CPSC 320: Intermediate Algorithm Design and Analysis
Assignment #5, due Friday, June 10th, 2016 at 2:15pm in Room x235, Box 32

[6] 1. Let $A[1 \ldots n]$ be an array of n distinct integers. A set $S \subseteq \{1, \ldots, n\}$ is called an *ultra-decreasing subsequence* if

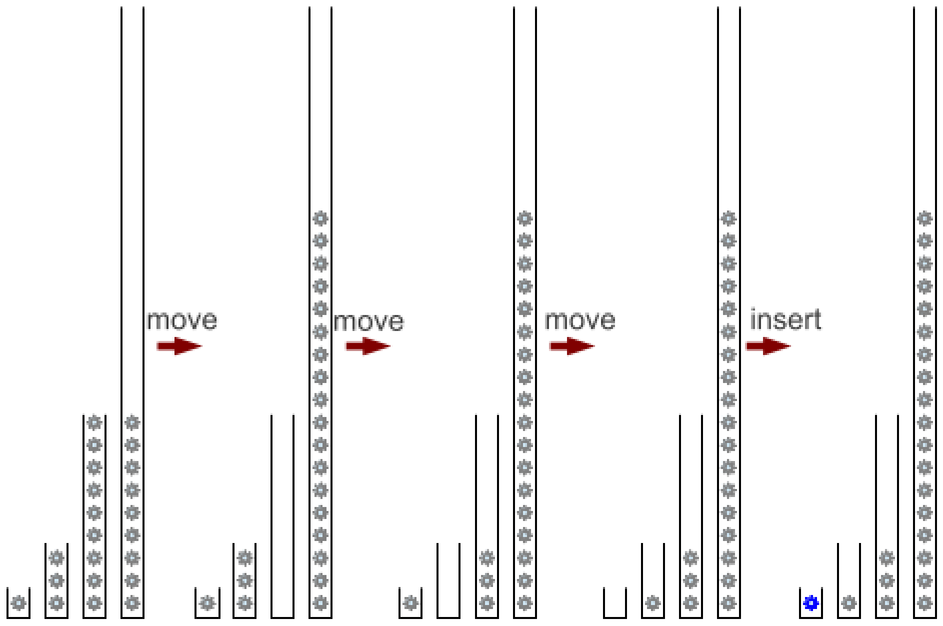
$$A[j] \geq A[i] \quad \text{for all } j \in S \text{ and all } i \in \{1, \ldots, n\} \text{ with } i > j.$$

Here is an algorithm that computes an ultra-decreasing subsequence. The function Top(S) returns the top element of the stack without removing it from the stack.

```
Algorithm: UltraDecrSubseq
let $S$ be an empty stack
for $i$ from 1 to $n$ do
    while $S$ is not empty and $A[i] > A[\text{Top}(S)]$ do
        Pop($S$)
    end
    Push($S$, $i$)
end
Return $S$
```

Use amortized analysis to analyze the runtime of `UltraDecrSubseq` as a function of n. **Hint:** the runtime is $o(n^2)$.

[10] 2. A multistack consists of a (potentially infinite) series of stacks $S_0, \ldots, S_{t-1}$ where the j^{th} stack S_j can hold up to 3^j elements. Whenever a user attempts to push an element onto a full stack S_j, we first pop all the elements off S_j and push them onto stack S_{j+1} to make room. Thus, if S_{j+1} is already full, we first recursively move all its members to S_{j+2}. Moving a single element from one stack to the next takes $O(1)$ time. For instance, here is how we would insert a new element into a multistack that already contains 22 elements:



[2] a. In the worst case, how long does it take to push one more element onto a multistack containing n elements?

[8] b. Prove that the amortized cost of a sequence of n push operation on an initially empty multistack is in $O(n \log n)$, where n is the maximum number of elements in the multistack. Hints:

- Use $\Phi(D_i) = 2\sum_{j=0}^{t-1} \texttt{Nelems}_i(S_j)(\log_3 n - j)$ where $\texttt{Nelems}_i(S_j)$ is the number of elements stack S_j contains in D_i.
- Start by computing the amortized cost of moving the elements of a full stack onto the next stack.

[10] 3. A problem with Bloom filters is that you cannot delete elements from a Bloom filter by changing the corresponding 1s to 0s. Counting Bloom filters are a variation of Bloom filters where you do not use an array of bits, but an array of small counters. Instead of changing 0s to 1s when an element hashes to a Bloom filter location, you increment the counter. To delete an element, you decrement the counter. To check if an element is in the set, you just check if all of the counter entries are bigger than 0; if they are, then you return the element is in the set. If you see a 0, you say that the element is not in the set.

[2] a. Explain, by giving an example, why deletions will not work properly with a standard Bloom filter if you change 1s back to 0s on a deletion

[6] b. Deletions will work properly for a Counting Bloom filter as long as the counters never overflow; once a counter overflows, you would not have an accurate count of the number of elements currently hashed to that location. A standard choice in this setting in practice to use 4 bit counters. Find an expression for the probability that a specific counter overflows when using 4 bit counters when n elements are hashed into m total locations using k hash functions. (You may have a summation in your expression.)

[2] c. Suppose that you use m counters for n elements and k hash functions for a Counting Bloom filter. What is the false positive probability (assuming there has never been an overflow), and how does it compare with the standard Bloom filter?

[5] d. **OPTIONAL BONUS:**

Let us consider again the overflow probability computed in part (b) of this question. Suppose we fix $k = (m/n) \cdot \ln(2)$, a standard number of hash functions for a Bloom filter. In this question, we will compute an actual numerical upper bound on the overflow probability.

To do so, you should use a theorem called the Chernoff bound. This theorem states that, for a random variable X with the binomial distribution with parameters n and p (hence with mean $\mu = np$), for any $\delta > 0$

$$\Pr[X \geq (1+\delta)\mu] \;\leq\; \frac{e^{\delta}}{(1+\delta)^{1+\delta}}.$$

[1] 4. (Bonus) How long did it take you to complete this assignment (not including any time you spent revising your notes before starting)?