PSET-2

Fatima Mahmood (24314)

February 2024

1 Question 1

Q1(a) For this, we will use a modified deterministic quick select. Since we want to find the largest $n^{3/4}$ elements, we want to find such a value of k (where k is the index of the array) that k represents the (n-$n^{3/4}$) smallest element of the array. This is because once we find this k, we will know that all the elements which lie to the left of this k are smaller so we can easily discard them and are then left with the largest $n^{3/4}$ elements.
We will find the pivot p similar to how we did in our Lecture Notes and then recurse on the pivot, comparing it to k.
Case 1: If $p < k$, then p+1
Case 2: If $p > k$, then p-1
We do this till $p = k$
Sorting the list will take $n^{3/4}$ $\log(n^{3/4})$ (because a list of length n takes us $n\log(n)$). As discussed in our lectures, we know that finding and recusing a pivot will take us $\Theta(n)$ comparisons so this can be done in $\Theta(n)$.

Q1(b) We will use a proof by contradiction. We will assume less than n-1 comparisons are needed. We will use a binary decision tree where the internal nodes will compare each element against the median where median is $\lceil n/2 \rceil$. If the element is smaller, it goes down the left path and if greater then the right path. The height of our decision tree will tell us the number of comparisons performed. The algorithm should be able to give us all the elements lesser and greater than the median so the tree should be able to determine $\lceil n/2 \rceil$ -1 smallest elements and the $\lfloor n/2 \rfloor$ largest elements so the height of the decision tree should be at least $max(\lceil n/2 \rceil$ -1 , $\lfloor n/2 \rfloor$). We need to find how large n must be for this to hold. When n is even, our median
$\lceil n/2 \rceil = \lfloor n/2 \rfloor = n/2$
So minimum height of tree is $max(n/2$-1 , $n/2)$. For the height of the decision tree to exceed n-1 our n should be:
$max(n/2$-1 , $n/2) > n$ -1
$n/2 > n$-1
$n < 3$
This means when n is even, n must be greater than or equal to 3. Let's see

what happens when n is odd:
Median will be: $\lceil n/2 \rceil = \lfloor n/2 \rfloor + 1 = (n+1)/2$
So minimum height of tree is $max((n+1)/2 - 1, (n+1)/2) > n-1$
$(n+1)/2 > n-1$
$n < 3$
This means that when n is odd, n must be greater than or equal to 3.
In both cases (even or odd) n must be greater for equal to 3 for the decision tree to have a length longer than n-1.

2 Question 2

The time taken to find the median of A and B each will take $\mathcal{O}(1)$ as both arrays are sorted. Let's divide this into 3 cases:
Case1: If median-A = median-B, then we have found the median for the union of A and B.
Case 2: If median-A <median-B this means the median of A union B is amongst the elements greater than median-A and less than median-B. Simply put, it means median lies in the range [median-A,median-B]
Case 3: If median-A >median-B this means the median of A union B is amongst the elements lesser than median-A and greater than median-B. Simply put, it means median lies in the range [median-B,median-A].
We will recursively, keep applying Case 2 or Case 3 (depending on the scenario) till we find the median. Since we are effectively discarding half the array with each search, this takes us $\mathcal{O}(logn)$ time.
Since finding the median for the arrays will take us $\mathcal{O}(1)$, the upper-bound for this algorithm will be $\mathcal{O}(logn)$ as stated as we are effectively reducing search space on the basis of finding the median.
To prove the lower-bound we will use decision trees. The height of our decision tree tells us the number of comparisons made. The height will be log(n!) as we have n! possible permutations. As solved in the previous PSET-1 Q5, we know the time complexity is $\Theta(nlogn)$

3 Question 3

Q3(a) The cost of a pop operation on the stack where k=L/2 will be L/2 as we will need to move L/2 elements. As stated in the question, for the push operation it will cost L. The worst case would be a scenario where the stack is constantly resizing as we perform push and pop operations. Assume we start with an empty array. Every push operation will incur a cost of L and every pop operation will incur a cost of L/2. Assume that the n operations performed are even in number. $n/2$ of the operations are push operations and $n/2$ are pop operations. This means:
L * $n/2$ + L/2 * $n/2$
$\frac{3nL}{4}$is the total resizing cost. The amortized cost per operation would be:
$\frac{(3nL/4)}{n}$
= $\frac{3L}{4}$

Q3(b)

Q3(c) The size of the array increases by the number of times the resize operation is called. For example, when the resize operation is called once, the array increases by 1. When the resize operation is called twice, the array increases by 2 and so on. Thus, we can say that the cost of resizing is:
S_i = 1+2+3+....i
=$\frac{(i(i+1)}{2}$
where i represents the number of times the resize operation is called. To find the amortized cost per operation we must divide the total cost by number of operations. Let's also assume that a pushing an element also gives us a constant cost of 1 so our total cost is:
$\frac{(n(n+1)}{2}$ + n
where n is the push operations. The amortized cost per operation is:
($\frac{(n(n+1)}{2}$ + n)
= n+1
Thus amortized cost per operation is $\Theta(n)$

4 Question 4