Your name
Your student number

CS 466/666: ALGORITHM DESIGN AND ANALYSIS
PROBLEM SET 5

DUE ON: NOVEMBER 15, 2017

Acknowledgements.

Acknowledge **all collaborators** with whom you discussed any of the problems in this problem set and **all external sources** that you may have used in the completion of the assignment. See the website for the full homework policy.

1. Problem 1: Analysis of splay trees

Recall that the *rank* of a node x in a tree T as $\text{rank}_T(x) = \log_2 N_T(x)$ with $N_T(x)$ being the number of nodes in the subtree of T rooted at x and we define the *potential* of a tree T to be $\Phi(T) = \sum_{x \in T} \text{rank}_T(x)$. In class, we showed that the amortized time complexity of the `Zig` and `ZigZig` operations are bounded as follows. (Throughout this question, T is the tree before the operation is called, and T' is the resulting tree afterwards.)

Lemma 1. $\text{AC}(\texttt{Zig}(x)) \le 1 + (\text{rank}_{T'}(x) - \text{rank}_T(x))$.

Lemma 2. $\text{AC}(\texttt{ZigZig}(x)) \le 3(\text{rank}_{T'}(x) - \text{rank}_T(x))$.

(a) Show that the amortized cost of `ZigZag` operation can also be bounded similarly.

Lemma 3. $\text{AC}(\texttt{ZigZag}(x)) \le 2(\text{rank}_{T'}(x) - \text{rank}_T(x))$.

Proof. Enter your answer here. □

(b) Use the results above to bound the amortized cost of `Splay`.

Lemma 4. $\text{AC}(\texttt{Splay}(x)) \le 1 + 3(\text{rank}_{T'}(x) - \text{rank}_T(x))$.

Proof. Enter your answer here. □

(c) Conclude by showing that `Query` has amortized time complexity that is only logarithmic in n.

Theorem 1. *When a Splay Tree has at most n nodes, then* $\text{AC}(\texttt{Query}(x)) \le O(\log n)$.

Proof. Enter your answer here. □

2. Problem 2: Zigging without zagging

The splay trees we saw in class have amortized time complexity $O(\log n)$, but they also require a somewhat complicated-looking splay operation that uses zig, zig-zig, and zig-zag operations. Are those really necessary?

Let a *zig tree* be a simpler type of binary search trees that behaves exactly like splay trees, except that the `Splay` operation is replaced with a simpler `Iterated-Zig` operation that only invokes `Zig` (i.e., simple rotations) until the target node reaches the root of the tree.

(a) Show that zig trees do not have amortized time complexity $O(\log n)$.

Theorem 2. *There is a sequence of n `Insert` and `Query` operations for which zig trees have amortized time complexity $\Omega(n)$.*

Proof. Enter your answer here. □

(b) **Bonus.** Prove an even stronger separation between the two types of BSTs: there are sequences of operations for which zig trees have amortized time complexity $\Omega(n)$ and splay trees have amortized time complexity $O(1)$.

3. Problem 3: Counting

We often think of counters as objects that we can increment (or generally modify) in constant time. But in this question, we take a closer look at this assumption. Define `counter` to be a bit array of length n initialized to 0, and define the INCREMENT operation as follows.

Algorithm 1: INCREMENT

```
$i \leftarrow 1$;
while counter[$i$] $== 1$ do
    counter[$i$] $\leftarrow 0$;
    $i \leftarrow i + 1$;
counter[$i$] $\leftarrow 1$;
```

In this question, the time complexity of the INCREMENT operation is measured in terms of the number of bits of `counter` that are read or updated.

(a) Show that INCREMENT does not have constant worst-case time complexity.

Theorem 3. *The* `Increment` *operation has worst-case time complexity* $\Omega(n)$.

Proof. Enter your answer here. ☐

(b) Show that the amortized complexity of `Increment` operations is constant.

Theorem 4. *For any* $m \leq 2^n$*, the total time complexity of* m *calls to* `Increment` *on a* `counter` *that was initially* 0 *is* $O(m)$.

Proof. Enter your answer here. ☐

Hint. Your first task should be to determine what the right potential function should be for this problem.

(c) **Bonus.** Can you also implement an *up-down counter* for which any sequence of INCREMENT and DECREMENT operations has amortized time complexity $O(1)$?

4. Problem 4: Brainless paging

Recall that in the paging problem, an algorithm maintains a cache that contains at most k pages and receives a stream $x_1, \ldots, x_n$ of n requests for pages. Whenever a page is not currently in the cache, the algorithm incurs a *page fault*, where it must evict a page from the cache and loads the requested page (from memory) into the freed cache space.

Consider the Brainless paging algorithm that, on page faults, evicts a page that is chosen uniformly at random from the cache. You will show that this algorithm has the same competitive ratio as the (optimal deterministic) algorithm LRU(1) that we saw in class.

Definition 1. The *competitive ratio* of a randomized paging algorithm A is

$$CR(A) = \max_{x=(x_1,\ldots,x_n)} \frac{\mathrm{E}[\mathrm{Cost}_A(x)]}{\mathrm{Cost}_{OPT}(x)}$$

where the expectation is over the internal randomness of A, the cost of an algorithm is the number of page faults incurred on the sequence of page requests, and OPT is the optimal algorithm (with hindsight).

(a) Show that Brainless has competitive ratio $O(k)$.

Theorem 5. *CR(Brainless) $= O(k)$.*

Proof. Enter your answer here. □

Hint. Your first task should again be to determine what the right potential function should be for this problem. It might be helpful to start by considering the elements that are in the Brainless cache but not in the OPT cache at a given time step.

(b) **Bonus.** Show that there is a randomized paging algorithm A with competitive ratio $CR(A) = O(\log k)$. (*Extra bonus:* And show that this ratio is optimal for all randomized paging algorithms.)

5. Problem 5: Nets and approximations

Let $X \subseteq \mathbb{R}^2$ be a set of n points on the plane, and let $\mathcal{C}$ be the family of all circles on the plane. In this question, you will explore two subset sampling questions regarding circles.

(a) A circle C is *minimal for* X if there is no $C' \subseteq C$ for which $C' \cap X = C \cap X$. Show that there are not too many circles that are minimal for X.

Lemma 5. *There are at most n^3 circles that are minimal for X.*

Proof. Enter your answer here. $\square$

(b) An ϵ-*approximator* for X is a subset $Y \subseteq X$ that satisfies

$$\left| \frac{|Y \cap C|}{|Y|} - \frac{|X \cap C|}{|X|} \right| \leq \epsilon$$

for each $C \in \mathcal{C}$. Show that there is a small ϵ-approximator for X.

Lemma 6. *For every $\epsilon > 0$, there is an ϵ-approximator for X of size $O(\frac{1}{\epsilon^2} \log n)$.*

Proof. Enter your answer here. $\square$

(c) An ϵ-*net* for X is a subset $Y \subseteq X$ where for each $C \in \mathcal{C}$ that satisfies $|X \cap C| \geq \epsilon |X|$, we have that $Y \cap C \neq \emptyset$.[1] Show that there is a small ϵ-net for X.

Lemma 7. *For every $\epsilon > 0$, there is an ϵ-net for X of size $O(\frac{1}{\epsilon} \log n)$.*

Proof. Enter your answer here. $\square$

(d) **Bonus.** Improve the results in parts (b) and (c) to remove any dependence on n.

[1] Aside: how do the notions of ϵ-approximators and ϵ-nets compare to each other?