CS 263 – Analysis of Algorithms

Homework 2, 50 Points
Due: Sunday, January 25, 11:00pm

This homework must be turned in electronically using the course GradeScope website.

Solutions **must** be submitted as a PDF file.

Problem 1: The "Expensive-1" Binary Counter

Consider a standard binary counter with k bits, initially all set to 0. We perform a sequence of n INCREMENT operations. However, the hardware is unique:

- Flipping a bit from **1 to 0** costs **1 unit of energy**.
- Flipping a bit from **0 to 1** costs **5 units of energy**.

1. Using the Accounting Method, assign a specific amortized cost (in cyber-dollars) to the INCREMENT operation.
2. Prove that your assigned amortized cost is sufficient to pay for any sequence of n operations.
3. What is the upper bound on the total actual cost for n operations?

Problem 2: Dynamic Arrays with 1.5x Expansion

In class, we analyzed dynamic arrays that double in size (factor of 2) when full. Consider a dynamic array that expands by a factor of **1.5** instead.

- The array starts with a capacity of 1.
- The cost of inserting into a non-full array is 1.
- When an insertion occurs and the array is full (size k), a new array of size 1.5k is allocated, all k items are copied over (cost k), and the new item is inserted (cost 1).

Using Accounting Method, show that the amortized cost of an insertion is still O(1).

Problem 3: The Queue via Two Stacks

A standard FIFO Queue can be implemented using two Stacks, S and T.

- Enqueue(x): Push x onto S. (Actual cost: 1)
- Dequeue(): If T is empty, pop all elements from S and push them onto T, then pop from T. If T is not empty, simply pop from T. (Actual cost: varies).

Define a potential function based on the number of elements in S and T to prove that the amortized cost of both Enqueue and Dequeue is O(1).

Problem 4: The "Clear-All" Stack

Suppose you are designing a data structure that supports the following operations:

1. Push(x): Adds item x to the stack. Actual Cost = 1.
2. Pop(): Removes the top item. Actual Cost = 1.
3. Clear(): Removes **all** items from the stack. Actual Cost = k, where k is the number of elements currently in the stack.

Argue whether the amortized cost of Clear() is O(1) or not.

- If it is O(1), perform an amortized analysis (using either the Accounting or Potential method) to prove that the amortized cost of all three operations is bounded by a constant.
- If it is not, provide a counter-example sequence of operations where the average cost per operation grows unboundedly.

Problem 5: The Oscillating Counter

A standard binary counter supports the INCREMENT operation with an amortized cost of O(1). Suppose we modify the counter to support a DECREMENT operation as well (which subtracts 1 from the value). Provide a sequence of n operations (consisting of both INCREMENT and DECREMENT) where the total actual cost is $\Theta(n \log n)$.