The 3rd Problem Set

January 15, 2018

Pre-requisites

You should do the followings

1. On a separate and clean paper, you need to describe your own strategy to solve the problems below, and to justify why your strategy is effective while handling each problem
2. On a new clean paper, transform your strategy into an algorithm, using your own form to express algorithms. Further, you need to analyze the total running steps of your algorithm and the required memory amount to finish your algorithm. Then express the total costs using Big-O notation.
3. Use the `Code ocean` (`https://codeocean.com`) platform; if necessary, you may invite me using my email address `lightsun.kim@gmail.com`.
4. Time limits for each problem
   - Problem #1: Within 5 hours
   - Problem #2: Within 3 hours
   - Problem #3: Within 3 hours

   Then you need to prepare two answer codes; one is a C code that you have made within each time limit, and the other is a C code augmented and fixed from the original code later.

1 Problem #1

In this problem, we consider the max subarray problem. Here we are given an integer array A and asked to find the subarray whose elements have the largest sum. More formally, given array $A = [a_1 : a_n] = [a_1, a_2, \ldots, a_n]$, you are requested to find indices α and β that maximize the sum

$$S_{\alpha:\beta} = a_\alpha + a_{\alpha+1} + \cdots + a_\beta = \sum_{\alpha \le i \le \beta} a_i.$$

You should note that each a_i is an integer, and thus it could have a positive, negative, or zero value.

Throughout Problem #1, we let $A[0] = 0$, and let $A[\alpha : \beta]$ denote the sequence of elements of A from index α to $\beta (0 \le \alpha \le \beta \le n)$. To wrap up, the max subarray problem consists of finding the sequence $A[\alpha : \beta]$ that maximizes $S_{\alpha:\beta}$, the sum of its values. Then such a max sum is called the max subarray sum of array A.

For example, given an array $A[1 : 11] = [-2, -4, 3, -1, 5, 6, -7, -2, 4, -3, 2]$, the max subarray is $A[3 : 6]$ and the max subarray sum is $S_{3,6} = 13$.

For this problem, a naive solution is the following algorithm, named MaxsubCubic:

```
Algorithm MaxsubCubic
INPUT: An $n$-element integer array $A$ whose elements are indexed from 1 to $n$.
OUTPUT: The max subarray sum $S_{\alpha:\beta}$ and its pair of indices $(\alpha, \beta)$ of $A[\alpha : \beta]$.
 1.  max $\leftarrow 0$
 2.  for $j \leftarrow 1$ to $n$
 3.      for $k \leftarrow j$ to $n$
 4.          $S \leftarrow 0$
 5.          for ____________________________          (1)
 6.              $S \leftarrow$____________________________          (2)
 7.          if $S >$ max then
 8.              max $\leftarrow S$
 9.              $(\alpha, \beta) \leftarrow (j, k)$
10.  return max, $(\alpha, \beta)$
```

Here you can easily see that thanks to for-loop in Line 5, the running time of the MaxsubCubic is $O(n^3)$. Therefore, this complexity is really bad, especially when the index n is huge. Now, your main mission in this problem is to find a way to improve this algorithm so that its running time complexity has only $O(n)$.

Language requirements. During tackling this problem, you should follow the programming rules:

- You should use an ANSI C programming language whose source code can run on `Code ocean` platform.
- Function naming: Begin with the lower character, and every parameters are strong-typed variables (i.e., do not use `void` typed variables). All functions should have a single return value; thus even if a function will return no values; you should provide `return` keyword.
- Variable naming: Begin with a type-discriminating prefix. For example, if a variable name is for an age and is with an integer type, you need to declare the variable as `int iAge;` Especially for string-type variables you are strongly recommended to use the prefix `sz`. For example, if a variable name is for a name, then `szName` is a preferable choice.

Input format. The input is given a text-format file, named `input.txt` and all strings are separated by commas. You should take as input integers a_i with $i \ge 1$ and construct an integer array $A[1 : n] = [a_1, a_2, \ldots, a_n]$. Note that the input file does not include the size of the array A; thus the input file is in form as follows:

$a_1, a_2, \ldots, a_n$

Output format. The output should be given as a text-format file, named `outputc.txt`, `outputq.txt`, or `outputl.txt`. The output file writes (1) the max subarray $A[\alpha : \beta]$, (2) its sum $S_{\alpha:\beta}$, and (3) the execution time as follows:

```
*************************************
By Maxsub{Cubic or Quadratic or Linear}
*************************************
** The max subarray **
A[α : β]=[a_α, a_{α+1}, . . . , a_β]

** Its sum **
S[α : β]=

The total execution time: _______ sec
```

You should provide three programs: The first program is your C code of MaxsubCubic that completes (1) and (2) of Line 5 and 6 in the algorithm above. Then you need to improve the algorithm so that it has the running time $O(n^2)$; this improved algorithm is named MaxsubQuadratic. Finally you should enhance your MaxsubQuadratic algorithm so that it has the running time $O(n)$ whose name is given by MaxsubLinear. Moreover, your codes should provide the execution times of all your C programs using described in Problem #2 of the first problem set.

2 Problem #2

Let $\mathbb{Z}$ be the set of integers, and let $a, b \in \mathbb{Z}$. We say that a divides b if there exists an integer k such that $b = ak$ and we write it as $a|b$. Given two integers a, b, a common divisor d of a and b is $d|a$ and $d|b$; moreover we call such a d the greatest common divisor (GCD) of a and b if $d \geq 0$ and all other common divisors of a and b divide d. For $a, b \in \mathbb{Z}$, a common multiple of a and b is an integer m such that $a|m$ and $b|m$; moreover, such an m is the least common multiple (LCM) of a and b if $m \geq 0$ and m divides all other common multiples of a and b.

Let $S = \{a_1, a_2, \ldots, a_n\} \subset \mathbb{Z}$ for some $n > 1$. Let $\ell = \binom{n}{2} = \frac{n!}{2!(n-2)!}$. For notational convenience, we use the following symbols

- For each of two distinct elements $a, b \in S$, we have the GCD of a and b; we write each GCD as $d_i = \gcd(a, b)(1 \leq i \leq \ell)$.
- Similarly, for each of two distinct elements $a, b \in S$, we have the LCM of a and b; we write each LCM as $m_i = \operatorname{lcm}(a, b)(1 \leq i \leq \ell)$.

In this problem, your mission is to develop a C code to find two pairs of values $X := (a, b)$ and $Y := (a^*, b^*)$ so that when $d = \gcd(a, b)$ and $m^* = \operatorname{lcm}(a^*, b^*)$ for some $a, b, a^*, b^* \in S$, one has $d = \min_{1 \leq i \leq \ell} d_i$ and $m^* = \max_{1 \leq i \leq \ell} m_i$.

Input format. The input is given a text-format file, named `input.txt` and all strings are separated by commas.

```
n
{a_1, a_2, ..., a_n}
```

where n is the number of elements is a set S. Namely, the set $S = \{a_1, a_2, \ldots, a_n | a_i \in \mathbb{Z}\}$ and an integer $n \geq 10$. We assume that every integer has the length of less than or equal to 32 bits, but note that S may have a negative element. One way to have such a set is (1) to randomly generate n integers, (2) to choose a positive integer $k < n$ randomly, (3) to create a set of random indices $J = \{i_1, i_2, \ldots, i_k\}$, and (4) to set a_{i_j} into $-a_{i_j}$ for each $j \in J$.

Output format. The output should be given as a text-format file, named `output.txt`. The output file writes the resulting values d and d^* along with the corresponding pairs (a, b) and (a^*, b^*) such that $d = \gcd(a, b)$ and $m^* = \operatorname{lcm}(a^*, b^*)$ under the condition described as above.

```
S={a_1, a_2, ..., a_n}

** The smallest GCD **
d=gcd(a, b)

** The largest LCM **
m*=lcm(a*, b*)
```

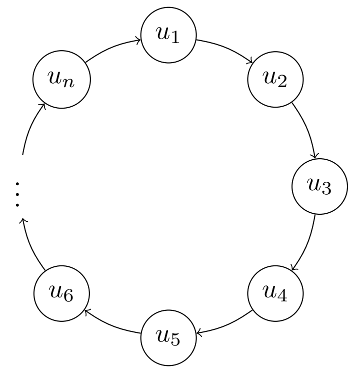
3 Problem #3

The Josephus problem is mentioned in Exercise **A-2.1** of the textbook [GT15], but the problem in its original form goes long back to the Roman historian Flavius Josephus [Jos27].

There has been known an interesting history as follows: In the Romano-Jewish conflict of 67 A.D., finally the Romans occupied the town "Jotapata" which *Josephus* was ruling. During the last battle against Rome's army, he and 40 companions escaped and trapped in a cave. They have fallen the fear of death, and so they decided to kill themselves at all time. At this point of decision, Josephus and a friend did not agree with that proposal, of course, but were afraid to stay in their opposition. Thus they (i.e., Josephus and the friend) suggested that they should arrange them in a circle and that counting around the circle in the same sense all the time, every third man should be killed until there was only one survivor who would kill himself. By choosing the position 31 and 16 in the circle Josephus and his companion saved their lives.[1]

Historically, it is not clear that he suggested such an idea to take care about only his own survival. In other words, no one knows the truth of the day among them. Today, one apparent thing is that a quite interesting problem, called the Josephus problem, has been introduced.

We now describe the Josephus problem more formally and generally. There is a group of n people ($n = 41$ in the case of Josephus and his 40 companions) and let each person (say u_i, $1 \leq i \leq n$) stand in a node of a circle, numbered consecutively clockwise from 1 to n as follows.



Then staring with person no. 1 (i.e., u_1), it kills u_2, u_3 kills u_4, u_5 kills u_6, and proceeding clockwise in the same manner. We continue the process with the remaining people; that is, u_1 kills u_3, u_5 kills u_7, and proceeding clockwise. We will continue this process until only one person survives. For example, suppose that $n = 8$.

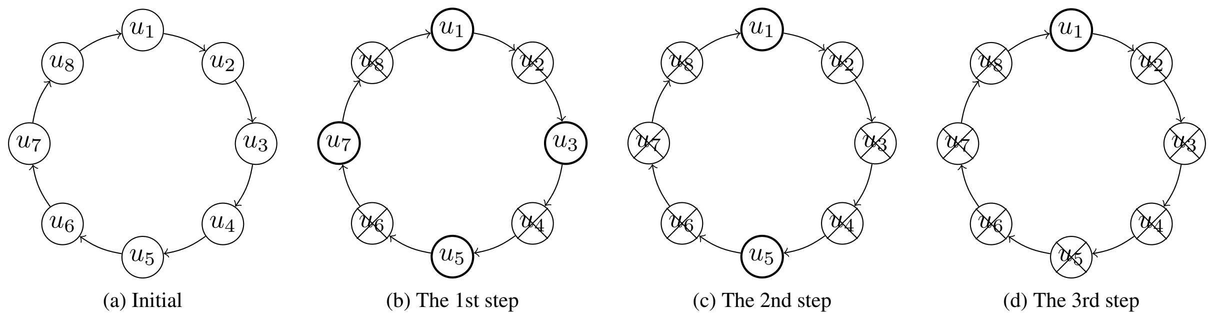


Figure 1: The case of $n = 8$

As you can see, during the first step, u_2, u_4, u_6, and u_8 will be kill in order. Then in the next step, u_1 kills u_3 and u_5 kills u_7. These processes are depicted in Figures from 1.(a) to 1.(c) in sequence. Lastly, person no. 1 (u_1) kills

[1] How did Josephus become Roman historian? How survived? Why historian of all choices?; Quite interesting :-)

person no. 5 (u_5); thus the last survival is u_1. This is given in Figure 1.(d).

Given the size of group n, your mission is to find the index of person who finally survives in this game. To do this, you should implement a C code which must **use a linked list** and **work recursively**.

Input format. The input is given a text-format file, named `input.txt` and all strings are separated by commas. You should take as input the size integers n with $i \geq 1$.

```
n=n
```

Output format. The output should be given as a text-format file, named `outputc.txt`, `outputq.txt`, or `outputl.txt`. The output file writes every survived person in each step following the format as described below.

```
** The initial step **
u_1 >> u_2 >> ··· >> u_n

** The 1st step **
u_{i_1} >> u_{i_2} >> ··· >> u_{i_k}

** The 2nd step **
u_{j_1} >> ··· >> u_{j_ℓ}
.
.
.
The last survival is person no. __
```

where all subscript numbers $i_1, \ldots, i_k, j_1, j_\ell \in \{1, \ldots, n\}$.

References

[GT15] M. Goodrich and R. Tamassia, *Algorithm design and applications*, Wiley.

[Jos27] Flavius Josephus, *The jewish war Book III*, Translated by H. S. Thackeray, Heinemann 1927, 342–366 & 387–391.