Data Structure and Algorithm, Spring 2024
Mini-Homework C

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

Due: 13:00:00, Tuesday, April 16, 2024

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://dsa2024.csie.org/.

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

  ```
  gcc main.c -static -O2 -std=c11
  ```

- For debugging in the programming part, we strongly suggest you produce your own test-cases 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 main.c -O2 -std=c11 -fsanitize=address # Works on Unix-like OS`
  - `gcc main.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:

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

- If you have questions about this mini-homework, please go to the 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 an email answer, please follow the rules outlined below to get a fast response:

  - The subject should contain one tag, `"[hwc]"`, specifying the problem where you have questions. For example, `"[hwc] Will there be a repeated number?"`. Adding these tags 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

Binary Search (20 pts)

Problem Description

Given an ordered array $\mathbf{a}$ with N numbers $\mathbf{a}[1], \mathbf{a}[2], \ldots, \mathbf{a}[N]$ (possibly repeated), with

$$\mathbf{a}[1] \leq \mathbf{a}[2] \leq \cdots \leq \mathbf{a}[N],$$

run the following binary search on the array to decide whether some *key* is within $\mathbf{a}$, as introduced in Lecture 2 of the class.

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

```
BIN-SEARCH(a, key, ℓ, r)
1  while ℓ ≤ r
2      m = floor((ℓ + r)/2)
3      if a[m] equals key
4          return m
5      elseif a[m] > key
6          r = m − 1 // cut out end
7      elseif a[m] < key
8          ℓ = m + 1 // cut out begin
9  return NIL
```

Input

The first line includes two integers N and *key*, representing the size of the array and the *key* for searching. 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

- For each execution of Line 1 of the binary search algorithm above, print a line of

  `Searching key in range [l, r].`

  with `key` being key, `l` being ℓ and `r` being r.

- Then, if the algorithm returns in Line 4, print a line of

  `Found at index m.`

  with `m` being the m returned.

- Otherwise, if the algorithm returns in Line 9, print a line of

  `Not found.`

Constraint

- $1 \leq N \leq 2^{22}$
- $1 \leq key \leq 2^{22}$
- $1 \leq \mathbf{a}[1] \leq \mathbf{a}[2] \leq \mathbf{a}[3] \leq \cdots \leq \mathbf{a}[N] \leq 2^{22}$

Sample Testcases

Sample Input 1

```
3 4
1 2 4
```

Sample Output 1

```
Searching 4 in range [1, 3].
Searching 4 in range [3, 3].
Found at index 3.
```

Sample Input 2

```
3 3
1 2 4
```

Sample Output 2

```
Searching 3 in range [1, 3].
Searching 3 in range [3, 3].
Searching 3 in range [3, 2].
Not found.
```

Sample Input 3

```
4 64
1 2 89 1126
```

Sample Output 3

```
Searching 64 in range [1, 4].
Searching 64 in range [3, 4].
Searching 64 in range [3, 2].
Not found.
```

Hint

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