CSC 310 A_0 Assignment Questions

1. To begin this problem, we need to consider what the question is asking. We have a data structure that requires 1 unit per operation if the operation is not a power of 2. This means that the operations will become fast as time goes on due to the high-cost operations being more spread out over time. Since this is powers of 2, the expensive operations will occur on i = 1, 2, 4, 8, 16,… Therefore, this sequence limited by n will be floor($\log_2$ n) + 1. This means that there are O(log n) expensive operations. Then we can start computing the total cost of the operations. This starts with the cost of every normal operation being 1, so the cost would just be n. The total cost of the expensive operations would be a summation of 1 + all numbers in the sequence 2^floor($\log_2$ n). This will eventually get us to the cost 2n for the higher operation costs bringing the total to n + 2n = 3n. Therefore, total cost is O(n). Finally with aggregate analysis, we can do the operation n / n to get O(1) as the final aggregate cost of the data structure.
2. We start by doing an analysis of the real costs of each operation. Pushing costs 1, popping costs 1 and then every single operation cost k which we don't know what k will be. Since backups are infrequent and random we will need amortized analysis to determine operation costs. We will increase the cost of push and pop operations to 2 so that way we can ensure that we are using the accounting method even though the actual cost would be 1 and we get 1 cost as credit. So, for every operation, we will have k stored cost to pay for the back up operation that costs k, and since each operation has cost 2, there will be 2n total cost for n operations meaning that we have a final cost of O(n).
3. We begin by analyzing the two functions that we are given here and the potential method formula. The potential method formula c^ = c + (after – before) will give us the

amortized cost of the operations. First, we take our two operations and remember that they are O(log n). One is an insert function, meaning we will be storing potential while the other is releasing potential and since both require the same cost, there should be enough to pay for the other operation. So now we need a potential function. We can define this as n log n as that will account for extra storage we need of the extra operations giving us the equation I(H) = c * n log n. n is the number of elements currently in the neap and c is a constant that scales so we can always have a number large enough to pay for the cost of operations. So now we can reanalyze our first functions. Starting with insert, using our function, we have c((n + 1) log(n + 1)) – n log n) and using the fact that this will be equivalent to O(log n) because of the same cost operations, we can assume the equation will equal O(log n). Therefore, our change in potential will equal O(log n).

4. Amortized analysis shows us that the operations that are more expensive do not ultimately affect the final cost of the operations. In this case, most operations will add little to nothing over the final cost. In this way, when the expensive operations come up and take more time than usual, it does not affect the overall cost of the function.
5. To start with the aggregate method, we start by analyzing the cost of flipping each bit. The cost of each operation gets smaller with each bit that we approach, so for the first bit it would be n, for the next n/2, for the next n/4, and so on. We can then do a summation of all operations which will be summation(n/2^i) from 0 to n where n is the number of total operations. Then finally to get the amortized cost of the operations, we will divide our number by n. For accounting method, we start by overestimating the cost for flipping a bit from 0 to 1. Then from there, we can use our stored cost to compute flipping a bit off (1 to 0). This works since we need a bit to be on for it to be flipped off, so if we

overestimate the cost of turning the bit on, then it should cover the cost of turning the bit off. Finally, for the potential method we first need to find a function. We have two operations here for turning the bits on and off. The actual cost will look like (t + 1) and (1 – t) for the change in potential. This gives us a final equation of (t + 1) + (1 - t). This gives us a final value of 2 meaning that the cost of each operation will ultimately show us that the final amortized cost will be O(1).

6. The amortized analysis of splay trees can only be done with the potential method because of the way the operations are more complicated than most data structures that we have looked at in the previous classes. Splaying restructures the tree itself so we need to take that into account as to why we can't use the other two methods for this. We need to analyze the number of nodes in the tree and the height of the tree to determine what our initial time complexity will be. We define this as s(x) where x is the subtree size. From here we can get the function for the operations being r(x) = log(s(x)). Now from here we can do summation(r(x)) for the potential, while the actual cost of a zag-zig operation (and by extension zig-zag operation) is c = 2. So now for change in potential, we get c^ <= 3(r'(x) – r(x)) which we can replace the r(x) with log s(x). Since each zig zag step will always be less than log n, we can reasonably assume that the total cost for an increase in rank will always be log n. Each step at most will cost 3(r'(x) – r(x)). Which brings our total amortized cost to <= 3 log n meaning the amortized cost of splaying is O(log n).

Relevant Sources:

https://dsu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=bb96699a-6dc1-43a6-a14f-b3e1014e2ef9

https://dsu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=af367c91-c4d0-44bc-9eb2-b3e7011cd319

https://dsu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=95c33502-a95f-4d0c-88fa-b3e000010fc0

https://www.geeksforgeeks.org/dsa/introduction-to-amortized-analysis/