SE 2C03 - Assignment 1 solution suggestions

1.3.45 Go over the operations stream and keep a counter c of *pops* (initially 0), as well as the largest element x that has been *pushed* so far. If at any point $c > x$, then report an underflow.

1.3.46 Look into Robert Tarjan's notes at
`https://www.cs.princeton.edu/courses/archive/fall10/cos226/precepts/14StacksAndQueues-Tarjan.pdf`

1.3.48 Use each end of the deque as a stack.

1.4.5 $N, 1, 1, 2N^3, 1$ (Note $lg(2N) = lg2 + lgN$), 2 (Note $lg(N^2+1) = 2lgN + lg(1+\frac{1}{N^2})$), $\frac{N^{100}}{2^N}$ (Note that the given function is *decreasing* on N, so it goes to 0 as $N \to \infty$; we divide it with itself to achieve a limit 1)

1.4.6

1. Note that $n = N, N/2, N/4, N/8, \ldots, 1$, hence the body of the inner loop will be executed at most
$$\sum_{j=0} lgNN/2^j = N\frac{1/2^{lgN+1} - 1}{1/2 - 1} = O(N)$$
2. Note that the times the inner loop is executed are the same as in the previous question (only in reverse order).
3. The inner loop has always N iterations, and the outer loop is executed lgN times, hence the order of growth is $O(NlgN)$.

1.4.9 We can predict that the running time $\sim aN^b$, where we can set $a := T/{N_0}^b$.

1.4.12 I'm not going to give an implementation but the idea is the following: Suppose the arrays are A, B, each of size N.. Keep pointers i, j for each array (initialized to 0). If $A[i] \leq B[j]$ then print $A[i]$ and increase i, otherwise print $B[j]$ and increase j; if you run out of A or B then print the remaining B or A respectively. Note that with each comparison (and print) a new array element (from either A or B) must be considered; therefore you cannot have more than about $2N$ comparisons and prints, and your running time is of $O(N)$ (linear) order-of-growth.

1.4.15 To solve the 2-sum problem for two sorted arrays A, B, keep a pointer i (initially 0) for A, and a pointer j (initially $N-1$) for B. If $A[i] = -B[j]$ then increase the counter of pairs. If not, then if $|A[i]| > B[j]$ then increase i, else ($|A[i]| < B[j]$) then decrease j. For the 3-sum problem, go over all elements $C[k]$ of the third array one-by-one, and subtract it from all elements of A (linear time) to produce a new array A'; then solve the 2-sum problem for A', B (again linear time). So you repeat this linear time process for all N elements of C, for a quadratic ($O(N^2)$) total time.

1.4.17 The biggest difference is $\max - \min$, and each of $\min, \max$ of the array can be found in linear time.

1.4.20 Find the max element in $\sim lgN$ time, and then do binary search in subarrays $A[0..max]$, $A[max..N]$ ($\sim lgN$ each).

1.4.25 Break interval $[1 \dots N]$ into $\sqrt{N}$ pieces, each of size $\sqrt{N}$. Then use one egg to find the interval that contains F ($\sqrt{N}$ tries), and the other egg to search all numbers in that interval ($\sqrt{N}$ tries). To decrease the number of throws, start throwing one egg at geometrically increasing numbers $(2^0, 2^2, 2^4, \dots, 2^{2i}, \dots, 2^{2x})$, until $lgF/2 \leq x < lgF/2 + 1$. Then F is in the last interval, and that interval has at most $2^{2x} - 2^{2(x-1)} = 3 \cdot 2^{2(x-1)} \leq 3F$ numbers (with F amongst them), and therefore we can break it into $c\sqrt{F}$ chunks (where $c \leq \sqrt{3}$ is a constant) which can be explored with the second egg (like before).

1.4.34 Start with testing $N/4$, and then $3N/4$. If you get "hotter" then the number is in $[N/2, N]$ else the number is in $[1, N/2]$. Your interval length has been halved. Next you test $5N/8$ and (possibly) also $7N/8$, or $N/8$ and (possibly) also $3N/8$ to halve again the search interval. So, the search interval is reduced in size just like binary search, but you may have to ask two questions (in the worst case) in each interval, instead of the one question (comparison) you ask in binary search. Hence your running time is $\sim 2lgN$ (instead of the binary search time $\sim lgN$). To reduce the number of questions, you have to be able to halve the interval with a single question every time. Start with testing $-N/2$. Then test $3N/2$. You now know whether the number is in $[1, N/2]$ or $[N/2, N]$. Suppose it is in $[1, N/2]$; then test $-N$. If "colder" then you know that it is in $[1, N/4]$, else it is in $[N/4, N/2]$. Continue in this fashion, so that the next number you test together with the last one halve the current interval. Notice that in this way, every new test (except the very first one) halves the interval, and the running time is the same as binary search, i.e., $\sim lgN$.

1.5.14 Similar to Proposition H. Suppose that for all tree sizes $k \leq n$ the height is at most lgk. Now suppose that a tree T of size $n + 1 = k_1 + k_2$ and height h is built from two trees T_1, T_2 of sizes $k_1 \leq k_2$ and heights $h_1 \leq lgk_1,\ h_2 \leq lgk_2$. Then we have three cases:

- $h_1 < h_2$: Then $h = h_2 \leq lgk_2 \leq lg(n+1)$
- $h_2 < h_1$: Then $h = h_1 \leq lgk_1 \leq lg(n+1)$
- $h_1 = h_2$: Then $h = h_1 + 1 \leq lgk_1 + 1 \leq lg(2k_1) \leq lg(n+1)$

1.5.15 The worst case happens when connecting the smaller tree to the larger tree increases the depth (by 1), and the two trees we are connecting are worst-case trees in their own right. Therefore, in order to make the smaller tree as "deep" as possible (i.e. as deep as the larger tree), we give it as many nodes as the large tree. This intuition leads us to defining the following family of trees, called *binomial trees*. We define these trees recursively:

- The order 0 binomial tree B_0 is just a node.
- The order k binomial tree B_k is constructed by connecting two B_{k-1} trees so that one is connected to the root of the other.

Note that by construction the depth of a B_k tree is exactly k. Several nice properties can be shown for these trees. Simple induction on k shows that B_k has 2^k nodes. Also, simple induction on k shows that at level i there are $\binom{k}{i}$ nodes: It is true for $k = 0$ (just the root), we assume that, for B_{k-1}, every level $i = 0, 1, \dots, k-1$ has $\binom{k-1}{i}$ nodes, and then each level i in B_k has $\binom{k-1}{i} + \binom{k-1}{i+1} = \binom{k}{i}$ (by construction and the inductive hypothesis). For the second part of the question, the average depth for B_n (i.e., $N = 2^n$), is

$$\frac{\text{collective depth}}{N} = \frac{\sum_{i=0}^{n} \binom{n}{i} i}{N} = \frac{n2^n}{2^n} = n = \lg N.$$