Rishad Rahman (Deepak's Recitation) March 5, 2015

6.046 Problem 4-1

Collaborators: *None*

(a) The data structure will consist of a queue (can be represented by a link list but will refer to it as a queue for the purpose of the problem) and a doubly linked list. The queue Q will handle maintaining the elements storing them from earliest to most recent push as usual. The linked list L will maintain the invariant where the first node contains the minimum element of the queue and that if a node n contains v, then node $n.next$ contains the minimum of the elements pushed into the queue after v. As a result the last element of the linked list will be the most recent push to the queue and it points to null.

(b) FindMin(): Return $L.head.key$.

Enqueue(v): Push the v to the end of the queue. Then insert v into L via the following. Let $L.tail \leftarrow n$. If $n.key < v$ then $n.next.key \leftarrow v$ and $n.next.next \leftarrow$ null and terminate. Else $n \leftarrow n.prev$ and repeat until $n.prev$ is null in which case we replace L with a linked list of one node which contains v.

Dequeue(): Pop off the first element e of the queue. If $L.head.key = e$ then $L.head \leftarrow L.head.next$.

(c) FindMin(): Correctness follows because of the invariant that $L.head$ contains the minimum element.

Enqueue(v): The only time $L.head.key$ is changed is when $L.head.key > v$ in which case we still maintain that $L.head$ contains the minimum since $v < L.head.key$ means v is less than all the other elements in the queue and our new L only containing v reflects the fact that there are no elements to the right of it after v is pushed. Now we need to verify the invariant that each element n in L after the Enqueue still maintains that $n.next.key$ contains the minimum of the elements in Q pushed after $n.key$. Suppose n' is the node in L is the one where we did $n'.next.key = v$ and $n'.next.next = =$ null. Let's look at a node n in L prior to n'. We have everything pushed after n is not bigger than $n.key$ so $n.key \leq n'.key \leq v$. so that checks. This is obviously also true for $n'' = n'.next$ where $n''.key = v$, and $n''.next = null$ since v is the most recently pushed element into Q.

Dequeue(): If $L.head.key = e$ then $L.head.next$ contained the minimum of the elements pushed afterwards so making it the new head maintains the minimum. Since everything else is greater than the new $L.head.key$, the rest of the structure does not need to be updated.

(d) FINDMIN : This is always $O(1)$ since it is just an access of data.

ENQUEUE: In the worst case we may have to iterate through the entire list so worst case $O(n)$.

DEQUEUE: This just messes with a constant number of pointers hence always $O(1)$.

AMORTIZED COST: I will just scale all constants to 1 for the following arguments since scaling the potential will counterbalance the constants in front of the actual cost. Let $\Phi = |L|$. We have that FINDMIN has an amortized cost of $\hat{c_1} = 1$ since $\Delta\Phi = 0$. DEQUEUE has an amortized cost of $\hat{c_2} = \{1, 0\}$ since $\Delta\Phi = 0, -1$ depending on whether we delete the minimum or not. ENQUEUE has an actual cost of $1 + k$ where k is the number of elements we traversed through during the insert into L. However since the size of L decreases by k we have that $\Delta\Phi = -k$ so the amortized cost is $\hat{c_3} = 1$. Hence m operations has $O(m)$ cost.

Rishad Rahman (Deepak's Recitation) March 5, 2015

6.046 Problem 4-2

Collaborators: *None*

(a) Suppose the elements of the array are $a_1, a_2, ..., a_m$ so that $a_1 \leq a_2 \leq a_3 ... \leq a_m$. If our pivot a_r is such that $\frac{m}{4} + 1 \leq r \leq \frac{3m}{4}$ then the partition splits the original array into two one whose elements are $\leq a_r$, $M[1...r-1]$, and one whose elements are $\geq a_r$, $M[r+1...m]$. Since $\frac{m}{4} \leq r - 1 \leq \frac{3m}{4} - 1$ and $\frac{m}{4} \leq m - r \leq \frac{3m}{4} - 1$ we have that their sizes cannot exceed $\frac{3m}{4}$ so either x_i is the pivot or must be in one of these subarrays. There are $\frac{m}{2}$ choices for r which guarantees these hence our probability is at least $\frac{1}{2}$

(b) Note the Chernoff Bound is true even if the coin was weighted towards heads because the probability that at least $c \log n$ are produced becomes strictly greater. Suppose QUICKSORT is run up to $3(\alpha + c) \lg n$ times on subarrays involving x_i. We let our coin flip be (a), heads if we recurse on a subarray of size at most $\frac{3}{4}m$ or x_i is a pivot, else heads. Note that if we get at least $\log_{\frac{4}{3}} n$ heads then the size of the remaining array which x_i is possibly in is at most $\left(\frac{3}{4}\right)^{\log \frac{4}{3} n} n = 1$ so this guarantees termination. So we let $c = \log_{\frac{4}{3}} 2$ and since $\alpha = 2$ we have that out of $3(2 + \log_{\frac{4}{3}} 2) \lg n$ Quicksorts we have that with probability at least $1 - \frac{1}{n^2}$ we guarantee termination on subarrays involving x_i and thus the number of comparisons is at most $3(2 + \log_{\frac{4}{3}} 2) \lg n$ so $d = 3(2 + \log_{\frac{4}{3}} 2)$.

(c) The probability that each x_i is compared with more than $d \log n$ pivots is less than $\frac{1}{n^2}$ so by the Union Bound the probability that this is true for all the x_i is less $\frac{1}{n}$ hence every x_i are compared with at most $d \log n$ pivots with probability $1 - \frac{1}{n}$ but there are n such x_i so the bound on the total number of comparisons is $dn \log n$ therefore $d = d' = 3(2 + \log_{\frac{4}{3}} 2)$.

(d) This means we want a probability of at least $\frac{1}{n^{\alpha+1}}$ in (b) hence we let $\alpha \rightarrow \alpha + 1$ and we get $d = 3(\alpha + 1 + \log_{\frac{4}{3}} 2)$.