50 marks **CS 218 Quiz 1** **8:30-9:25, 20/1/16**

Crisp answers will receive more credit. You need not repeat anything proved in the book or in class (just state as much), unless the question explicitly asks exactly that.

Problem 1:[20 marks] Suppose we have n numbers $x_1, \ldots, x_n$. We wish to determine the largest and the smallest of these using the minimum number of comparisons. Develop a divide and conquer algorithm for this. Use two way divide step. For simplicity assume that n is a power of two. State exactly (not O or θ) how many comparisons are needed as a fuction of n.

We write a recursive program which returns a pair of keys, the maximum and the minimum. We recurse on x[1..n/2] and x[n/2+1..n]. From each part we get its maximum and minimum. The combined maximum and minimum are obtained by comparing the maxima and the minima returned by the calls. Thus the combine step does two comparisons. Letting $C(n)$ number of comparisons for n keys, we have $C(n) = 2C(n/2) + 2$. 6 marks

The base case is $n = 2$, where a single comparison gives the maximum and the minimum. Thus $C(2) = 1$. (Having base case as $n = 1$ requires more comparisons.) 6 marks

Thus $C(n) = 2 + 2(2 + 2C(n/4)) = (2 + 4) + 2^2C(n/2^2) = 2 + 4 + 2^2(2 + 2C(n/8)) = (2 + 4 + 8) + 2^3C(n/2^3) = (2 + \cdot + 2^{k-1} + 2^{k-1}C(n/2^{k-1})$, where $k = \log n$. Thus $C(n) = n - 2 + n/2 = 3n/2 - 2$.

7 marks

A few people also counted the comparisons between the array index and the array length. There are simple ways to eliminate these, which must be used if you are to minimize the number of comparisons. For example you could write:

if(n == 4){ write out all the comparisons explicitly without looping }

else if(n == 8){ same }

else if(n == 16) { ... }

Tricks like this will work for any constant n, and so you would need loops only beyond that. But it will eliminate a huge number of comparisons.

Had you explained all this, I would have given full marks; otherwise I have deducted some marks.

Problem 2:[10 marks] We are given two sorted sequences, one of length 3 and another of length 4. They must be merged together using only comparison and copying operations. Let C denote the number of comparisons performed by the best possible algorithm (in the worst case). Give a lower bound on C. Justify your answer.

The final answer will interleave the two sequences. There are 7 choose 3 ways of doing this. Hence 5x6x7/6 = 35 ways. Thus there must be 35 leaves, and thus the height must be at least $\log_2 35 = 6$

which is the number of comparisons necessary.

Many people assumed that the standard mergesort is the best algorithm – it is not. Say you are merging a sequence of length n and a sequence of length 1. You would do binary search to decide where to insert the second length 1 sequence. This would need $\log n$ comparisons and not n that mergesort would need.

Problem 3: You have 3 sorted sequences $A[1..m], B[1..n], C[1..p]$. You wish to determine the rth smallest from the union of the 3 sequences. Call this an (m, n, p, r) instance. By making $O(1)$ comparisons it is possible to produce an (m', n', p', r') instance such that (a) the two instances have the same solution, (b) $r' \leq \alpha r$ for some constant α.

(a)[15 marks] State what comparisons you should make, and what the value of α is.

Compare the $A[r/3], B[r/3], C[r/3]$rd with each other. Suppose $A[r/3] < B[r/3] < C[r/3]$. Now we can be sure $A[1..r/3]$ cannot be the rth smallest. Thus we can throw out that part. Thus the new instance has $m' = m - r/3$, $n' = n$, $p' = p$, $r' = 2r/3$. Thus $\alpha = 2/3$.

A few people wrote that $r/3$ elements of B could also be removed. This is false – you could lose about 10 marks for this.

A few people compared $r/2$th elements. This does not work at all. You get a 0 for this.

(b)[5 marks] State how much time you will need to solve an (m, n, p, r) instance.

$T(r) = O(1) + T(2r/3)$. So $T(r) = O(\log_{3/2} r)$.