Problem Set 1

Data Structures and Algorithms, Fall 2021

Due: September 9 23:59:59 (UTC+8), mail to `dsalg21ps@chaodong.me`.

Problem 1

`BubbleSort` is a popular, but inefficient, sorting algorithm. It works by repeatedly swapping adjacent elements that are out of order.

`BubbleSort`$(A[1 \cdots n])$

1: **for** $(i \leftarrow 1$ **to** $n-1)$ **do**
2:     **for** $(j \leftarrow n$ **downto** $i+1)$ **do**
3:         **if** $(A[j] < A[j-1])$ **then**
4:             Swap $A[j]$ and $A[j-1]$

(a) Let A' denote the output of `BubbleSort`(A). To prove that `BubbleSort` is correct, we need to prove that it terminates and that

$$A'[1] \leq A'[2] \leq \cdots \leq A'[n] \tag{1}$$

However, in order to show that `BubbleSort` actually sorts, what else do we need to prove?

(b) State precisely a loop invariant for the **for** loop in lines 2–4, and prove that this loop invariant holds.

(c) Using the loop invariant proved in part (b), state and prove a loop invariant for the **for** loop in lines 1–4. Finally, using this newly proved loop invariant to prove inequality (1).

Problem 2

Given $c_0, c_1, \cdots, c_n$ and a value for x, we can evaluate a polynomial

$$P(x) = \sum_{k=0}^{n} c_k x^k = c_0 + x(c_1 + x(c_2 + \cdots + x(c_{n-1} + x c_n) \cdots))$$

using the following algorithm:

`PolyEval`$(x, c_0, c_1, \cdots, c_n)$

$y \leftarrow 0$
for $(i = n$ **downto** $0)$ **do**
    $y \leftarrow c_i + xy$

(a) Give the runtime of `PolyEval` in Θ-notation.

(b) What loop invariant does `PolyEval` maintain? State it, prove it, and then use it to show the correctness of `PolyEval`.

Problem 3

In each of the following situations, indicate whether $f \in O(g)$, or $f \in \Omega(g)$, or both (in which case $f \in \Theta(g)$). You do *not* need to prove your answer.[1]

	$f(n)$	$g(n)$
(a)	$n - 100$	$n + 200$
(b)	$n^{1/2}$	$n^{2/3}$
(c)	$100n + \lg n$	$n + (\lg n)^2$
(d)	$n \lg n$	$10n \lg (10n)$
(e)	$\lg (2n)$	$\ln (3n)$
(f)	$10 \lg n$	$\lg (n^2)$
(g)	$n^{1.1}$	$n \lg^2 n$
(h)	n^2/lgn	$n(\lg n)^2$
(i)	$n^{0.1}$	$(\lg n)^2$
(j)	$(\lg n)^{\lg n}$	$n/ \lg n$
(k)	$\sqrt{n}$	$(\lg n)^3$
(l)	$n^{0.4}$	$5^{\ln n}$
(m)	$n2^n$	3^n
(n)	2^n	2^{n+1}
(o)	$n!$	2^n
(p)	$(\lg n)^{\lg n}$	$2^{(\lg n)^2}$

Problem 4

Sort the following functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. You do *not* need to prove your answer. To simplify notation, write $f(n) \ll g(n)$ to denote $f(n) \in o(g(n))$ and $f(n) = g(n)$ to denote $f(n) \in \Theta(g(n))$.[2]

$\lg(\lg^* n)$	$2^{\lg^* n}$	$(\sqrt{2})^{\lg n}$	n^2	$n!$	$(\lg n)!$
$(3/2)^n$	n^3	$\lg^2 n$	$\lg(n!)$	2^{2^n}	$n^{1/\lg n}$
$\ln \ln n$	$\lg^* n$	$n \cdot 2^n$	$n^{\lg \lg n}$	$\ln n$	1
$2^{\lg n}$	$(\lg n)^{\lg n}$	e^n	$4^{\lg n}$	$(n+1)!$	$\sqrt{\lg n}$
$\lg^*(\lg n)$	$2^{\sqrt{2 \lg n}}$	n	2^n	$n \lg n$	$2^{2^{n+1}}$

Problem 5

Explain how to implement two stacks in one array $A[1 \cdots n]$ in such a way that neither stack overflows unless the total number of elements in both stacks together is n. You should give a brief overview of your implementation, then provide pseudocode for the `push` and `pop` operations of each of the two stacks. In your implementation, `push` and `pop` operations should run in $O(1)$ time.

[1] Throughout this course, $\lg(n) = \log_2(n)$.

[2] Read Section 3.2 of CLRS for the definition of function $\lg^* n$.

Problem 6

Explain how to implement a stack using two FIFO queues. You should give a brief overview of your implementation, then provide pseudocode for `push` and `pop` operations. Assuming `enqueue` and `dequeue` each takes $\Theta(1)$ time, analyze the running time of `push` and `pop`.

Bonus Problem

*(This is a **bonus** problem, which means you are not required to solve it. However, correct solving it will award you bonus credit.)*

Design a RANDOMQUEUE data structure. This is an implementation of the QUEUE interface in which the `remove` operation removes an element that is chosen uniformly at random among all the elements currently in the queue. You may assume you have access to a `random` function which takes a positive integer as input: `random`(x) returns a uniformly chosen random integer in $[1, x]$, in $O(1)$ time. You should give a brief overview of your data structure, then provide pseudocode for `add` and `remove` operations. You should also discuss the time complexity of `add` and `remove` operations. (To get full credit, `add` and `remove` operations should run in $O(1)$ time in your implementation, you also need to argue the correctness of your implementation. You may assume the number of elements in the queue never exceeds N, and the value of N is known in advance.)