National University of Singapore

School of Computing

CS2040S - Data Structures and Algorithms

Midterm Quiz @ LT15

(Wed, 20 Sep 2023, S1 AY2023/24, 70m)

INSTRUCTIONS TO CANDIDATES:

1. You can start immediately after the password to open this PDF file is given.
   You can also opt for the printed version of the question paper if that is your preference.
2. This assessment paper contains FOUR (4) sections.
   It comprises SIX (6) printed pages for the question paper, including this page.
   Then, there is a separate answer sheets with FOUR (4) pages.
3. This is an **Open Book Assessment**.
4. Additionally, this is also an **Open Laptop Assessment (in airplane/no Internet mode)**.
   You are only allowed to use mouse/trackpad in your laptop (no typing).
5. Answer **ALL** questions within the **boxed space** of the answer sheets.
   You will only need to hand over the answer sheets (page 7-10) after this quiz.
   You can use either pen or pencil. Just make sure that you write **legibly**!
6. You must shade your Student Number *twice*, using (2B) pencil, on page 7 and page 9.
7. The total marks of this Midterm Quiz is 77.
   It will be scaled down by a factor of 7 so that a score of 77 gets the full 11%.

Steven printed 131 answer sheets but only 33% of 131 = 44 questions papers this semester to continue the experiment from last year (only 31.5% students out of 208 CS2040C students used hardcopy paper). Let's see what is the rate this time.

A Sorting Problem (24 marks)

You are given an integer n ($1 \leq n \leq 2 \times 10^5$) in the first line that describes an array A.
Then n single-space-separated *distinct* Integers $\in [-100\,000..99\,999]$ in the second line.
Notice that there are at most 2×10^5 distinct Integers in this range, same as the largest n.

Your task is to output one line that contains the indices (in 1-based, single-space-separated) where each Integers should be moved so that array A become sorted in ascending order.

For example, if you are given the following example input:

```
4
777 -77 -7 7777
```

Then, you should output:

```
3 1 2 4
```

Explanation: The sorted ascending order of $A = \{777, -77, -7, 7777\}$ is $\{-77, -7, 777, 7777\}$.
Thus, the first (777) / second (-77) / third Integer (-7) of A should be moved to
index three / one / two (1-based), respectively, so that A become sorted in ascending order.
Note that fourth Integer (7777) remains at index four for this example.

A.1 Manual Test Cases ($4 \times 2 = 8$ marks)

You are given four small test cases below.
For each test case, write down the answer (1 mark) and a short explanation (the other 1 mark).

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

This is a giveaway 2 marks. The output is trivial, as A is already sorted:

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

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

This is another giveaway 2 marks. A contains [1..7] in reverse sorted order:

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

iii. 7
```
20000 70000 50000 30000 10000 40000 60000
```

A contains [1..7] in some random order, but the values in A (after dividing by 10000) already describes how each Integer should be moved:

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

iv. 7

```
-5 12 7 -2 9 -3 -100000
```

After sorting, A becomes $\{-100000, -5, -3, -2, 7, 9, 12\}$, so the required output is:

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

A.2 Propose an Algorithm and Analyze It (16 marks)

Solve this problem and analyze the time complexity of your algorithm. You can use pseudo-code.
If your algorithm is **incorrect**, you will get 0 **mark**.
If you leave this box **blank**, you will automatically get a **free** 1 **mark** .
Correct $O(n^3)/O(n^2)/O(n \log n)/O(n)$ algorithm will get 4/7/12/16 marks, respectively.

Actually it is hard to end up with a $O(n^3)$ solution, and this is just a placeholder in case someone does (repeated $O(n \log n)$ or $O(n^2)$ sorting perhaps).

For $O(n^2)$ solution, we can store N Integers in ArrayList A and $Asorted$ (two copies), then use Collections.sort to sort $Asorted$ in $O(n \log n)$ time. But then use an $O(n^2)$ algorithm to find where is $A[i]$ in $Asorted$ (index j) using a naive two nested loops...

For $O(n \log n)$ solution, one can simply improve the previous solution by using $O(\log n)$ binary search to find where $A[i]$ is located in the sorted $Asorted$.

An alternative $O(n \log n)$ solution is to store n pairs of information (original Integer, and 1-based index), sort this pairs based on the distinct original Integer, then use reverse mapper (of size n) to map $A.get(i).second()$ to index $i+1$. Basically, the $A.get(i).second()$'th index in the original A moves to index $(i+1)$ to get A to be sorted.

For the $O(n)$ solution, one needs to use Counting Sort (with offset of 100K so that -100K becomes 0 and 100K becomes 200K, respectively) and simply do a prefix sum of the frequencies $k = 200K$, $k = O(n)$, so $O(n+k) = O(2n) = O(n)$. Detailed Java solution code is not shown as Prof Halim may turn this into a future PS2 (sorting-related problem) task.

B Stack with 'findMin()' Operation (24 marks)

Assuming that we are dealing with a stack of nn-negative Integers, let's define one *new* operation of a standard Stack on top of what we have discussed in class. This new operation is 'findMin()' that simply returns the current *minimum* Integer currently inside the stack (or -1 if the stack is currently empty). Show how you are going to implement this new 'findMin()' operation in $O(1)$ **while maintaining that the standard peek, pop, push operations of a stack all remains** $O(1)$ **too**.

For example, if stack S contains : 1 (top) $\rightarrow$ 5 $\rightarrow$ 1 $\rightarrow$ 6 $\rightarrow$ 8 $\rightarrow$ 7 $\rightarrow$ 2, then a call of 'findMin()' should report 1. If we pop the topmost element 1, 'findMin()' should still report 1 (there is still the other copy of 1). However if we then pop the next two topmost elements 5 and then (the other) 1, then 'findMin()' should now report 2.

A skeleton Java code has been written for you below. Please complete and/or modify it as you see fit. If you are not sure of the required Java syntax, you can write your answer in pseudo-code for slightly lesser marks. Also, if you are unable to implement this new 'findMin()' operation in $O(1)$, you can answer in $O(n)$, for approximately half marks.

```
import java.util.Stack;

class SpecialStack {
  private Stack<Integer> internalS; // basically a normal stack

  public SpecialStack() {
    internalS = new Stack<>();

  }

  public int peek() { // return the topmost Integer (-1 if empty)
    int ret = -1;



    return ret;
  }

  public int pop() { // pop+return the topmost Integer (-1 if empty)
    int ret = -1;









    return ret;
  }
```

```
  public void push(int value) { // put value at the top of Stack
    internalS.push(value); // O(1)




  }

  public int findMin() { // report the min Integer (-1 if empty)
    int ret = -1;









    return ret;
  }
}

// using assertions, so run Java with -ea (enableassertions) flag:
// java -ea SpecialStackTest
class SpecialStackTest {
  public static void main(String[] args) {
    SpecialStack S = new SpecialStack();
    assert S.peek() == -1;
    assert S.findMin() == -1;
    S.push(2); S.push(7); S.push(8); S.push(6);
    S.push(1); S.push(5); S.push(1);
    assert S.peek() == 1;
    assert S.findMin() == 1;
    assert S.pop() == 1; // we pop 1 out and return 1
    assert S.peek() == 5;
    assert S.findMin() == 1; // there were two copies of 1 before
    assert S.pop() == 5; // we pop 5 out and return 5
    assert S.pop() == 1; // we pop (the other) 1 out and return 1
    assert S.peek() == 6;
    assert S.findMin() == 2;
    System.out.println("Test cases are all executed correctly.");
  }
}
```

One way: Use a second Stack called minS (min-Stack). During pushes, if we are pushing a value smaller than (or equal to) current minS.peek(), we push this value to minS too. During pop, if we are popping a value equal to the current minS.peek(), we pop this value from minS too. An example solution can be seen below:

```
import java.util.Stack;

class SpecialStack {
  private Stack<Integer> internalS; // basically a normal stack
  private Stack<Integer> minS; // with this additional bookkeeping

  public SpecialStack() {
    internalS = new Stack<>();
    minS = new Stack<>();
  }

  public int peek() { // return the topmost Integer (-1 if empty)
    int ret = -1;
    if (!internalS.isEmpty())
      ret = internalS.peek(); // O(1)
    return ret;
  }

  public int pop() { // pop+return the topmost Integer (-1 if empty)
    int ret = -1;
    if (!internalS.isEmpty())
      ret = internalS.pop(); // O(1)
    if (ret == minS.peek()) // the same min is about to be popped out, O(1)
      minS.pop(); // pop over there too, O(1)
    return ret;
  }

  public void push(int value) { // put value at the top of Stack
    internalS.push(value); // O(1)
    // if (minS.isEmpty() || (value < minS.peek())) // warning, this is buggy!
    if (minS.isEmpty() || (value <= minS.peek())) // new (OR EQUAL) min, O(1)
      minS.push(value); // push over there too, O(1)
  }

  public int findMin() { // report the min Integer (-1 if empty)
    int ret = -1;
    if (!minS.isEmpty())
```

```
      return ret = minS.peek(); // O(1)
    return ret;
  }
}



// using assertions, so run Java with -ea (enableassertions) flag:
// java -ea SpecialStackTest
class SpecialStackTest {
  public static void main(String[] args) {
    SpecialStack S = new SpecialStack();
    assert S.peek() == -1;
    assert S.findMin() == -1;
    S.push(2); S.push(7); S.push(8); S.push(6);
    S.push(1); S.push(5); S.push(1);
    assert S.peek() == 1;
    assert S.findMin() == 1;
    assert S.pop() == 1; // we pop 1 out and return 1
    assert S.peek() == 5;
    assert S.findMin() == 1; // there were two copies of 1 before
    assert S.pop() == 5; // we pop 5 out and return 5
    assert S.pop() == 1; // we pop (the other) 1 out and return 1
    assert S.peek() == 6;
    assert S.findMin() == 2;
    System.out.println("Test cases are all executed correctly.");
  }
}
```

Alternative way: Just keep the running min in minS that has the same size as internalS. During push, we push value to InternalS and min of (value, minS.peek()).

Also do not forget to do quick time complexity analysis and convince the grader that all methods run in $O(1)$.

C Adding n Integers (24 marks)

You are given an integer n ($2 \leq n \leq 2 \times 10^5$) in the first line that describes an array A.
Then n single-space-separated positive Integers $\in [1..100\,000]$ in the second line.
Notice that the Integers can be repeated.

Your task is to sum these n Integers of A, *using the cheapest cost*, and output this cost.

This time the addition ('+') operation has a cost: the summation of those two to be added (in SGD). For example, to add 1 and 10, we need to pay a cost of 11 SGD.

So, if you are given the following example input 1:

```
3
1 2 4
```

Then, you should output:

```
10
```

Explanation: There are three ways to sum $1 + 2 + 4 = 7$ (all sums to 7, obviously).
Here are the enumeration.

1. Do $2 + 4 = 6$ first with cost = 6, then do $6 + 1 = 7$ with cost 7, we pay $6 + 7 = 13$ SGD.
2. Do $1 + 4 = 5$ first with cost = 5, then do $5 + 2 = 7$ with cost 7, we pay $5 + 7 = 12$ SGD.
3. Do $1 + 2 = 3$ first with cost = 3, then do $3 + 4 = 7$ with cost 7, we pay $3 + 7 = 10$ SGD. This is the cheapest. Output this cost 10 SGD.

And, if you are given this other example input 2:

```
4
2 4 5 5
```

Then, you should output:

```
32
```

C.1 Manual Test Cases ($4 \times 2 = 8$ marks)

You are given four small test cases below.
For each test case, write down the answer (1 mark) and a short explanation (the other 1 mark).

i. 2
```
7 3
```

This is a giveaway 2 marks. There is only one way. $7 + 3 = 10$**, with cost** 10 **too.**

```
10
```

ii. 8
```
1 1 1 1 1 1 1 1
```

This is an easy 2 marks. All same values and n **is a power of two. Like Merge Sort analysis:** $N \log N$

```
24
```

iii. 5
```
32 256 128 64 32
```

If we sort the 7 Integers and do the summations from smallest to largest, it will happen to be correct.

```
960
```

iv. 5
```
5 1 3 2 5
```

35**, not** 45 **(if only sort the Integers in non-decreasing order upfront once).**

```
35
```

C.2 Special Case 1 (3 marks)

n is always a power of two and all Integers in A are ones (1), like in C.1.ii.
Design an $O(\log n)$ solution that only works for this special case 1.

Simply output $n \log_2 n$**. Note that log operation in computer is not really** $O(1)$**, it is** $O(\log n)$**.**

C.3 Special Case 2 (3 marks)

We have extra input constraint, $\forall i \in [1..n-1]$, we have $\sum_{k=0}^{i-1} A[k] \leq A[i]$, like in example input 1. Design an $O(n)$ solution for that only works for this special case 2.

Simply do an $O(n)$ simulation from left to right. The extra input constraint ensures that the resulting running summation (prefix sum from index 0 to $i-1$) is always no larger than the next $A[i]$, making the greedy left-to-right summation works.

C.4 Propose an Algorithm and Analyze It (10 marks)

Solve this problem and analyze the time complexity of your algorithm. You can use pseudo-code.
If your algorithm is **incorrect**, you will get 0 **mark**.
If you leave this box **blank**, you will automatically get a **free** 1 **mark** .
Correct $O(n^3)/O(n^2)/O(n \log n)$ algorithm will get 3/6/10 marks, respectively.

For $O(n \log n)$ solution, one can use a Min PQ simulation. First, enqueue all n Integers into a Min PQ first. $O(n)$ CreateHeap or the $O(n \log n)$ one doesn't matter. At all times, simply sum the smallest two remaining Integers, and re-enqueue the resulting sum back into the PQ. We keep the running cost. We keep doing this until there is only 1 Integer left. We output the final total cost.

For $O(n^2)$ solution, maybe via repeated calls of $O(n)$ find-Min (and find-second-Min) operations.

For $O(n^2 \log n)$ solution, perhaps repeated calls of $O(n \log n)$ sorting algorithms and sum the smallest two every time.

For $O(n^3)$ solution, perhaps repeated calls of $O(n^2)$ sorting algorithms and sum the smallest two every time, but I doubt anyone will answer this.

Note that just sorting the entire n Integers in non-decreasing order and summing the them successively usually does not yield the minimum total cost (see the special case 2).

D The Last Question (5 marks)

To qualify for up to easy 5 marks, you need to **write both full names correctly**.
My CS2040S lecturer is **A/Prof Steven Halim** and Teaching Assistant (TA) is **one of the eight**,

Write a **short** (maybe limit yourself to around 2 minutes to do this and about 3-4 sentences) **but honest (and not anonymous)** feedback on what you have experienced in the first 6 weeks of CS2040S in Semester 1 AY 2023/24 (including Week -02/-01 experience, if any). Feedback that are shared by *majority* (**not a one-off**) and can be easily incorporated to make the next 7 weeks of CS2040S better will be done. Grading scheme: 0-blank, 3-considered trivial feedback but not blank, 5-good and constructive feedback, thanks. (Penalty -1 mark for each wrong name above...).