Advanced Algorithm HW6

Seungho Jang

October 2020

1 Q1

Average-case analysis is analyzing average or mean on each operation. For example, we know that the worst case scenario for multi-pop operation could result in O(n) and total of n operations is involved. Then, we can calculate the worst case big O(n^2). In the average-case analysis, we make an assumption about the total number of n operations such that this n operation could be among MULTIPOP, POP, and PUSH. Whichever we pick what types of operations will be involved for this analysis. The result can be worse than the actual algorithm, and this will give much slower than the average case. Therefore, this case analysis relies on probabilistic assumptions about what types operations will be involved in order to calculate the running time such as stack.

On the contrary to average-case, amortized analysis is to analyze the data structure in terms of a sequence of operation. In the stack operation, if we have any n sequence of operations; pop, push, multipop on an empty stack at initial point, we can only pop each element at most once for each time after pushing that element into the stack. Same concept for multipop can be applied as well such that we can only do multipop(s, k) if there are k number of elements in the stack. So this result shows that number operations and running time are dependent on number of push operations, and this will give big O(n).

The amortized analysis is important because it is possible that a single operation can take a lot of time but if we take the consideration of the sequence of operations, on average cases, one operation may not take as much time. Also, it shows that the average cost of an each operation. Therefore, all these reasons above, the amortized analysis will give you a better upper-bounds for worst case.

2 Q2

This was discussed on Q1 as well. On average-case analysis, we make an assumption that what types of operation should be involved such that that specific operation could give a worst case scenario. For our example in stack, the worst case scenario of using multipop is O(n) because of the number of pop opera-

tions inside of the while loop, and if there are n operations, then the worst case scenario will be O(n * n), which it will become O(n^2).

On the other hand, within the aggregate analysis, we do not make any assumption about what operations can be done in data structure such that every operation in that sequence will be participated. In other words, we will look at the sequence as a whole process to solve some problem. If we take a close look at pop and multipop operations in our stack example, they might not participate as much as push operations because pop or multipop can be done only if there are elements in the stack. For example, multipop(s,k) will participate in the running time until a number of k elements are filled in the stack, if not then, it will be just ignored or skipped. Same concept will apply to pop operation, if there is no element in the initial stack, then this operation would not affect on running time as much. This will give the conclusion that most of running time will be dependent on the number of push operation(O(1)). So that's why it will be O(n).

3 Q3

I believe it does hold true even if it is non-empty stack. Even if the initial stack has some elements inside of stack, it's still dependent on the push operation. In the worst-case scenario, if the stack is already filled, then we need to copy the current stack to new stack and push it into the new stack if there is new incoming element or resize it, which it will cost mostly. But again, based on what we have discussed, pop and multipop can affect only if there are elements. Also, even though the stack is not empty, it is possible that running time might be slow in the few operations, but it will still bound to O(n) or at most O(kn).

4 Q4

In the context of aggregate analysis, for the incrementing a binary counter problem, derive and/or discuss the expression for the total number of flips when we perform a sequence of n INCREMENT operations. Assume that the number of bits is k. Why is this calculation important?

The k-bit counter can starts with incrementing or counting upward from 0, such as 000 to 111 if there are 3-bit counter. By using the array A[0..k-1] of bits as the counter, we can count A[0] to A[k-1], and the binary number x that is stored in A can be defined by x = $\sum_{i=0}^{k-1} A[i] * 2^i$.

The following figures will derive/discuss about k-bit counter.

What we can achieve by looking at this problem, we can achieve worst-case is $\in O(k) for single execution of increment. So if there are sequence of n increment operation then it will be \in O(k * n)$.

So, we have to observe A[i] which is 0 through k - 1. From the table above, we can approximate how many flips are associated with in each index of A[i],

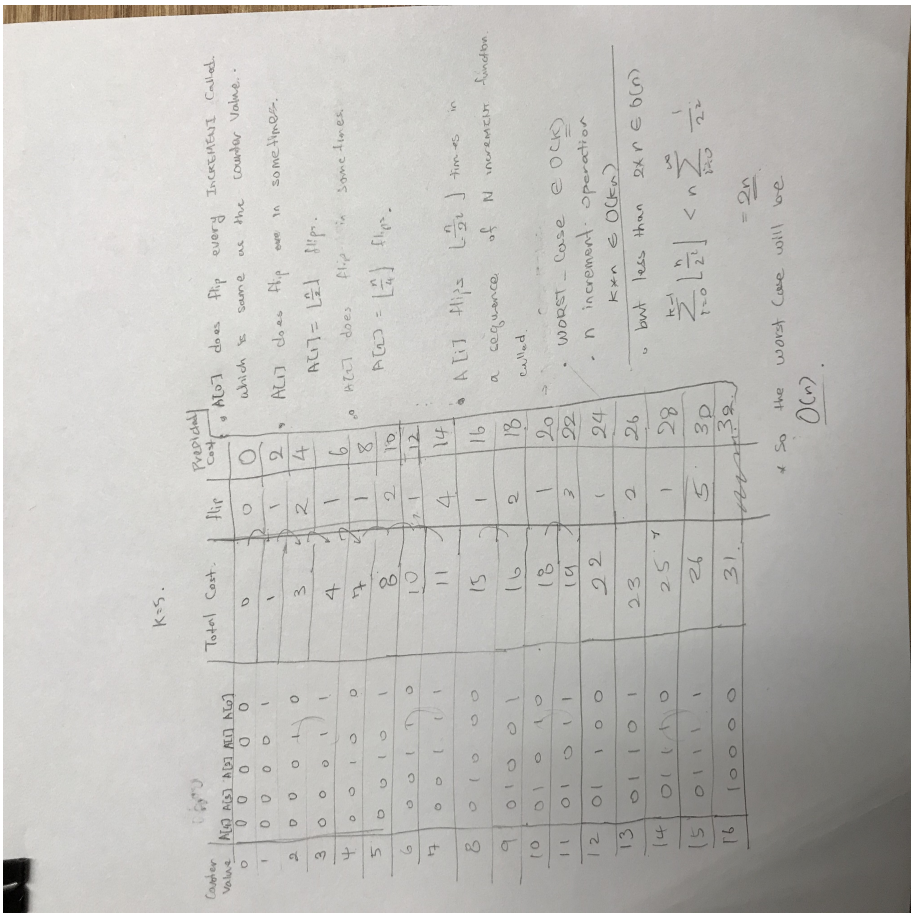

Figure 1: Derivation on K-bit Counter

which is counter value. As the result, we can conclude that in each index, there will be$[n/2^n]$ times flips.

We can tight our approximation O(k*n) to something else. Such that if we put result 2^n where n is the counter value. we can see that the actual total cost doesn't go over the predicted cost of 2^n. So that means that the worst case will be O(n).

Why amortized analysis on k-bit counter is important is it will minimize the overestimation in worst-case scenario, such as having O(k*n) will be worse than O(2*n).

5 Q5

No. If multipush is in seqeuce of operation, the running time of this sequence of operations will be dependent on the number of pushes. So if multipush operations pushes k items on the stack, the one time operation could be O(k) or $\theta(k)$ time. If there is n MULTIPUSH operations were involved, that would take O(kn) or $\theta(kn)$ time. Then the amotized cost will be $\theta(k)$ or O(k) at max.