Lab07-Amortized Analysis

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

∗ If there is any problem, please contact TA Yihao Xie.
∗ Name:Liu Yutian Student ID:519021910548 Email: stau7001@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 amoritized cost as:

Operation	Real Cost C_{op}	Amortized Cost $\widehat{C_{op}}$
$i = 2^k$th operation	i	3
else	1	3

With the above definition, we can use the following diagram to illustrate the relationship between C_{op} and $\widehat{C_{op}}$:

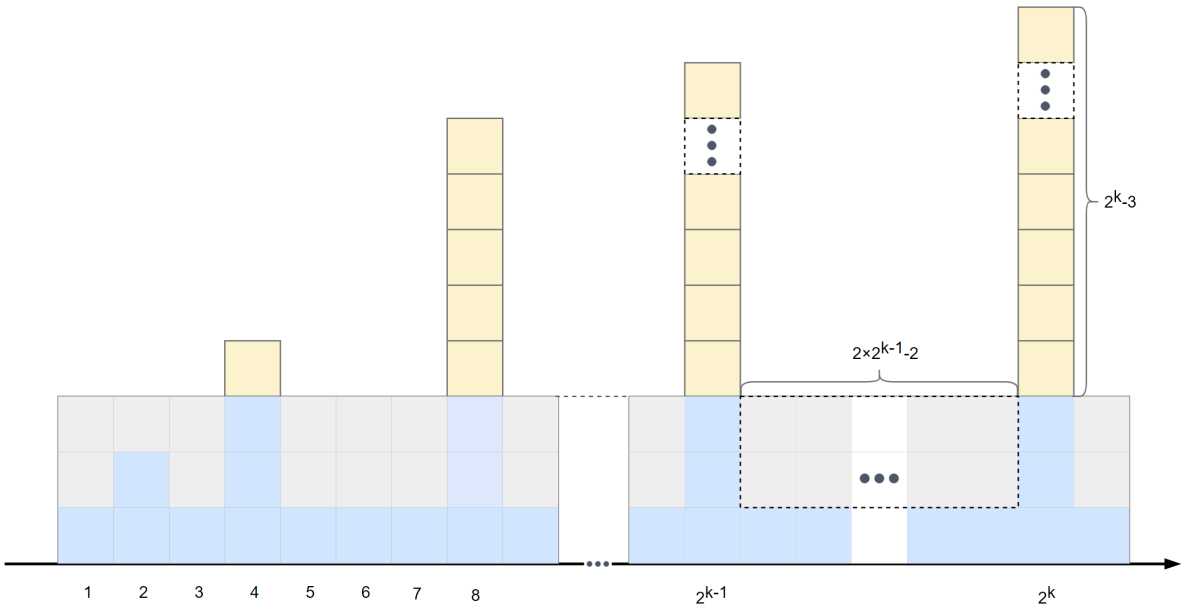

Figure 1: Relationship between C_{op} and $\widehat{C_{op}}$

Since there must be $2^{k-1}-1$ operations between the 2^k-1th and 2^kth operation, $2(2^{k-1}-1) = 2^k-2$ costs will be stored as prepaid credit. And this prepaid credit will be used later for the $2k$th operation which need 2^k-3 more costs.

Thus, $\forall n$,we have:

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} C_i \\ &= n_1 \times 1 + n_2 \#2^k\text{th } operation \\ &= (n - \lfloor \log_2 n \rfloor - 1) + (1 + 2 + \cdots + 2^{\lfloor \log_2 n \rfloor}) \\ &\leq 3n \end{aligned}$$

where n_1 is the number of 2^kth operation and $n_2 = n - n_1$.

Therefore, the amortized cost $\widehat{C_{op}} = 3$. □

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 Φ 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.

Solution. Assume that worst-case time complexities of the instructions INSERT and EXTRACT-MIN are less than $C_1 \log n$ and $C_2 \log n$ when n is sufficiently large.

Define potential function Φ:$S \to R$:

$$\Phi(S_i) = \max\{C_1, C_2\} \cdot n \log(n+1)$$

where n is the number of element in min-heap S_i.

Thus, we have:
INSERT:

$$\Phi(S_i) - \Phi(S_{i-1}) = \max\{C_1, C_2\}[n \log(n+1) - (n-1)\log n] \leq C_3 \log n$$
$$\widehat{C_i} = C_i + \Phi(S_i) - \Phi(S_{i-1}) \leq C_1 \log n + C_3 \log n = O(\log n)$$

EXTRACT-MIN

$$\Phi(S_i) - \Phi(S_{i-1}) = \max\{C_1, C_2\}[(n-1)\log n - n \log(n+1)] \leq -\max\{C_1, C_2\} \log n$$
$$\widehat{C_i} = C_i + \Phi(S_i) - \Phi(S_{i-1}) \leq C_2 \log n - \max\{C_1, C_2\} \log n = O(1)$$

□

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 2. 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?

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

   (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?

   Solution.

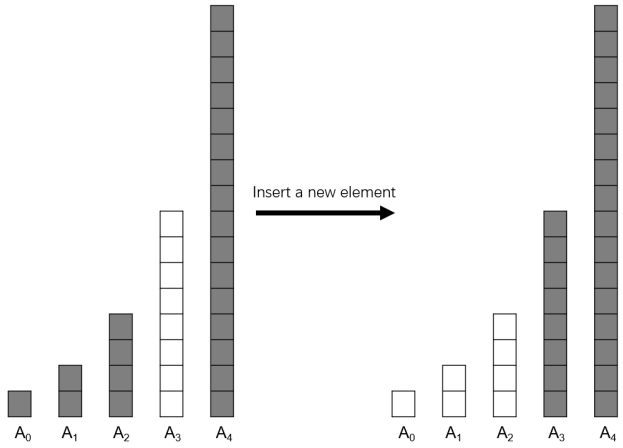

Figure 2: An example of making room for one new element in the set of arrays.

(a) The worst case happens when $A_0, A_1, \cdots, A_m$ ($m = \log_2 n$) are all full. Therefore, we need to pop all members in $A_0, A_1, \cdots, A_m$ and insert them and the new element into A_{n+1}, and the time complexity will be $O(1) + 2\sum_{i=0}^{m} 2^m = O(1) + n - 1 = O(n)$.

(b)

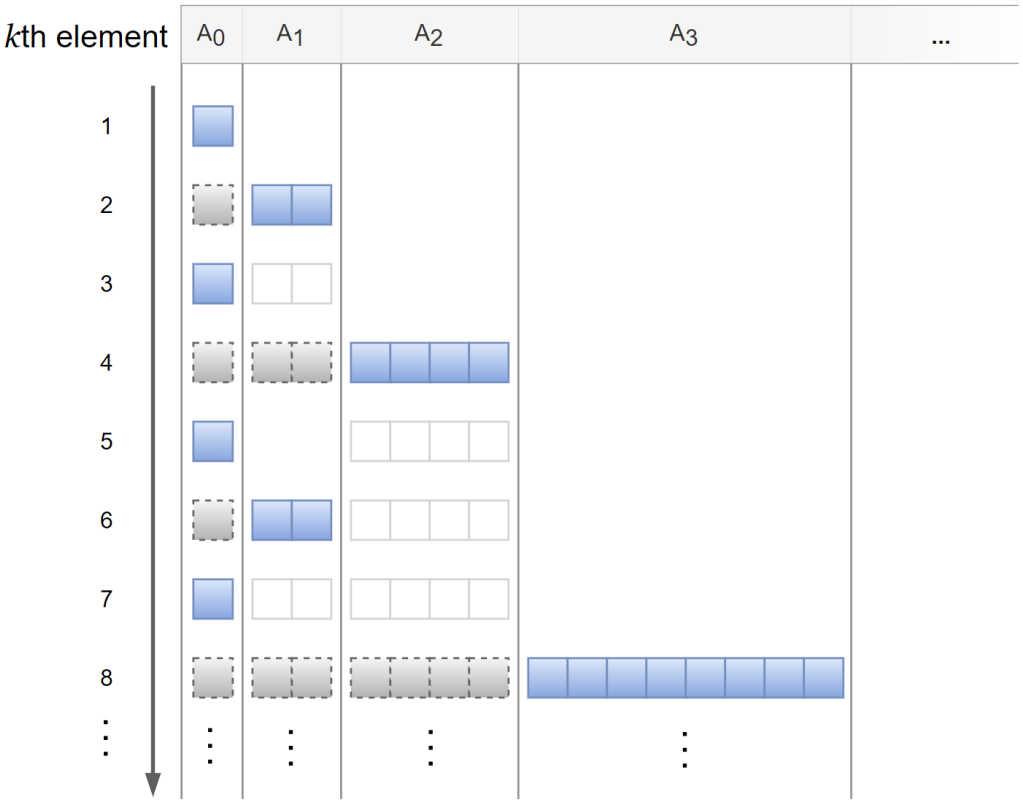

Figure 3: POP and INSERT operations when inserting the i-th element. Blue represents INSERT operations and gray represents POP operations

We can see some patterns in the graph that in a sequence of n Add operations:
A_0 is inserted or popped each time ADD is called; $\rightarrow n$ times
A_1 is inserted or popped every other time ADD is called; $\rightarrow \frac{n}{2}$ times
...

$A_m \to \frac{n}{2^m}$ times

Thus, we have:

$$\begin{aligned} T(n) &= \sum_{i=0}^{\lfloor \log_2 n \rfloor} \lfloor \frac{n}{2^i} \rfloor \times A_i \\ &= \sum_{i=0}^{\lfloor \log_2 n \rfloor} \lfloor \frac{n}{2^i} \rfloor \times 2^i \\ &\leq n(\log_2 n + 1) \end{aligned}$$

Then the amortized cost of adding an element is $T(n)/n \leq \log_2 n + 1 = O(\log n)$

(c) Suppose we use a binary search in each A_i. In the worst case, we need to look up from A_0 to A_m ($m = \lfloor \log_2 n \rfloor$).
Thus, the time complexity is $O(\frac{\log_2 n(\log_2 n+1)}{2}) = O(\log_2^2 n)$.

(d) Suppose when multiple sequences need to be merged, we choose the two shortest sequences, merge them using the following algorithm, and then use the same algorithm to merge the merged sequence with the next shortest sequence until all sequences are merged into one. In fact, in the case of this question, the order in which we merge the sequences is always newly added element$\to A_0 \to A_1 \to \cdots \to A_m$, and the time consumption will be $2\,(1 + 2 + 2^2 + \cdots + 2^m) = 2^{m+2} - 2$.

Algorithm 1: Merge Algorithm

Input: 2 sorted sequence $A_i[], A_j[]$
Output: An ordered sequence $S[]$ formed by the merging of 2 sequences

```
1  i, j, pos ← 0;
2  while i < sizeof(A_i)&&j < sizeof(A_j) do
3      if A_i[i] < A_j[j] then
4          S[pos++] ← A_i[i++];
5      else
6          S[pos++] ← A_j[j++];
7  while i < sizeof(A_i) do
8      S[pos++] ← A_i[i++];
9  while j < sizeof(A_j) do
10     S[pos++] ← A_j[j++];
```

And if we accumulate all the time consumption of the merge into the merged sequences, which are marked in orange on the graph, we can get that the time consumption contained in orange sequence A_i is $2^{i+1} - 2$. (Here we don't need to calculate the time consumption of Pops because it is already included in the merge time)

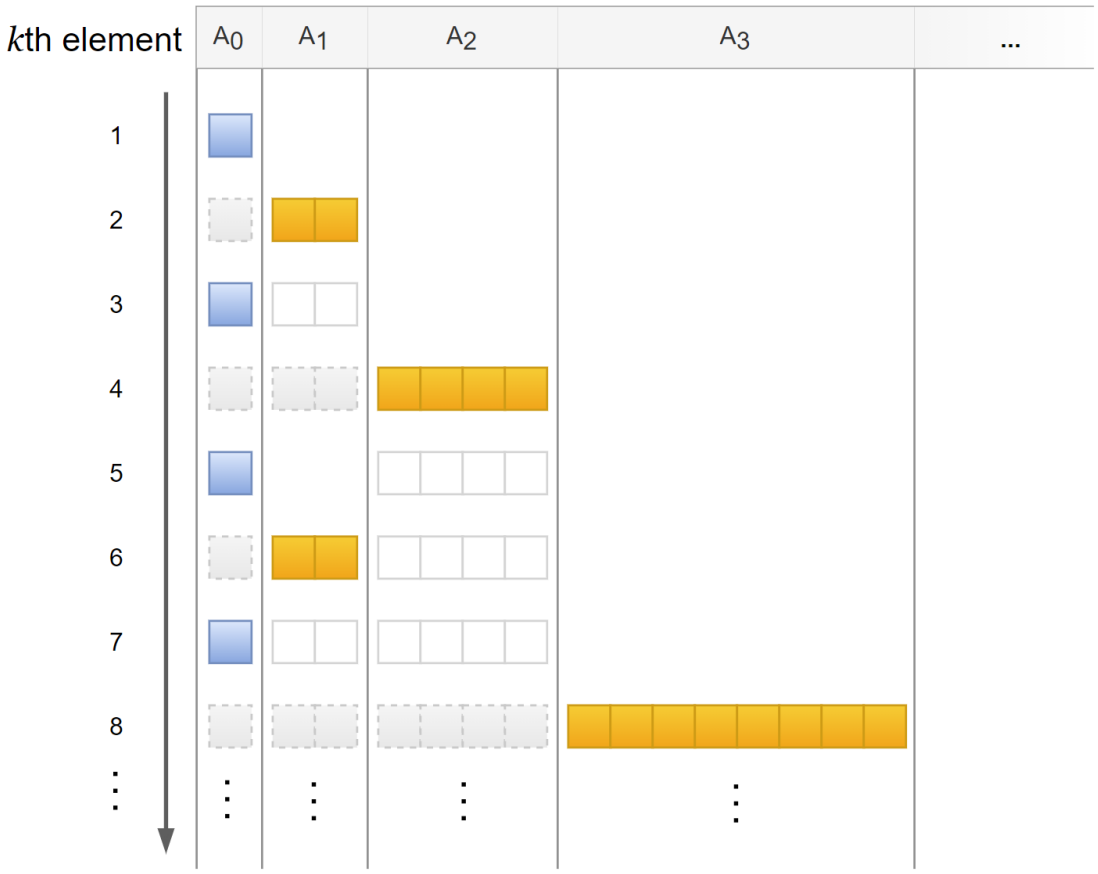

Figure 4: MERGE operations (orange) when inserting the i-th element.

Then, sum up all time consumption (Orange and blue sequences):

$$\begin{aligned} T(n) &= \sum_{i=0}^{\lfloor \log_2 n \rfloor} \lfloor \frac{n}{2^{i+1}} \rfloor \times (2^{i+1} - 2) + \lceil \frac{n}{2} \rceil \\ &\leq \sum_{i=0}^{\lfloor \log_2 n \rfloor} n + \lceil \frac{n}{2} \rceil \\ &\leq n \log n + \lceil \frac{n}{2} \rceil \end{aligned}$$

So the amortized cost of adding an element is $T(n)/n = O(\log n)$.

□

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