Sample Solution for Problem Set 1

Data Structures and Algorithms, Fall 2021

September 15, 2021

Contents

1 Problem 1 **2**

2 Problem 2 **3**

3 Problem 3 **4**

4 Problem 4 **5**

5 Problem 5 **6**

6 Problem 6 **7**

7 Bonus Problem **8**

1 Problem 1

(a) We need to prove that A' consists of the elements of A . In other words, A' and A are permutation of each other.

(b) The loop invariant for the **for** loop in lines 2–4 can be stated as follows: at the start of the $(n-j+1)^{th}$ iteration, the subarray $A[j,...,n]$ consists of the elements originally in $A[j,...,n]$ and the smallest element in subarray $A[j,...,n]$ is the first element $A[j]$. Proof that this loop invariant holds via mathematical induction:

- (basis) At the start of the first iteration, $A[n]$ is the only element in the subarray, which is also the smallest.
- (inductive step) Assume at the start of the $(n-j+1)^{th}$ iteration, the subarray $A[j,...,n]$ consists of the elements originally in $A[j,...,n]$ and the smallest element in subarray $A[j,...,n]$ is the first element $A[j]$. During the $(n-j+1)^{th}$ iteration, we compare $A[j]$ with $A[j-1]$ and swap them if $A[j]$ is the smaller one. Thus, at the start of the $(n-j+2)^{th}$ iteration, the smallest element in subarray $A[j-1,...,n]$ is the first element $A[j-1]$.

(c) The loop invariant for the **for** loop in lines 1–4 can be stated as follows: by the end of the i^{th} iteration, the elements in subarray $A[1,...,i]$ are i smallest elements in array $A[1,..,n]$ and they are in sorted order. Proof that this loop invariant holds via mathematical induction:

- (basis) By the end of the first iteration, according to part (b), $A[1]$ will be the smallest elements in array $A[1,...,n]$ after the inner loop.
- (inductive step) Assume by the end of the i^{th} iteration, the elements in subarray $A[1,...,i]$ are i smallest elements in array $A[1,...,n]$ in sorted order. During the $(i+1)^{th}$ iteration, according to part (b), $A[i+1]$ will be the smallest element of the subarray $A[i+1,...,n]$ after the execution of the inner loop. We have $A[i] \leq A[i+1]$ holds and $A[i+1]$ is the $(i+1)^{th}$ smallest elements in array $A[1,...,n]$. Thus, by the end of the $(i+1)^{th}$ iteration, the elements in subarray $A[1,...,i+1]$ are $i+1$ smallest elements in array $A[1,...,n]$ in sorted order.

We achieve an output A' by the end of the $(n-1)^{th}$ iteration. According to the proof above, the elements in subarray $A'[1,...,n-1]$ are $(n-1)$ smallest elements in array $A'[1,...,n]$ in sorted order. We have $A'[1] \leq A[2]' \leq ... \leq A'[n-1]$ and $A'[n-1] \leq A'[n]$ holds trivially.

2 Problem 2

(a) Let c_i $T(n) = c_1 + c_2(n+2) + c_3(n+1) = (c_2 + c_3)n + (c_1 + 2c_2 + c_3) = \Theta(n)$.

(b)

- Loop Invariant: At the beginning of each iteration of the for loop, which is indexed by i, $y = \sum_{j=i+1}^{n} c_j x^{j-(i+1)}$.
- Proof:
  - Initialization:
$$y = 0 = \sum_{j=n+1}^{n} c_j x^{j-(i+1)}$$
    , when $i = n$.
  - Maintain:
$$\begin{aligned} y_{new} &= c_i + x \times y_{old} \\ &= c_i + x \times \sum_{j=i+1}^{n} c_j x^{j-(i+1)} \\ &= c_i + \sum_{j=i+1}^{n} c_j x^{j-i} \\ &= \sum_{j=i}^{n} c_j x^{j-i} \end{aligned}$$
    ,
  - Termination: $y = \sum_{j=0}^{n} c_j x^j$, At the end of the for loop
- Correctness: $y = \sum_{j=0}^{n} c_j x^j$ is equal to $P(x)$.

3 Problem 3

- $f \in O(g)$: (b), (l), (m), (p)
- $f \in \Omega(g)$: (g), (h), (i), (j), (k), (o)
- $f \in \Theta(g)$: (a), (c), (d), (e), (f), (n)

4 Problem 4

$$
\begin{aligned}
1 = n^{1/\lg n} &\ll \\
\lg(\lg^* n) &\ll \\
\lg^* n = \lg^*(\lg n) &\ll \\
2^{\lg^* n} &\ll \\
\ln \ln n &\ll \\
\sqrt{\lg n} &\ll \\
\ln n &\ll \\
\lg^2 n &\ll \\
2^{\sqrt{2 \lg n}} &\ll \\
(\sqrt{2})^{\lg n} &\ll \\
n = 2^{\lg n} &\ll \\
n \lg n = \lg(n!) &\ll \\
n^2 = 4^{\lg n} &\ll \\
n^3 &\ll \\
(\lg n)! &\ll \\
(\lg n)^{\lg n} = n^{\lg \lg n} &\ll \\
(3/2)^n &\ll \\
2^n &\ll \\
n \cdot 2^n &\ll \\
e^n &\ll \\
n! &\ll \\
(n+1)! &\ll \\
2^{2^n} &\ll \\
2^{2^{n+1}}
\end{aligned}
\tag{1}
$$

5 Problem 5

We can maintain two pointers, $p1$ and $p2$, to the tops of two stacks. At beginning, $p1$ is 0, and $p2$ is $n + 1$.

Algorithm

- Push of stack1(x) : $p1 \leftarrow p1 + 1, A[p1] \leftarrow x$
- Push of stack2(x) : $p2 \leftarrow p2 - 1, A[p2] \leftarrow x$
- Pop of stack1 :$p1 \leftarrow p1 - 1$
- Pop of stack1 :$p2 \leftarrow p2 + 1$

Correctness

We can get the size of two stacks are $p1$ and $n + 1 - p2$, and overflow occurs only when the total size of both stacks exceed n, equivalent to $p2 <= p1$. Otherwise $p2 > p1$, and the two stacks do not overlap to make errors.

Time Complexity

Time complexity of every operation is $\Theta(1)$.

6 Problem 6

We may assume that queries are valid. Let P, Q be two FIFO queues.

Algorithm

- Push(x): Enqueue x to FIFO queue P.
- Pop(): Repeat dequeue an element in queue P and Enqueue it to queue Q, until the size of queue P is 1. Dequeue the last element in P and stored it as top. Repeat dequeue an element from queue Q and Enqueue it to queue P, until queue Q is empty. Return top.

Correctness

We will show that following condition holds after each type of query.

- FIFO queue Q is empty. Element x arrives earlier than element y iff x is in front of y in FIFO queue P.

Base case is obvious. For every push operation, the above condition trivially holds, as we have the newest element placed at the end of the queue P. For every pop operation, we return the tail element of queue P, which is also the newest pushed element that satisfied stack's property. Here, enqueue and dequeue operations do not change the relative location of any two remaining elements in queue P after push operation, compared with the original queue P.

Pseudocode

```
procedure push(x):
    P.enqueue(x)

function pop():
    while P.size > 1:
        Q.enqueue(P.dequeue())
    top = P.dequeue()
    while !Q.empty:
        P.enqueue(Q.dequeue())
    return top
```

Time Complexity

- Pop function: $\Theta(1)$, as we only do a push operation.
- Push function: $\Theta(n)$, where n is the number of remaining elements in FIFO queue P at the beginning of the push function.

Remark

- Algorithm will be scored according to the overview of the algorithm, pseudocode, and the analysis of the time complexity.
- Please giving the definition of n, if you use it as a factor while analyzing time complexity.

7 Bonus Problem

Algorithm

The data structure contains an array $A[]$ and an integer $size$. $A[]$ is initialized to empty and $size$ is initialized to 0.

Algorithm 1 add(x)

$A[size] \leftarrow x$
$size \leftarrow size + 1$

Algorithm 2 remove()

$i \leftarrow random(size) - 1$
$size \leftarrow size - 1$
swap($A[i]$,$A[size]$)
return $A[size]$

Correctness

We claim that the data structure maintains the following invariance:

invariance:$\{A[0], A[1], ..., A[size - 1]\}$ are all the elements in the queue.

initialization: $size = 0$ and $queue$ is empty, which is true.

maintaining: After function add, we plus $size$ by 1. Since $\{A[0], ..., A[size - 2]\}$ are elements before we add x to queue, and we set $A[size - 1] = x$, $\{A[0], ..., A[size - 1]\}$ are elements in queue. After function $remove$, the $\{A[0], ..., A[size - 1]\}$ are exactly all the elements in queue except $A[i]$.

To prove that we always return a uniform random element in $remove$ procedure, notice that $A[size]$ is equal to $A[i]$ in the queue, and each element in the queue has the same probability to be $A[i]$ according to the property of $random$.

Complexity

It is trivially $O(1)$ for both add and $remove$.