# Lab07-Amortized Analysis

CS214-Algorithm and Complexity, Xiaofeng Gao & Lei Wang, Spring 2021.

∗ If there is any problem, please contact TA Yihao Xie.
∗ Name: Zilong Li Student ID: 518070910095 Email: logcreative-lzl@sjtu.edu.cn

1. Suppose we perform a sequence of $n$ operations on a data structure in which the $i$ th operation costs $i$ if $i$ is an exact power of 2, and 1 otherwise. Use an accounting method to determine the amortized cost per operation.

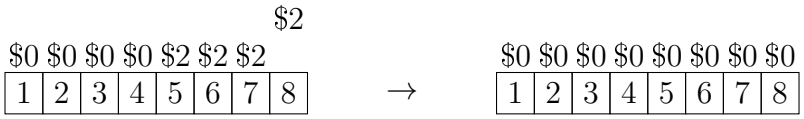
   **Solution.** Define $C_i$ as the cost of the $i$th operation,

   $$C_i = \begin{cases} i, & \text{if } i \text{ is an exact power of 2,} \\ 1, & \text{otherwise} \end{cases}$$

   **Case 1: $i$ is not the power of 2.** an amortized cost $\hat{C}_i = \$3$ is charged:

   - \$1 pays for itself;
   - \$2 is stored for later cost expanding, \$1 \$1 for paying one of the recent $\frac{i}{2}$ items, \$1 for paying one of the old $\frac{i}{2}$ items.

   **Case 2: $i$ is the power of 2.** an amortized cost $\hat{C}_i = \$2$ is charged, in order not to have leftouts.

   Thus,

   $$T(n) = \sum_{i=1}^{n} C_i \leq \sum_{i=1}^{n} \hat{C}_i = 3n - \lfloor \log_2 n \rfloor$$

   □

2. Consider an ordinary **binary min-heap** data structure with $n$ elements supporting the instructions INSERT and EXTRACT-MIN in $O(\log n)$ worst-case time. Give a potential function $\Phi$ such that the amortized cost of INSERT is $O(\log n)$ and the amortized cost of EXTRACT-MIN is $O(1)$, and show that it works.

3. Assume we have a set of arrays $A_0, A_1, A_2, \cdots$, where the $i^{th}$ array $A_i$ has a length of $2^i$. Whenever an element is inserted into the arrays, we always intend to insert it into $A_0$. If $A_0$ is full then we pop the element in $A_0$ off and insert it with the new element into $A_1$. (Thus, if $A_i$ is already full, we recursively pop all its members off and insert them with the elements popped from $A_0, ..., A_{i-1}$ and the new element into $A_{i+1}$ until we find an empty array to store the elements.) An illustrative example is shown in Figure 1. Inserting or popping an element take $O(1)$ time.

   (a) In the worst case, how long does it take to add a new element into the set of arrays containing $n$ elements?

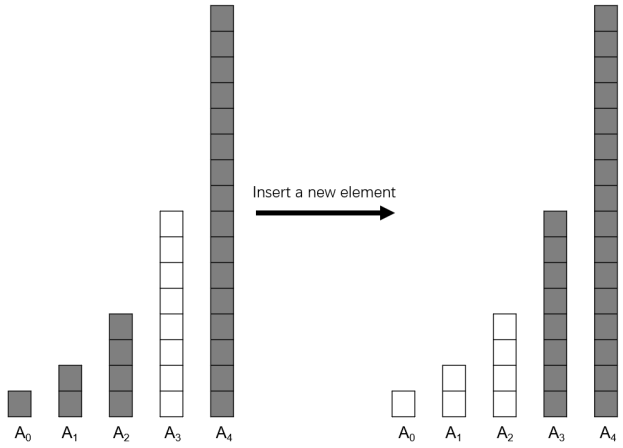



Figure 1: An example of making room for one new element in the set of arrays.

**Solution.** The worst case comes to that $n$ is the power of 2, assumming it to be $n = 2^i$. All elements from $A_0, \cdots, A_{i-1}$ has to be popped and pushed into $A_i$. The inserted element will be inserted into $A_i$ as well.

$$\Omega(n) = 2(1 + \cdots + 2^{i-1}) + 1 = 2^{i+1} - 1$$

□

(b) Prove that the amortized cost of adding an element is $O(\log n)$ by *Aggregation Analysis*.

**Solution.** Enumerate the cost of different $n$:

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| $C_i$(new pop) | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $C_i$(pop+push) | | 2 | | 6 | | 2 | | 14 |
| $i$ | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| $C_i$(new pop) | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $C_i$(pop+push) | | 2 | | 6 | | 2 | | 30 |

Denote $T(n)$ as the sum of operations for inserting $n$ elements. The recurrence of $T(n)$ is:

$$\begin{aligned} T(1) &= 1 \\ T(2^i - 1) &= 2T(2^{i-1} - 1) + \Omega(2^{i-1}) \\ T(2^i) &= T(2^i - 1) + \Omega(2^i) \end{aligned}$$

Because the worst case comes to $n = 2^i$, the aggregation analysis will provide the upper bound reflected by the case of $\frac{T(2^i)}{2^i}$.

$$\begin{aligned} T(2^i) &= 2^{i+1} - 1 + T(2^i - 1) \\ &= 2^{i+1} - 1 + 2T(2^{i-1} - 1) + \Omega(2^{i-1}) \\ &= 2^{i+1} - 1 + 2^{i-1}T(1) + 2^{i-2}\Omega(2^1) + \cdots + 2^1\Omega(2^{i-2}) + \Omega(2^{i-1}) \\ &= 2^{i+1} - 1 + 2^{i-1} + 2^i - 2^{i-2} + 2^i - 2^{i-3} + \cdots + 2^i - 1 \\ &= 2^{i+1} + (i-1)2^i \end{aligned}$$

By *Aggregation Analysis*, the amortized cost of adding an element is

$$\frac{T(2^i)}{2^i} = i + 1 = \log_2 n + 1 = O(\log n)$$

□

(c) If each array $A_i$ is required to be sorted but elements in different arrays have no relationship with each other, how long does it take in the worst case to search an element in the arrays containing $n$ elements?

(d) What is the amortized cost of adding an element in the case of (c) if the comparison between two elements also takes $O(1)$ time?

**Remark:** Please include your .pdf, .tex files for uploading with standard file names.