Diego Escobedo February 17, 2020

6.046 Problem Set 2

Collaborators: *Temi Omitoogun, Ricky Avina, Marco Fleming, Tema Nwana*

Problem 1

(A) In this case, we are going to be representing the board members' decisions as 100 indicator random variables. In this case , we know that the chance of them rejecting a salary of 200,000 is 20%, since they are choosing uniformly over $[0, 1000000]$ which due to basic probability means each board member has an 80% chance of approving a salary of 200,000. Now, we transform that into an indicator random variable I_i, which takes on 1 (reject salary) with probability 0.2 and 0 (accept salary) with probability 0.8. We can declare a new random variable $Y = \sum_i I_i$. In Parry's case, we want to check whether there will be 50 or more rejections (49 or less mean he gets his money), so we can use Markov's inequality to state that $Pr[Y \geq 50] \leq \frac{E[Y]}{50}$, and since the expectation of a sum of random variables is the sum of the individual expectations, and the expected value of I_i is 0.2, we get that the probability of Parry's salary getting rejected is $\frac{20}{50}$ or 40%. If we want to bump this up to 50%, then we know that the expected value of Y has to be 25. This would happen if Parry increased his proposed salary to 250,000. In other words, if Parry bumps his salary to 250,000, a Markov bound would tell us that the chance of him getting his salary rejected has an upper bound of 50%.

(B) To use Chebyshev's inequality, we need to first find the variance. We know that I_i is a Bernoulli random variable with p = 0.2. Therefore, Var(I_i) = p(1-p) = 0.2*0.8 = .16. So, now that we want to use Chebyshev's, we want to see what the chance of failure would be. In this case, the expected value of Y is 20, and Parry's salary will not get approved if there are 50 or more rejections. So, a failure would constitute a difference between the expected and the actual of 30 or more. Therefore, we can set up a Chebyshev inequality stating that $Pr[|Y - 20| \geq 30] \leq \frac{Var(x)}{900} = \frac{16}{900}$ which is roughly a 1.7% chance. Sounds like Parry has a good chance of getting his modest salary approved!

(C) We want to use the second case of the Chernoff Bounds in this case, $Pr[Y > (1+\beta)\mu] < e^{-\beta\mu/3}$. We want $(1 + \beta)\mu$ to be 49, so since μ is 20, we get $\beta = 1.45$. So, when we plug those numbers into $e^{-\beta\mu/3} = e^{-29/3}$, we get 0.0000633607, or roughly 0.006%. Parry has an excellent chance of getting his salary approved.

(D) Algorithm: Assume we have two arrays, one storing the board members' salaries and another storing their votes. We assume that we can access a given member's salary from the location of their votes and vice versa in constant time. Using the median-finding we derived in lecture, find the median salary from within the salary array (call it the 'proposed salary' for now). Then, check every element of the array of votes and add up whether they would pass the proposed salary. If the proposed salary would not pass, then we want to revise the salary to be lower. So, we eliminate all the salaries higher than the proposed salary, find the median of the new set of salaries. Conversely, if the proposed salary would pass, then we want to revise the salary to be higher (because we might be able to get more money). We recursively perform this step of eliminating the lower half of salaries if the salary passes or eliminating the upper half of salaries if the salary does not pass. Eventually we will arrive at a single salary that will be the maximum amount of money we can get while still passing the salary.

Correctness: This algorithm is correct because at each recursive step, we are checking whether it works or not. If the salary is too high, then we check the lower salaries, because if it did not pass, then there is no way in which a higher salary could pass. Similarly, if the salary passes, we can check the higher salaries (including the salary that passed), because there could potentially be a higher salary that passed in there. This is basically just a fancy version of binary search, with the only difference being that it takes us linear time instead of constant time to find the 'new median'.

Runtime: The recurrence is $T(n) = O(n)$ (to find the median of our array) + $O(n)$ (to check whether the salary would pass) + $T(n/2)$ (to run the problem recursively on either the higher salaries or the lower salaries). Using Master's theorem, we can see that this would take $O(n)$ time.

Problem 2

(A)
The cost of the first insertion is going to be 1 (just insert into single element array). The cost of the second insertion is going to be 1 + 1 (copy array, insert new element). The cost of the third is going to be 2 + 1 (copy array, insert new element). Thus, we get a pattern that can be summarized by $\sum_{i=1}^{n} 1 + (i-1) = \sum_{i=1}^{n} i$ which leads to a cost of n*(n+1)/2 for the insertions. If we do the deletions, we get that the first one is going to cost (n-1), the next one (n-2), so on and so forth (we don't need to delete an element since we can effectively do just that by copying the elements we actually want. So, we get a pattern that can be summarized by 1 (cost of deleting the first element without resizing the array) + $\sum_{i=2}^{n}(i-1) = (n-1) * n/2$. This gives us a total cost of $n^2/2 + 1$, which is $O(n^2)$. If we divide by 2n operations, we get that each operation is $O(n)$.

(B)
Imagine we already have a full array of size n (filled with n elements). At this point, if we add a single element to the array, we are going to create a length 2n array and copy the n elements, and then add the next element. Thus, this takes n+1 operations, or $O(n)$ time. If we remove this element we just added, then we go from a length 2n array to a length n array, which means we need to copy the n elements (exclude the n+1st element because we're deleting it), which takes $O(n)$ time. If we repeat these two procedures repeatedly, each operation takes $O(n)$ time. This shows that there is an adversarial set of inputs that can make the amortized $O(1)$ operations take way longer in practice.

(C)
We will do a proof by cases. By our potential function, we can see that the first expression in the function will be bigger than the second only when m is greater than n/2, so we need to check for those edge cases. There are 4 things that can happen:

1. **Addition of element, array stays the same size:** In this case, we want to solve for our amortized cost $\hat{c_i} = c_i + \Delta\Phi(m, n)$. We will check two cases:
   - **First potential function:** We assume that we have a length n array, and m starts off at n-1 and goes to n after our addition. We know the cost of the addition is 1, the $\Phi_i(m, n)$ is (2*(n) - n) = n, and that $\Phi_{i-1}(m, n)$ is (2*(n-1) - n) = n-2. So, $\hat{c_i} = c_i + \Delta\Phi(m, n) = 1 + n - n + 2 = 3$, which is $O(1)$.
   - **Second potential function:** We assume that we have a length n array, and m starts off at n/4+1 and goes to n/4+2 after our addition. We know the cost of the addition is 1, the $\Phi_i(m, n)$ is (n/2-n/4-1) = n/2-1, and that $\Phi_{i-1}(m, n)$ is (n/2-n/4-2) = n/2-2. So, $\hat{c_i} = c_i + \Delta\Phi(m, n) = 1 + n/2 - 1 - n/2 + 2 = 2$, which is $O(1)$.

- **Transition from second to first potential function:** Just from seeing the form of the potential function, we know that the change in our potential function is going to be a constant, and since the cost of the operation is also constant, the amortized cost is constant.

2. **Deletion of element, array stays the same size:** In this case, we want to solve for our amortized cost $\hat{c_i} = c_i + \Delta\Phi(m, n)$. We will check two cases:

   - **First potential function:** We assume that we have a length n array, and m starts off at n and goes to n-1 after our deletion. We know the cost of the deletion is 1, the $\Phi_i(m, n)$ is (2*(n-1) - n) = n-2, and that $\Phi_{i-1}(m, n)$ is (2*(n) - n) = n. So, $\hat{c_i} = c_i + \Delta\Phi(m, n) = 1 + n - 2 - n = -1$, which is $O(1)$.
   - **Second potential function:** We assume that we have a length n array, and m starts off at n/4+2 and goes to n/4+1 after our deletion. We know the cost of the deletion is 1, the $\Phi_i(m, n)$ is (n/2-n/4-2) = n/2-2, and that $\Phi_{i-1}(m, n)$ is (n/2-n/4-1) = n/2-1. So, $\hat{c_i} = c_i + \Delta\Phi(m, n) = 1 + n/2 - 2 - n/2 + 1 = 1$, which is $O(1)$.

3. **Insertion of element, array doubles:** In this case, we want to solve for our amortized cost $\hat{c_i} = c_i + \Delta\Phi(m, n)$. So, we start off with a full array, which means n = n and m = n. The actual cost of copying n elements and adding another one is n+1. Our $\Phi_{i-1}(m, n)$ is (2*(n) - n) = n. When we add the element, we go to n = 2n, and m = n+1. Therefore, our $\Phi_i(m, n)$ is now (2*(n+1) - 2n) = 2. So, $\hat{c_i} = c_i + \Delta\Phi(m, n) = n + 1 + 2 - n = 3$, which is $O(1)$.

4. **Deletion of element, array gets halved:** In this case, we want to solve for our amortized cost $\hat{c_i} = c_i + \Delta\Phi(m, n)$. So, we start off with a quarter filled array, which means n = n and m = n/4+1. The actual cost of copying n/4 elements is n/4. Our $\Phi_{i-1}(m, n)$ is (n/2 - (n/4+1)) = n/4-1. When we delete the element, we go to n = n/2, and m = n/4. Therefore, our $\Phi_i(m, n)$ is now 0 (with either potential function). So, $\hat{c_i} = c_i + \Delta\Phi(m, n) = n/4 + 0 - n/4 + 1 = 1$, which is $O(1)$.