CS 430 Lecture 20 Activities

Opening Questions

1. How do you think the allocated size growth of a dynamic array like Java's ArrayList is implemented? How much bigger does it grow when needed? What is the runtime for a sequence of n insertions starting from a default size of 10 considering the worst individual insert? The first few inserts run in $O(1)$ time in the continious memory that was initially assigned. When the array runs out of memory, it has to copy all of the current elements into a new, larger set of memory. The amount of time it would take to copy everything fron the initial array would be $O(\#\ of\ copies)$. If the worst case append is $O(n)$, then the runtime would be $nO(n) \rightarrow O(n^2)$.

Amortized (to pay off gradually) Analysis

So far, we have analyzed best and wort case running times for an operation without considering its context. With amortized analysis, we study a sequence of operations rather than individual operations. An amortized analysis is any strategy for analyzing a sequence of operations to show that the average cost per operation is small, even though a single operation within the sequence might be expensive.

Aggregate Method of Amortized Analysis

1. Can we do a better analysis by amortizing the cost over all inserts? Starting with a table size one and doubling the size when necessary, make a table showing the first 10 inserts and determine a formula for $\text{COST}(i)$ for the cost of the ith insert. Then aggregate "add up" all the costs and divide by n (aggregate analysis).

Table 20.1: Aggregate Analysis for Appends

Append #:	1	2	3	4	5	6	7	8	9	10
Memory Size	1	2	4	4	8	8	8	8	16	16
Total Cost (units)	1	2	3	1	5	1	1	1	9	1
Memory Cost (units)		1	2						8	

The memory cost is the cost in that append of creating a larger volume, it was created by removing the $O(1)$ cost of appending form the total cost row. Any iteration that didn't create more memory is 0, but is shown as blank.

$$\text{Append Cost per Operation} + \text{Memory Allocation Cost}$$
$$\sum_{i=1}^{n} 1 + \sum_{j=0}^{\lg n} 2^j \tag{20.1}$$
$$n + \sim 2n = \theta(n)$$

The total cost for n appends amortized average cost is $\frac{\theta(n)}{n} = \theta(1)$.

Accounting Method of Amortized Analysis

Figure out a specific amortized cost to be allocated to each operation to ensure you have enough "balance" to handle the bad operations.

Charge ith operation a fictitious amortized cost $\hat{c}_i$, where \$1 pays for 1 unit of work (i.e., time).

- This fee is consumed to perform the operation.
- Any amount not immediately consumed is stored in the bank for use by subsequence operations.
- The bank balance must not go negative! We must ensure that for all n

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i$$

Thus, the amortized costs provide an upper bound on the total true costs.

2. For the previous ArrayList example, determine the amortized cost $\hat{c}_i$ necessary. If we give \$3 per append and each assignment costs \$1. Series of n append (some need growth).

Table 20.2: Accounting Analysis for Appends

Append #:	1	2	3	4	5	6	7	8	9	10
Insert (\$):	+3	+3	+3	+3	+3	+3	+3	+3	+3	+3
Assignment (Out \$)	1	2	3	1	5	1	1	1	9	1
Balance After:	2	3	3	5	3	5	7	9	3	5

We chose \$3 as the insert "fee" as the coefficient of n we found earlier was ~ 3.

Consider, as a second example, a binary counter that is being implemented in hardware. Assume that the machine on which it is being run can flip a bit as its basic operation. We now want to analyze the cost of counting up from 0 to n (using k bits).

Decimal	Binary
1	000001
2	000010
3	000011
4	000100
5	000101
...	...
n	100110

3. What is the naive worst-case analysis for how many bits we need to flip? Assuming you have some set of bits that starts with a 0 and every other bit is a 1 $01_1 1_2 1_3 1_4 \ldots 1_{k-1}$ (where $k-1$ is only the number of 1's), the next flip would flip all k bits, so we would have k operations ...

4. Use the aggregate method to perform a more careful analysis for n increments of a binary counter.

Table 20.3: Amortized Cost of Flipping bits

1	10	11	100	101	110	111
$0 \to 1$	$1 \to 2$	$2 \to 3$	$3 \to 4$	$4 \to 5$	$5 \to 6$	$6 \to 7$
1	2	1	3	1	2	1

- How often does the lowest bit flip? **Flips on every increment (n)**
- 2nd lowest bit? $\mathbf{\frac{n}{2}}$ times.
- 3rd lowest bit? $\mathbf{\frac{n}{4}}$ times.
- $\vdots$
- kth leftmost bit? $\frac{n}{2^{k-1}}$

$$n + \frac{n}{2} + \frac{n}{4} + \cdots + \frac{n}{2^{k-1}} \leq 2n$$

$$\text{Amortized Average Cost for } n \text{ increments } = \frac{2n}{n} = O(1)$$

5. Use the accounting method to perform a more careful analysis for n increments of a binary counter. Price for each increment = \$2. Cost to flip a bit = \$1

Table 20.4: Accounting Cost of Flipping bits

	1	10	11	100	101	110	111	1000
Counter	$0 \to 1$	$1 \to 2$	$2 \to 3$	$3 \to 4$	$4 \to 5$	$5 \to 6$	$6 \to 7$	$7 \to 8$
In (Price)	2	2	2	2	2	2	2	2
Out Cost	1	2	1	3	1	2	1	4
Balance	1	1	2	1	2	2	3	1