assumption 3** to calculate the maximum candy you can get. Also, you need to explain why your algorithm meets the time complexity requirement.

Consider visit the professor with smaller id first, starts from the 1st professor. Maintain the total visit time of the previous professors T which is 0 initially and the total amount of obtained candy D which is 0 initially. When we encounter a new professor i, if $T + t_i \leq r_i$, update T to $T + t_i$ and C to $C + 1$. The ultimate value of C is the maximum amount of candy.

The size of sequence is N and the update time is $O(1)$ for each professor, hence the total time complexity is $O(N)$.

(3) (6%) Please design an algorithm with time complexity $O(N \log N)$ under **Assumption 1** to calculate the maximum candy you can get. Briefly explain the correctness and why your algorithm meets the time complexity requirement.

Generate a permutation p from 1 to N such that $\forall i < N, r_{p_i} \leq r_{p_{i+1}}$. Starts iterate through p from p_1, and maintain a visiting set S which is empty initially and the sum of visit time T which is 0 initially. When we encounter a new professor p_i, update S_i to $S_{i-1} \cup \{p_i\}$ and T to $T + t_{p_i}$. After that, if $T > r_{p_i}$ update S_i to $S_i \setminus \{\arg\max_{k \in S_i} t_k\}$ and T to $T - max_{k \in S_i} t_k$. The maximum amount of candy is $|S_N|$ eventually.

To generate p, we could sort by the value of r, the time complexity is $O(N \log N)$. Iterate through p cost $O(N)$ time, while getting the argument max of a set in each round cost $O(\log N)$ utilizing heap or self-balancing binary tree. Checking and updating T cost $O(1)$ in each round. Hence, the totally complexity is $O(N)(O(\log N) + O(1)) + O(N \log N) = O(N \log N)$

Correctness:

Note that there are multiple approaches to prove these properties. Any valid proof will be considered correct.

Property 1. *If a set S of professor could all be visited, in all of the legal visit order, at least one of it has the non-decreasing r_i order.*

Proof. For any optimal visit order $a = [a_1, a_2, ..., a_{|S|}]$, if exist $r_{a_i} > r_{a_{i+1}}$, we could always swap a_i and a_{i+1}. Since the visit time before a_i and after a_{i+1} would not change, only a_i and a_{i+1} need to be consider. It's trivial that a_{i+1} could still be visited after swap. As for a_i, observe the constrain of the original sequence that $\sum_{k=1}^{i-1} t_{a_k} + t_{a_i} + t_{a_{i+1}} \leq r_{a_{i+1}}$ and $r_{a_i} > r_{a_{i+1}}$, $\sum_{k=1}^{i-1} t_{a_k} + t_{a_i} + t_{a_{i+1}} < r_{a_i}$, which implies a_i could still be visited after swap. By swapping all such pairs, a could be rearranged into non-decreasing order. □

By property 1, if a set of professor could not be visited after sorted by r_i, it could not be visited completely, while the opposite direction is trivial. We could maintain a set of professor and added professor in by non-decreasing order of r_i. For convenience, suppose the original sequence is already sorted in non-decreasing r_i order.

Lets define some additional notation.

OPT_i: optimal strategy solving problem of first i professors which use **minimum time** to get the maximum candy.

SOL_i: the strategy we proposed solving problem of i professors.

T_{strategy}: spending time of a strategy.

C_{strategy}: obtained candy amount of a strategy.

S_{strategy}: visit set of a strategy.

Proof by induction that using the algorithm, $\forall i > 0, C_{SOL_i} = C_{OPT_i} \wedge T_{SOL_i} = T_{OPT_i}$.

Base case: $C_{SOL_1} = C_{OPT_1} \wedge T_{SOL_1} = T_{OPT_1}$
By our algorithm, 1 would be add into S_{SOL_1} if $t_1 \leq r_1$. It's trivial that this is the optimal solution. $SOL_1 = OPT_1 \Rightarrow C_{SOL_1} = C_{OPT_1} \wedge T_{SOL_1} = T_{OPT_1}$.

Induction Step: Suppose $C_{SOL_i} = C_{OPT_i} \wedge T_{SOL_i} = T_{OPT_i}$ when $k < i$. SOL_i could be in one of the following two cases by the algorithm.

Case 1: $C_{SOL_i} = C_{SOL_{i-1}} + 1$

Base on our algorithm, i is added directly to the end of visit sequence, $T_{SOL_{i-1}} + t_i \leq r_i$ holds. By $T_{OPT_{i-1}} = T_{SOL_{i-1}}$, $T_{OPT_{i-1}} + t_i \leq r_i$, and thus $C_{OPT_i} \geq C_{SOL_i}$. Suppose $C_{OPT_i} > C_{SOL_i} = C_{SOL_{i-1}} + 1 = C_{OPT_{i-1}} + 1$, by removing i from S_{OPT_i}, $C_{OPT_{i-1}} > C_{SOL_{i-1}}$, contradiction occur. Hence, $C_{OPT_i} = C_{SOL_i}$.

Case 2: $C_{SOL_i} = C_{SOL_{i-1}}$

Base on our algorithm, $S_{SOL_i} = (S_{SOL_{i-1}} \cup \{i\}) \setminus \{\arg\max_{k \in (S_{SOL_{i-1}} \cup \{i\})} t_k\}$. This happens if $T_{SOL_{i-1}} + t_i > r_i$. By $T_{OPT_{i-1}} = T_{SOL_{i-1}}$, $T_{OPT_{i-1}} + t_i > r_i$. Suppose $C_{OPT_i} > C_{SOL_i}$, by removing $\{i\}$ from S_{OPT_i}, $T_{OPT_{i-1}} + t_i \leq r_i$, contradiction occur. Hence $C_{OPT_i} \leq C_{SOL_i}$, by our definition, $C_{OPT_i} = C_{SOL_i}$.

To prove $T_{OPT_i} = T_{SOL_i}$, we should prove that $\max_{k \in (S_{OPT_{i-1}} \cup \{i\})} t_k \leq \max_{k \in (S_{SOL_{i-1}} \cup \{i\})} t_k$. Suppose $\max_{k \in (S_{OPT_{i-1}} \cup \{i\})} t_k > \max_{k \in (S_{SOL_{i-1}} \cup \{i\})} t_k$. By the condition $C_{OPT_{i-1}} = C_{SOL_{i-1}}$, at least one element in $S_{SOL_{i-1}}$ is not $S_{OPT_{i-1}}$, lets say it's j. If we replace $\arg\max_{k \in (S_{OPT_{i-1}} \cup \{i\})} t_k$ with j in $S_{OPT_{i-1}}$, $T_{OPT_{i-1}}$ could be smaller, contradiction. The remaining step is same as case 1, and thus $T_{OPT_i} = T_{SOL_i}$

Conclusion: By the principle of induction, $\forall i > 0, C_{SOL_i} = C_{OPT_i} \wedge T_{SOL_i} = T_{OPT_i}$.

(4) (6%) Please design an algorithm with time complexity $O(NM)$ under **Assumption 4** to calculate the maximum candy you can get. Briefly explain the correctness and why your algorithm meets the time complexity requirement.

First, eliminate the professor i with $c_i = 0$ from candidate sequence using $O(N)$ time. Let's say the remaining sequence length is n. By the property 1 of previous proof, let's consider the r_i sequence in non-decreasing order, otherwise we could spend $O(n \log n)$ time to sort it. Let $dp_{i,j}$ be the minimum time to get j candy form first i professors with the following transition function.

$$dp_{i,j} = \begin{cases} \min(dp_{i-1,j-c_i} + t_i, dp_{i-1,j}), & \text{if } i > 0 \wedge j \geq 0 \\ 0, & \text{if } i = 0 \wedge j = 0 \\ \infty, & \text{if } i < 0 \vee j < 0 \end{cases}$$

The answer of this problem is k such that $dp_{n,k} \leq r_n$ and $dp_{n,i} > r_n$ for all $i > k$.

The table size of dp is nM with $O(1)$ transition time, the sorting time is $O(n \log n)$. By pigeonhole principle, $n < M$, and thus $\log n < M$, the total time complexity is $O(nM + n \log n) = O(nM) = O(NM)$.

Correctness:

Note that there are multiple approaches to prove these properties. Any valid proof will be considered correct. For convenience, suppose the original sequence is already sorted in non-decreasing r_i order.

Subproblems:

$T(i, j)$: the minimum time to get i candy form first j professors

Optimal Substructure:

Suppose OPT is an optimal solution for $T(i, j)$, there are only 2 possible cases.

Case 1: professor i is in OPT

$OPT \setminus \{i\}$ is an optimal solution to $T(i - 1, j - c_i)$

Case 2: professor i is not in OPT

OPT is an optimal solution of $T(i - 1, j - 1)$

By proving the optimal substructure, the algorithm is correct.

On the next day of Trick-or-Treating, you and your friend Ada would like to visit and thank all N professors individually. You know the i-th professor would like to chat with you and Ada for f_i and g_i unit time, respectively. In order to save time, you want to come up with two visit plans a and b for you and Ada, separately.

A visit plan is a non-negative sequence p with length N, indicating that one visits the i-th professor at time p_i and starts chatting until the conversation is over. It is possible that $a_i \neq b_i$ since Ada and you could visit professors in a different order.

Furthermore, the conversation should always be one-to-one; that is, one professor could only talk to one student at one time and vice versa. Thus, for all $1 \leq i \leq N$, either $b_i + g_i \leq a_i$ or $a_i + f_i \leq b_i$.

Now, your task is to construct two visit plans for you and Ada to minimize the ending time that both of you had visited the N professors individually.

To sum up, given the chatting time sequences f and g, please find out two visit plans a and b for you and Ada respectively such that $\max_{i=1}^{N}(\max(a_i + f_i, b_i + g_i))$ is minimum.

Please answer the following problems. **Note that if you answer (7) correctly, you will automatically get the score of (6).**

(5) (3%) Given the following information, please calculate the minimum ending time and construct a and b (if there are several possible answers, you could write down anyone arbitrarily).

- $N = 5$
- $f = [1, 2, 3, 5, 8]$
- $g = [13, 21, 1, 2, 3]$

- **Method**
  - I can visit professor $5, 3, 4$ respectively and continuously, waiting until time point 37, and then visit professor $1, 2$ respectively and continuously.
  - Ada can visit professor $3, 4, 1, 2, 5$ respectively and continuously.

  This generates $a = [37, 38, 8, 11, 0], b = [3, 16, 0, 1, 37]$, and satisfies the restrictions, lead to the minimum ending time 40.

(6) (3%) Prove that if it exists an x such that $(f_x + g_x) > \max(\sum_{i=1}^{N} f_i, \sum_{i=1}^{N} g_i)$, the minimum ending time is $f_x + g_x$.

(7) (4%) Please design an algorithm with time complexity $O(N)$ to construct two visit plans a and b. Briefly explain the correctness and why your algorithm meets the time complexity requirement.

- Hint: If there is no x such that $(f_x+g_x) > \max(\sum_{i=1}^{N} f_i, \sum_{i=1}^{N} g_i)$, then you may want to consider time arrangement of a specific professor x that satisfies $min(f_x, g_x) = \max_{i=1}^{N}(min(f_i, g_i))$ first

- **Method**

  (i) If there exists x s.t. $f_x \geq \sum_{i=1}^{N} g_i - g_x$ and $g_x \geq \sum_{i=1}^{N} f_i - f_x$, then I can visit professor x first, Ada can visit the other professor in any order but continuously. After I finish, Ada can visit professor x immediately, and I can visit the other professor in any order but continuously. And then finish the construction with ending time $f_x + g_x$. $(O(N))$

(ii) Otherwise, we can find x s.t. $\min(f_x, g_x) = \max_{i=1}^{N}\{\min(f_i, g_i)\}$. $(O(N))$.

(iii) We can divide the remain professor into two group, one is $f_i \geq g_i$ we define as $p_1 \sim p_L$, the other is $f_i < g_i$ we define as $q_1 \sim q_R$. $(O(N))$

(iv) Without loss of generality we assume that $\sum_{i=1}^{N} g_i \geq \sum_{i=1}^{N} f_i$, then,

- Ada can visit professor $p_1 \sim p_L$ in order and continuously, and then immediately visit professor $q_1 \sim q_R$ in order and continuously, finally, immediately visit professor x.
- I can visit professor x first, and then immediately visit professor $p_1 \sim p_L$ in order and continuously, and then wait until time point $\sum_{i=1}^{N} g_i - \sum_{i=1}^{R} f_{q_i}$, finally, immediately visit professor visit professor $q_1 \sim q_R$ in order and continuously.

And we can finish the construction with ending time $\sum_{i=1}^{N} g_i$(Or in case $\sum_{i=1}^{N} f_i > \sum_{i=1}^{N} g_i$, ending time is $\sum_{i=1}^{N} f_i$). $(O(N))$

The time complexity of the algorithm above is $O(N)$.

Proof,
For what the algorithm does in (i), since $f_x \geq \sum_{i=1}^{N} g_i - g_x$, Ada can definitely visit all of the professors without x during the time when I visit professor x, same as I visit all of the professors without x during the time when Ada visit professor x. So the ending time will be $f_x + g_x$.
This is the minimum ending time since it is impossible for Ada and I visit professor x at the same time, lead to that answer is greater or equal than $f_x + g_x$.

For what the algorithm does in (ii)∼(iv), since $\min(f_x, g_x) = \max_{i=1}^{N}\{\min(f_i, g_i)\} \Rightarrow f_x \geq g_{p_i} \forall 1 \leq i \leq L \Rightarrow f_x + \sum_{j=1}^{i-1} f_{p_j} \geq \sum_{j=1}^{i} g_{p_j} \forall 1 \leq i \leq L$, this tells us when I visit professor p_i, Ada must have finished chatting with professor p_i. Here we proved that the visiting of professor $p_1 \sim p_L$ will not conflict.
Same, let $S = \sum_{i=1}^{N} g_i$, since $\min(f_x, g_x) = \max_{i=1}^{N}\{\min(f_i, g_i)\} \Rightarrow g_x \geq f_{q_i} \forall 1 \leq i \leq R \Rightarrow g_x + \sum_{j=1}^{i-1} g_{q_j} \geq \sum_{j=1}^{i} f_{q_j} \forall 1 \leq i \leq R \Rightarrow S - (g_x + \sum_{j=1}^{i-1} g_{q_j}) \leq S - (\sum_{j=1}^{i} f_{q_j}) \forall 1 \leq i \leq R$, this tells us when I visit professor q_i, Ada must have finished chatting with professor q_i. Here we proved that the visiting of professor $q_1 \sim q_R$ will not conflict.
Finally, since there isn't exists x s.t. $f_x \geq \sum_{i=1}^{N} g_i - g_x$ and $g_x \geq \sum_{i=1}^{N} f_i - f_x$, we have $f_x < \sum_{i=1}^{N} g_i - g_x$ or $g_x < \sum_{i=1}^{N} f_i - f_x$, impling that the visiting of professor x will not conflict, so the ending time will be $\sum_{i=1}^{N} g_i$(Or in case $\sum_{i=1}^{N} f_i > \sum_{i=1}^{N} g_i$, ending time is $\sum_{i=1}^{N} f_i$).
This is the minimum ending time since the answer is same as $\max\{\sum_{i=1}^{N} g_i, \sum_{i=1}^{N} f_i\}$, and both I and Ada can't visit two or more professors at the same time, lead to the answer is greater or equal than $\max\{\sum_{i=1}^{N} g_i, \sum_{i=1}^{N} f_i\}$. □

Problem 6 - String Problems (Hand-Written) (20 points)

*In problem 6, please **briefly** explain your solution in text. **Do not** use pseudo code, or you will receive penalty. Note that if you use Greedy or Dynamic Programming in any subproblem of this problem, you should prove their properties (optimal substructure, greedy-choice property).*

You're given two strings s1 and s2 ($1 \leq |s1|, |s2| \leq N$). Please determine whether it's possible to change the string from s1 to s2 with no more than K moves. (one addition, deletion, or replacement of one character counts as one move)

Please answer the following problems. **Note that if you answer (3) correctly, you will get the score of (1), (2) automatically.**

(1) (3%) If $K = 1$, Please design an algorithm with time complexity $O(N)$ to answer the questions. Briefly explain the correctness and why your algorithm meets the time complexity requirement.

(2) (3%) If $K > 1$, Please design an algorithm with time complexity $O(NK^2)$ to answer the questions. Briefly explain the correctness and why your algorithm meets the time complexity requirement.

(3) (4%) If $K > 1$, Please design an algorithm with time complexity $O(NK)$ to answer the questions. Briefly explain the correctness and why your algorithm meets the time complexity requirement.

- **Method**

  Consider the method of the **Sequence Alignment Problem** in the course(Slide Dynamic-Programming-1 p54), we have an $O(N^2)$ dp method with $C_{\text{INS}} = C_{\text{DEL}} = 1, C_{p,q} = [s_{1,i} = s_{1,q}]$ which has been proved can calculate the minimum cost of changing s_1 to s_2. We rewrite the dp table as $dp[i][j]$ is the minimum cost of changing s_1 to s_2 and its transitive function is,

$$dp[i][j] = \begin{cases} j & \text{if } i = 0 \\ i & \text{otherwise if } j = 0 \\ \min\{dp[i-1][j-1] + [s1_i = s2_j], & \text{otherwise} \\ \quad dp[i-1][j] + 1, dp[i][j-1] + 1\} & \end{cases}$$

  To modify the algorithm become $O(NK)$, we can simply ignore $dp[i][j]$ with $|i - j| > K$ since it will not be a valid transitive source. That is because if we want to modify a string to a new string with length difference exceed K, than we must cost exceed K. Thus, we redefine the dp table to,

$$dp[i][j] = \begin{cases} j & \text{if } i = 0 \\ i & \text{otherwise if } j = 0 \\ K + 1 & \text{otherwise if } |i - j| > K \\ \min\{dp[i-1][j-1] + [s1_i = s2_j], & \text{otherwise} \\ \quad dp[i-1][j] + 1, dp[i][j-1] + 1\} & \end{cases}$$

  For clearly implementation, we don't need to declare a $O(N^2)$ dp table, for each $1 \leq i \leq |s1|$, we just need to declare at most $2K + 1$ space for $dp[i][i - K] \sim dp[i][i + K]$, and write a function return the dp value, which can use some if else statements to return the value when $i = 0, j = 0$ or $|i - j| > K$. Hence, we only need to do $O(NK) \times O(1) = O(NK)$ calculation and reach the time complexity of $O(NK)$. And the answer obviously is deciding if $dp[|s1|][|s2|] \leq K$.

You're given two strings s1 and s2 ($1 \leq |s1|, |s2| \leq N$) over alphabet Σ. Consider all possible strings S over alphabet Σ, such that both of s1 and s2 are subsequence of S. Over all possible S, you want to find the one with minimum $D(S)$. If there is more than one possible S, anyone is acceptable.

- For any two characters c_1 and c_2 from Σ, we define $Dis(c_1, c_2)$ as the distance from c_1 to c_2.(You could consider $Dis(c_1, c_2)$ as a non-negative integer given by the problem)
- for any three characters c_1, c_2, c_3 from Σ, satisfied $Dis(c_1, c_2) + Dis(c_2, c_3) \geq Dis(c_1, c_3)$
- If $|S| = 1, D(S) = 0$
- If $|S| > 1, D(S) = \Sigma_{i=0}^{|S|-2}(Dis(S_i, S_{i+1}))$
- $|\Sigma| = K$
- A subsequence is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.

Please answer the following problems. **Note that if you answer (6) correctly, you will get the score of (5) automatically.**

(4) (3%) Given the following information, please find the S with minimum $D(S)$ and also the value of $D(S)$.

- $\Sigma = \{a, b, c, d, e, f\}$
- $Dis(c_1, c_2) = |ord(c_1) - ord(c_2)|$
- ord(c) returns the ASCII value of the character
- $s1 =$ "$adefc$"
- $s2 =$ "$accfd$"

We can construct the string $accdefdc$ to reach the minimum $D(S) = 2+0+1+1+1+2+1 = 8$.

(5) (3%) Please design an algorithm with time complexity $O(N^2K)$ to find the S with minimum $D(S)$ and also the value of $D(S)$. Briefly explain the correctness and why your algorithm meets the time complexity requirement.

(6) (4%) Please design an algorithm with time complexity $O(N^2)$ to find the S with minimum $D(S)$ and also the value of $D(S)$. Briefly explain the correctness and why your algorithm meets the time complexity requirement.

Method

(i) Let $dp[i][j][0]$ be the minimum $D(S)$ of the subproblem of $s1_{1..i}$ and $s2_{2..j}$, and S will end with $s1_i$, $dp[i][j][1]$ be the minimum $D(S)$ of the subproblem of $s1_{1..i}$ and $s2_{2..j}$, and S will end with $s2_j$.

(ii) Define $s1_0 = s2_0 = \epsilon$, and $Dis(\epsilon, c) = 0$ for any $c \in \Sigma$, the transitive function will be,

$$dp[i][j][0] = \begin{cases} D(s1_{1..i}) & \text{if } j = 0 \\ \infty & \text{otherwise if } i = 0 \\ \min\{dp[i-1][j-1][0] + Dis(s1_{i-1}, s1_i), & \text{otherwise if } s1_i = s2_j \\ dp[i-1][j-1][1] + Dis(s2_{j-1}, s1_i), & \\ dp[i-1][j][0] + Dis(s1_{i-1}, s1_i), & \\ dp[i-1][j][1] + Dis(s2_j, s1_i)\} & \\ \min\{dp[i-1][j][0] + Dis(s1_{i-1}, s1_i), & \text{otherwise} \\ dp[i-1][j][1] + Dis(s2_j, s1_i)\} & \end{cases}$$

$$dp[i][j][1] = \begin{cases} D(s2_{1..j}) & \text{if } i = 0 \\ \infty & \text{otherwise if } j = 0 \\ \min\{dp[i-1][j-1][0] + Dis(s1_{i-1}, s2_j), & \text{otherwise if } s1_i = s2_j \\ dp[i-1][j-1][1] + Dis(s2_{j-1}, s2_j), & \\ dp[i][j-1][0] + Dis(s1_i, s2_j), & \\ dp[i][j-1][1] + Dis(s2_{j-1}, s2_j)\} & \\ \min\{dp[i][j-1][0] + Dis(s1_i, s2_j), & \text{otherwise} \\ dp[i][j-1][1] + Dis(s2_{j-1}, s2_j)\} & \end{cases}$$

(iii) The answer is $\min\{dp[|s1|][|s2|][0], dp[|s1|][|s2|][1]\}$.

Since the size of the dp table is $2|s1||s2| = O(N^2)$, and the transitive time is $O(1)$, the time of building the dp table which euqal to the total time is $O(N^2)$.

Proof,
For any optimize solution OPT of $dp[i][j][0]$, if we remove the last character($s1_i$), the remaining string could lead to four kind of subproblems,

1. $dp[i-1][j][0]$, the remain string have finished the subproblem of $s1_{1..i-1}$ and $s2_{1..j}$ and end with $s1_{i-1}$.
2. $dp[i-1][j][1]$, the remain string have finished the subproblem of $s1_{1..i-1}$ and $s2_{1..j}$ and end with $s2_j$.
3. $dp[i-1][j-1][0]$, the remain string have finished the subproblem of $s1_{1..i-1}$ and $s2_{1..j-1}$ and end with $s1_{i-1}$. This case could only happen if $s1_i = s2_j$ since we will want to let the two characters share the same character in S.
4. $dp[i-1][j-1][0]$, the remain string have finished the subproblem of $s1_{1..i-1}$ and $s2_{1..j-1}$ and end with $s2_{j-1}$. This case could only happen if $s1_i = s2_j$ since we will want to let the two characters share the same character in S.

It is obviously that if we remove the last character($s1_i$), $s1_{1..i-1}$ will still be a subsequence of S, $s2_{1..j-1}$ will still be a subsequence of S, $s2_{1..j}$ will still be a subsequence of S iff $s1_i$ doesn't share the same character with $s2_j$.
The remaining problem is that why the ending character could only be s_{i-1}, s_{j-1} or s_j. Since for any three characters c_1, c_2, c_3 from Σ, satisfied $Dis(c_1, c_2) + Dis(c_2, c_3) \geq Dis(c_1, c_3)$, this impiles that if there is a extra character between two useful character, we can remove it then get a solution that

won't get worse. So the transitive function of the dp is reasonable. Similar to $dp[i][j][1]$.
After we prove the correctness of the dp table, the method of getting the answer have only a problem, why we can guarantee that the last character is $s1_{|s1|}$ or $s2_{|s2|}$? And this is trivial since we can remove the last character because it must not useful, and won't get a worse solution.
Hence, we proved the correctness of our algorithm. □