Data Structure and Algorithm, Spring 2025
Mini-Homework B

TA E-mail: dsa_ta@csie.ntu.edu.tw

Due: 13:00:00, Tuesday, April 15, 2025

Rules and Instructions

- Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

- Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources (including chatGPT) can be consulted, but not copied from.

- Since everyone needs to write the final solutions alone, there is **absolutely no need to lend your homework solutions and/or source codes to your classmates at any time.** In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

- In each mini-homework, we will have one programming problem that is closely connected to what we have taught in the class. The purpose of the problem is for you to *practice* what you have learned, before conquering the more challenging problems in other homework sets.

- Like other programming problems, you should write your code in C programming language, and then submit the code via the *DSA Judge*:

  https://dsa-2025.csie.org/.

  Each day, you can submit up to 5 times for each problem. The judge system will compile your code with

  ```
  gcc [id].c -static -O2 -std=c11
  ```

- For debugging in the programming part, we strongly suggest you produce your own testcases with programs and/or use tools like `gdb` or compile with flag `-fsanitize=address` to help you solve issues like illegal memory usage. For `gdb`, you are strongly encouraged to use compile flag `-g` to preserve symbols after compiling.

  Note: For students who use IDE (VScode, DevC++...) you can modify the compile command in the settings/preference section. Below are some examples:

  - `gcc [id].c -O2 -std=c11 -fsanitize=address # Works on Unix-like OS`
  - `gcc [id].c -O2 -std=c11 -g # use it with gdb`

- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

  $$\textit{Late Score} = \max\left(\left(\frac{5 \times 86400 - \textit{Delay Time}(\textit{sec.})}{5 \times 86400}\right) \times \textit{Original Score}, 0\right)$$

  Please note that the "gold medal" of the class cannot be used on mini-homework problems.

- If you have questions about this mini-homework, please go to the Discord channel and discuss the issues with your classmates. This is *strongly preferred* because it will provide everyone with a more interactive learning experience. If you really need email clarifications, you are encouraged to include the following tag in the subject of your email:

  - The subject should contain one tag, `"[hwb]"`, specifying the problem where you have questions. For example, `"[hwb] Will there be a repeated number?"`. Adding these tags potentially allows the TAs to track the status of each email and to provide faster responses to you.
  - If you want to provide your code segments to us as part of your question, please upload it to Gist or similar platforms and provide the link. Please remember to protect your uploaded materials with proper permission settings. Screenshots or code segments directly included in the email are discouraged and may not be reviewed.

Programming Part

Reverse Insertion Sort (15 pts)

Problem Description

Given an array **a** with N numbers $\mathbf{a}[1], \mathbf{a}[2], \ldots, \mathbf{a}[N]$ (possibly repeated), run the following **reverse** insertion sort algorithm on the array, as modified from Lecture 2 of the class, for a budget of B back-moves or until the array is sorted. The back-move is defined as Line 4 of the INSERT-BIGGER routine.

The budget part is defined as follows. We add a condition to Line 3 of the **while** loop in INSERT that checks whether there is still any back-move budget. If so, Line 4 and Line 5 are executed; if not, the **while** loop will not be entered (but *data* will still be inserted in Line 6). Then, print out the content of the array.

Note that there are many different variant versions of the insertion sort algorithm. You are asked to implement the version below *exactly* to produce the correct answer for this problem.

```
INSERTION-SORT($A$)
1  for $i = 1$ to $A.length$
2      INSERT-BIGGER($A, i$)
3  return $A$
```

```
INSERT-BIGGER($A, m$)
1  $data = A[m]$
2  $i = m - 1$
3  while $i > 0$ and $A[i] < data$
4      $A[i+1] = A[i]$
5      $i = i - 1$
6  $A[i+1] = data$
```

Input

The first line includes two integers N and B, representing the size of the array and the budget on the number of back-moves. The second line includes N integers, representing the elements of the array $\mathbf{a}[1], \mathbf{a}[2], \ldots, \mathbf{a}[N]$. All numbers are separated by a space.

Output

- If the array can be sorted within B back-moves, output a line of

  ```
  The array is [the content of the array] after [X] back-moves.
  ```

  where `[the content of the array]` lists the final $\mathbf{a}[1], \mathbf{a}[2], \ldots, \mathbf{a}[N]$ after sorting, and `[X]` is the actual number of back-moves taken, which would be a unique number using the version of the insertion sort algorithm above.

- If the array is not sorted after B back-moves, output a line of

  ```
  The array is [the content of the array] after [B] back-moves.
  ```

  where `[the content of the array]` lists the intermediate $\mathbf{a}[1], \mathbf{a}[2], \ldots, \mathbf{a}[N]$ after the B back-moves, and `[B]` is just the number B.

Constraint

- $1 \le N \le 2^{10}$
- $1 \le B \le 2^{20}$
- $-2^{30} \le \mathbf{a}[n] \le 2^{30}$ for $n \in \{1, 2, \ldots, N\}$

Sample Testcases

Sample Input 1

```
3 2
2 3 1
```

Sample Output 1

```
The array is 3 2 1 after 1 back-moves.
```

Sample Input 2

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

Sample Output 2

```
The array is 4 3 1 2 after 2 back-moves.
```

Sample Input 3

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

Sample Output 3

```
The array is 4 2 1 2 after 2 back-moves.
```

Hint

- By design, you can pass this homework by simulating the insertion sort algorithm properly. There is no need for other arithmetic calculations or cuts.