

Homework 5

Ryan Johnston

No Institute Given

CS 3500 Summer 2003

Problem 1

QUICKSORT' does exactly what QUICKSORT does, hence it sorts correctly. QUICKSORT and QUICKSORT' do the same partitioning, and then each calls itself with arguments A, p, q . QUICKSORT then call itself again, with arguments $A, q + 1, r$. QUICKSORT' instead sets $p = q + 1$ and reexecutes itself. This executes the same operation as calling itself with $A, q + 1, r$, because in both cases the first and third arguments (A, r) have the same values as before and p has the old value of $q + 1$.

b) The stack depth of QUICKSORT' will be $\Theta(n)$ on a n -element input array if there are $\Theta(n)$ recursive calls to QUICKSORT'. This happens every call to PARTITION(A, p, r) returns $q = r$. The sequence of recursive call in this scenario is :

QUICKSORT'(1,1, n), QUICKSORT'(1,1, $n-1$), QUICKSORT'(1,1, $n-2$), ... QUICKSORT'(1,1,1).

An array that is already sorted will cause the behavior.

c) The trick is to sort on the smaller sub-array first:

```
Alg QUICKSORT''(A,p,r)
1. while  $p < r$ 
2. do Partition and sort the small subarray first
3.    $q = \text{PARTITION}(A,p,r)$ 
4.   if  $q - p < r - 1$ 
5.     then QUICKSORT''(A,p,q-1)
6.      $p = q + 1$ 
7.   else QUICKSORT''(A,q+1,r)
8.      $r = q$ 
```

Since the size of the array is reduced by half on each call the stack depth is $\Theta(\lg n)$

Problem 2

a) This is obvious

b) Consider the decision tree associated with this algorithm. Notice that there are $\binom{2n}{n}$ leaves, one for each way you can divide the list in two. At each branch the algorithm has two options, i.e. one if the comparison determines \leq and another if $>$. So given the total height of the tree we know that $2^h \geq \binom{2n}{n}$. Notice we have the following

$$\sum_{k=0}^n \binom{n}{k} = 2^n.$$

So it must be that $\binom{2n}{n} \geq \frac{2^n}{n+1}$. Taking logs we obtain $h \geq 2n - \log(n+1)$

c) Assume that we have two lists $a_1 \dots a_n$ and $b_1 \dots b_n$. Assume that a_i, b_j are consecutive. Further assume that for every other pair the other is known, i.e. for each k and ℓ we know relationship between a_k and a_ℓ , but assume we don't know the relationship between a_i and a_j . We can see that both $a_i < b_j$ and $a_i \geq b_j$ would give a valid ordering.

d) Using the notation from c) above consider if the sorted order of our list was $a_1, b_1, a_2, b_2, \dots, a_n, b_n$ notice that by c) we would have to make $2n - 1$ comparisons.

Problem 3

a) For groups of 7, the algorithm still works in linear time. The number of elements greater than x (and similarly, the number less than x) is at least $4(\lceil \frac{1}{2} \rceil \lceil \frac{n}{7} \rceil - 2) \geq \frac{2n}{8}$, and the recurrence becomes

$$T(n) \leq T(\lceil \frac{n}{7} \rceil) + T(\frac{5n}{7} + 8) + O(n)$$

which can be shown to be $O(n)$ by substitution, as for the groups of 5 case in the text.

b) For groups of 3 however, the algorithm no longer works in linear time. The recurrence becomes

$$T(n) \leq T(\lceil \frac{n}{3} \rceil) + T(\frac{2n}{3} + 4) + O(n)$$

which does not have a linear solution. This can be proved by showing that the worst case time for groups of 3 is $\Omega(n \lg n)$, which can be done by deriving a recurrence for a particular case that takes $\Omega(n \lg n)$ time. In counting up the number of elements greater than x (and similarly, the number less than x), consider the particular case in which there are exactly $\lceil \frac{1}{2} \rceil \lceil \frac{n}{3} \rceil$ groups with

medians $\geq x$ and in which the leftover group does contribute 2 elements greater than x . Then the number of elements greater than x is exactly $2(\lceil \frac{1}{2} \rceil \lceil \frac{n}{3} \rceil - 1) + 1$ (the -1 discounts x 's group, as usually, and the +1 is contributed by the x 's group) $= 2\lceil \frac{n}{6} \rceil - 1$, and the recursive step for elements $\leq x$ has $n - (2\lceil \frac{n}{6} \rceil - 1) \geq \frac{2n}{3} - 1$ elements. This gives the recurrence

$$T(n) \geq T(\lceil \frac{n}{3} \rceil) + T(\frac{2n}{3} - 1) + \Theta(n) \geq T(\frac{n}{3}) + T(\frac{2n}{3} - 1) + \Theta(n)$$

from which you can show that $T(n) \geq cn \lg n$ by substitution. You can also see that $T(n)$ is nonlinear by noticing at each level of the recursion tree sums to n .

Problem 4: We first note that this problem generalizes both sorting and median finding: sorting is equivalent to finding the n th quantile, and finding the median is equivalent to finding the 1st quantile.

We make a simplifying assumption that $n = 2^i$ and $k = 2^j$ for some integers i and j .

FindQuantile(A, k)

0. Let $n = |A|$. If $k > n$, return FALSE.
1. Find the median m of A . If $k = 1$, return m .
2. Partition A around m . Let L and R be the left and the right subarrays after partitioning. Note that $|L| = |R| = (n - 1)/2$, where $n = |A|$.
3. Output FindQuantile($L, \frac{k-1}{2}$), m , FindQuantile($R, \frac{k-1}{2}$).

The correctness of the algorithm is clear. Let $T(n, k)$ be the running time of the algorithm. Then $T(n, k) = 2T((n - 1)/2, (k - 1)/2) + O(n)$. It's not hard to see that $T(n, k) = O(n \log k)$. In fact, FindQuantile(A, k) generalizes Quick Sort: it executes $\log k$ levels of recursion of Quick Sort.

For general n and k , the algorithm is similar, and some care is needed in finding the rank of the pivot element. Details are omitted.

Problem 5 a)

Alg 5a Input: array A and value x

1. Initialize an array B of length $|A|$ to be all 0s
2. Initialize an INTEGER variable *counter* and set it equal to 0
4. While *counter* $\neq |A|$ do
5. Randomly choose a value i in $0 \dots |A| - 1$.
6. If $A[i] = x$ then quit weve found it
7. Otherwise increment counter by 1 if $B[i] = 0$

8. set $B[i] = 0$
end while

b) The probability that x is found on the i^{th} run is $p^i(1-p)^{i-1}$ this is the geometric distribution, thus the from 1112 in your book you can see that this has expectation $\frac{1}{p} = n$

c) Same as above but now $p = \frac{k}{n}$. Thus the expectation is $p = \frac{n}{k}$.

d) This is equivalent coupon collector problem on p110 and the expected number of indices into A that must be picked is $n(\ln n + O(1))$

Problem 6

Given a matrix A of dimensions $4^n \times 4^n$, we define the matrix $A_{i,j}$ to be the submatrix of A of dimension $4^{n-1} \times 4^{n-1}$ whose upper-left hand element is $a_{4i,4j}$

We modify W. Strassens algorithm so that given a $4^n \times 4^n$, but instead of numbers it uses $4^{n-1} \times 4^{n-1}$ matrices, and instead of scalar multiplication and scalar addition it uses matrix multiplication and matrix addition. It should be clear that matrix addition takes $(m/4)^2$ scalar additions where $m = 4^n$. To accomplish matrix multiplication on each pair of $\frac{m}{4} \times \frac{m}{4}$ submatrices we recall W. Strassens modified algorithm.

Now to find the running time we solve the following recurrence:

$$32T\left(\frac{m}{2}\right) + 3500m^2 = \Theta(m^{\log_4 32}) = O(m^{2.5})$$