CSC265 F18: Assignment 1

Due: September 26, by midnight

Guidelines: (read fully!!)

- Your assignment solution must be submitted as a *typed* PDF document. Scanned handwritten solutions, solutions in any other format, or unreadable solutions will **not** be accepted or marked. You are encouraged to learn the LaTeX typesetting system and use it to type your solution. See the course website for LaTeX resources. Solutions typed using LaTeX receive 2 bonus marks.
- Your submission should be no more than 6 pages long, in a single-column US Letter or A4 page format, using at least 9 pt font and 1 inch margins.
- To submit this assignment, use the MarkUs system, at URL `https://markus.teach.cs.toronto.edu/csc265-2018-09`. MarkUs will be active after September 20.
- This is a *group assignment.* This means that you can work on this assignment with *at most one other* student. You are *strongly encouraged* to work with a partner. Both partners in the group should work on and arrive at the solution together. Both partners receive the same mark on this assignment.
- Work on all problems together. For each problem, one of you should write the solution, and one should proof-read and revise it. The first page of your submission must list the *name*, *student ID*, and *UTOR email address* of both group members. It should also list, for each problem, which group member wrote the problem, and which group member proof-read and revised it.
- You **may not** consult any other resources except: your partner; your class notes; your textbook and assigned readings. *Consulting any other resource, or collaborating with students other than your group partner, is a violation of the academic integrity policy!*
- You may use any data structure, algorithm, or theorem previously studied in class, or in one of the prerequisites of this course, by just referring to it, and without describing it. This includes any data structure, algorithm, or theorem we covered in lecture, in a tutorial, or in any of the assigned readings. Be sure to give a *precise reference* for the data structure/algorithm/result you are using.
- Unless stated otherwise, you should justify all your answers using rigorous arguments. Your solution will be marked based both on its completeness and correctness, and also on the clarity and precision of your explanation.

Question 1. (10 marks)

In the following procedure, the input is an array $A[1\,..\,n]$ of arbitrary integers, $A.size$ is a variable containing the size n of the array A (assume that A contains at least $n \geq 2$ elements), and "**return** " means return from the procedure call.

```
WHAT(A)
1  $A[1] = 2$
2  for $i = 2$ to $A.size$
3      $s = 1$
4      for $j = 1$ to $i - 1$
5          $s = s * A[j]$
6      if $A[i] \neq s$
7          return
```

Let $T(n)$ be the worst-case time complexity of executing procedure WHAT on an array A of size $n \geq 2$. Assume that assignments, comparisons and arithmetic operations take a constant amount of time each.

Part a. (5 marks)

State whether $T(n)$ is $O(n^2)$ and justify your answer.

[Solution]

The first line of the procedure runs in constant time. Then the outer loop makes at most $n-1$ iterations, and for each one of them the inner loop makes at most $n-1$ iterations. Therefore, the inner loop makes at most $(n-1)^2$ iterations in total. Since each iteration of the inner loop runs in constant time, the worst-case time complexity is bounded by $O(n^2)$.

Part b. (5 marks)

State whether $T(n)$ is $\Omega(n^2)$ and justify your answer.

[Solution]

We will show that there is an input A on which the inner **if** statement is never triggered. Take $A[1\,..\,n]$ defined by $A[1] = 1$ and $A[i] = 2^{2^{i-2}}$ for $i > 1$. A simple induction argument shows that for each value that i takes in the outer loop, after the inner loop has been completed

$$s = \prod_{j=1}^{i-1} A[j].$$

For $i = 2$, this is $A[1] = A[2] = 2$ and the **if** statement is not triggered. For $i > 2$, we have

$$s = \prod_{j=1}^{i-1} A[j] = 2^{1+\sum_{j=2}^{i-1} 2^{j-2}} = 2^{1+\sum_{j=0}^{i-3} 2^{j}} = 2^{1+2^{i-2}-1},$$

which is equal to $A[i] = 2^{2^{i-2}}$, and, again, the **if** statement is not triggered. Therefore, on this input A, all iterations of the nested loops are executed, and, in particular, the total number of iterations of the inner loop are

$$1 + 2 + \ldots + n - 1 = \frac{n(n-1)}{2} = \Omega(n^2).$$

Since each iteration takes constant time, this is $\Omega(n^2)$ time in total.

Question 2. (28 marks) In this problem, you are given as input a *sorted* array $A[1\,..\,n]$ of *distinct* integers, i.e. you can assume that $A[1] < A[2] < \ldots < A[n]$. Assume that $n > 1$. Your goal is to use a priority queue to efficiently enumerate pairs (i, j), $1 \leq i < j \leq n$, in non-decreasing order of $A[j] - A[i]$.

Part a. (3 marks)

Suppose that the pair (i_1, j_1) is such that $1 \le i_1 < j_1 \le n$ and $A[j_1] - A[i_1]$ is minimized. (If there are several such pairs, pick an arbitrary one.) What are the possible values of $j_1 - i_1$? Justify your answer. Suppose that the pair (i_2, j_2) is such that $1 \le i_2 < j_2 \le n$, (i_2, j_2) is different from (i_1, j_1), and $A[j_2] - A[i_2]$ is minimized. (If there are several such pairs, pick an arbitrary one.) What are the possible values of $j_2 - i_2$?

[Solution]

We must have $j_1 - i_1 = 1$, since if $j_1 - i_1 > 1$, then $i_1 < i_1 + 1 < j_1$, and $A[i_1 + 1] - A[i_1] < A[j_1] - A[i_1]$ because $A[i_1 + 1] < A[j_1]$ by the assumption that the values in A are strictly increasing.

We must have $j_2 - i_2 = 1$ as well. Assume otherwise. Once again we have that $i_2 < i_2 + 1 < j_2$. Observe that at most one of the pairs $(i_2, i_2 + 1)$ or $(i_2 + 1, j_2)$ could be equal to (i_1, j_1). If it is the first one, or neither of them, then $A[j_2] - A[i_2 + 1] < A[j_2] - A[i_2]$ because $A[i_2] < A[i_2 + 1]$, and also $(i_2 + 1, j_2)$ is different from (i_1, j_1), which contradicts the choice of (i_2, j_2). An analogous argument gives a contradiction when $(i_2 + 1, j_2)$ equals (i_1, j_1).

Part b. (5 marks)

Let $P = \{(i, j) : 1 \le i < j \le n\}$. Let $N = \binom{n}{2}$, and let us list all pairs in P as a sequence $\sigma = ((i_1, j_1), (i_2, j_2), \ldots, (i_N, j_N))$ so that $A[j_{k+1}] - A[i_{k+1}] \ge A[j_k] - A[i_k]$ for any $1 \le k \le N - 1$. Show that for any k such that $1 \le k \le N$, either $j_k - i_k = 1$ or the pairs $(i_k, i_k + 1)$ and $(i_k + 1, j_k)$ appear before (i_k, j_k) in the sequence σ.

[Solution]

The argument is similar to the arguments in the previous subproblem. We already established that if $1 \le k \le 2$, then $j_k - i_k = 1$, so the claim is true. Assume then that $3 \le k \le N$. If $j_k - i_k = 1$ there is nothing to prove, so let $j_k - i_k > 1$. Take $\ell = i_k + 1$: we have $i_k < \ell < j_k$, so $A[i_k] < A[\ell] < A[j_k]$. It follows that

$$A[\ell] - A[i_k] < A[j_k] - A[i_k],$$
$$A[j_k] - A[\ell] < A[j_k] - A[i_k],$$

so both (i_k, ℓ) and (ℓ, j_k) must come before (i_k, j_k) in σ.

Part c. (20 marks) Design a data structure that supports the following two operations:

- PREPROCESS(A) initializes the data structure using the values in the array $A[1\,..\,n]$. This operation should run in worst-case time $O(n)$.
- NEXTPAIR returns the next pair of integers (i, j), $1 \le i < j \le n$ in non-decreasing order of $A[j] - A[i]$. The first time this operation is called it should return a pair that minimizes $A[j] - A[i]$. In every next call it should return a pair that has not been returned before and minimizes $A[j] - A[i]$ among all such pairs. Every call to this operation should run in worst-case time $O(\log n)$.

You can assume that PREPROCESS(A) is called once, before any call to NEXTPAIR. You can assume that at most $\binom{n}{2}$ calls to NEXTPAIR are made.

Describe how you implement the data structure. Describe how the operations PREPROCESS(A) and NEXTPAIR are implemented as algorithms, both using clear and precise English, and also using pseudocode. Justify why your algorithms correctly implement the operations and why they run in the required worst-case time complexity.

[Solution]

In PREPROCESS(A), we create a new array $H[1\,..\,n-1]$ whose elements the pairs $(i, i+1)$, $1 \le i \le n-1$, with keys $A[i+1] - A[i]$. We then run BUILDMINHEAP(H) to turn H into a min-heap in time $O(n)$.

In NEXTPAIR, we first call EXTRACT-MIN on H to remove the interval (i, j) with the next smallest $A[j] - A[i]$. Then, if $i > 1$, we add the element $(i-1, j)$ with key $A[j] - A[i-1]$ to the heap.

The procedure PREPROCESS(A) runs in time $O(n)$ because constructing H takes time $O(n)$ and BUILDMINHEAP(H) runs in time $O(n)$ as we saw in a tutorial.

To analyze the worst-case running time of NEXTPAIR notice first that H never has more than $n-1$ elements, since it is initialized to $n-1$ elements, and at every call of NEXTPAIR one element is removed and one element is added. The running time of NEXTPAIR is dominated by the calls to EXTRACT-MIN and INSERT, and both take time $O(\log n)$ because H has size $n-1 \leq n$.

For correctness, let $\sigma = ((i_1, j_1), (i_2, j_2), \ldots, (i_N, j_N))$ be the sequence of pairs returned by $N = \binom{n}{2}$ consecutive calls to NEXTPAIR. We need to show that these pairs are distinct and that for any $1 \leq k \leq n$, (i_k, j_k) minimizes $A[j_k] - A[i_k]$ among all pairs not in $((i_1, j_1), \ldots, (i_{k-1}, j_{k-1}))$

To prove distinctness, we claim that every pair (i, j) is inserted at most once into the heap H. This is obvious when $j - i = 1$ since every pair we insert in NEXTPAIR has $j - i > 1$. Towards contradiction, assume then that (i, j) is the first pair that's inserted a second time to H. This pair is inserted when $(i+1, j)$ is extracted from the heap, so $(i+1, j)$ must've been extracted twice, which can only happen if it was already inserted twice in the past. This contradicts the assumption that (i, j) was the first pair inserted twice into H.

It remains to prove the minimality of (i_k, j_k). Assume for contradiction that k is the first step when $A[j_k] - A[i_k]$ is not minimal. This cannot be $k = 1$ because on the first call of NEXTPAIR we return the pair $(i, i+1)$ that minimizes $A[i+1] - A[i]$, and we know from the first subproblem that $j_1 - i_1 = 1$. Let us then assume that $k > 1$ and consider the k-th call of NEXTPAIR. Let (i, j) be the pair that minimizes $A[j] - A[i]$ among all pairs *not* in $\sigma_{k-1} = ((i_1, j_1), \ldots, (i_{k-1}, j_{k-1}))$. If $j - i = 1$, then this pair was inserted in the heap during PREPROCESS, and has not been extracted because it is not in σ_{k-1}. Therefore, during the k-th call of NEXTPAIR the pair (i, j) was in the heap, and the pair (i_k, j_k) could not have been returned, a contradiction. Assume finally that $j - i > 1$. Then by the second subproblem, the intervals $(i, i+1)$ and $(i+1, j)$ must be in σ_{k-1}. When $(i+1, j)$ was extracted, the interval (i, j) was inserted in the heap, and has not been extracted yet, since it is not in σ_{k-1}. Therefore, we conclude again that during the k-th call of NEXTPAIR the pair (i, j) was in the heap, and the pair (i_k, j_k) could not have been returned, a contradiction. This completes the proof.