This exam lasts 120 minutes.
Write your answers in the separate answer booklet.
Please return this question sheet and your cheat sheet with your answers.

Given any positive integer k as input, the function call RANDOM(k) returns an integer chosen independently and uniformly at random from the set $\{1, 2, \ldots, k\}$ in $O(1)$ time.

---

1. (a) The *left spine* of a binary tree is the path from the root to the leftmost node. For example, if the root has no left child, the left spine contains only the root. What is the expected number of nodes in the left spine of an n-node treap? *[Hint: What is the probability that the node with kth smallest search key is in the left spine?]*

   (b) What is the expected number of leaves in an n-node treap? *[Hint: What is the probability that the node with kth smallest search key is a leaf?]*

   You do ***not*** need to prove that your answers are correct.

2. Recall that a *queue* maintains a sequence of items subject to the following operations.

   - PUSH(x): Add item x to the end of the sequence.
   - PULL: Remove and return the item at the beginning of the sequence.
   - SIZE: Return the current number of items in the sequence.

   It is easy to support all three operations in $O(1)$ worst-case time, using a doubly-linked list and a counter. Now consider the following new operation, which removes every tenth element from the queue, starting at the beginning, in $\Theta(n)$ worst-case time.

```
DECIMATE:
  $n \leftarrow$ SIZE
  for $i \leftarrow 0$ to $n-1$
      if $i \bmod 10 = 0$
          PULL    ⟨⟨result discarded⟩⟩
      else
          PUSH(PULL)
```

   Prove that in any intermixed sequence of PUSH, PULL, and DECIMATE operations, the amortized cost of each operation is $O(1)$.

3. Let $G = (V, E)$ be a connected undirected graph. For any vertices u and v, let $d_G(u, v)$ denote the length of the shortest path in G from u to v. For any *sets* of vertices A and B, let $d_G(A, B)$ denote the length of the shortest path in G from any vertex in A to any vertex in B:

$$d_G(A, B) = \min_{u \in A} \min_{v \in B} d_G(u, v).$$

   Describe and analyze a fast algorithm to compute $d_G(A, B)$, given the graph G and subsets A and B as input. You do ***not*** need to prove that your algorithm is correct.

4. Consider the following modification of the standard algorithm for incrementing a binary counter.

```
INCREMENT($B[0..\infty]$):
    $i \leftarrow 0$
    while $B[i] = 1$
        $B[i] \leftarrow 0$
        $i \leftarrow i + 1$
    $B[i] \leftarrow 1$
    SOMETHINGELSE($i$)
```

The only difference from the standard algorithm is the function call at the end, to a black-box subroutine called SomethingElse.

Suppose we call Increment n times, starting with a counter with value 0. The amortized time of each Increment clearly depends on the running time of SomethingElse. Let $T(i)$ denote the worst-case running time of SomethingElse(i). For example, we proved in class that Increment algorithm runs in $O(1)$ amortized time when $T(i) = 0$.

(a) What is the amortized time per Increment if $T(i) = 42$?

(b) What is the amortized time per Increment if $T(i) = 2^i$?

(c) What is the amortized time per Increment if $T(i) = 4^i$?

(d) What is the amortized time per Increment if $T(i) = \sqrt{2}^i$?

(e) What is the amortized time per Increment if $T(i) = 2^i/(i+1)$?

You do ***not*** need to prove your answers are correct.

5. A *data stream* is a long sequence of items that you can only read only once, in order. Every data-stream algorithm looks roughly like this:

```
DOSOMETHINGINTERESTING(stream $S$):
    repeat
        $x \leftarrow$ next item in $S$
        ⟨⟨do something fast with $x$⟩⟩
    until $S$ ends
    return ⟨⟨something⟩⟩
```

Describe and analyze an algorithm that chooses one element uniformly at random from a data stream.

Your algorithm should spend $O(1)$ time per stream element and use only $O(1)$ space (excluding the stream itself). ***You do not know the length of the stream in advance***; your algorithm learns that the stream has ended only when a request for the next item fails.

You do ***not*** need to prove your algorithm is correct.