15-451: Algorithms

Recitation 3: Some Problems

1 k-th Element

Let A and B be two sorted arrays of n elements each. We can easily find the kth smallest element in A in $\mathcal{O}(1)$ time by just outputting $A[k]$. Similarly, we can easily find the kth smallest element in B. Give an $\mathcal{O}(\log k)$ time algorithm to find the kth smallest element overall – i.e., the kth smallest in the *union* of A and B. Give a proof that your algorithm works as advertised. For purposes of defining what we mean here, you may assume there are no duplicate elements.

1.1 Solution

Clearly the required kth smallest element lies within $A[1] \dots A[k]$ or $B[1] \dots B[k]$. Consider the following algorithm foo():

Input:

- a sorted array A
- an index I_A in A
- a sorted array B
- an index I_B in B
- an integer k

Output: The kth smallest element in the union of $A[I_A] \dots A[I_A+k-1]$ and $B[I_B] \dots B[I_B+k-1]$.

Clearly the solution to our problem is foo$(A, 1, B, 1, k)$.

The following is an implementation of foo():

```
int foo(int A[],int I_A,int B[],int I_B,int k)
{
   if(k == 2) return the second smallest among A[I_A],A[I_A + 1],
      B{I_B] and B{I_B + 1];

   int M_A = A[I_A + floor(k/2)];
   int M_B = B[I_B + floor(k/2)];

   if(M_A < M_B) {
      return foo(A,I_A + floor(k/2),B,I_B,ceil(k/2));
   } else {
```

```
        return foo(A,I_A,B,I_B + floor(k/2),ceil(k/2));
    }
}
```

1.2 Proof of correctness

The median of $2k$ elements can have at most k elements greater than it and at most k elements smaller than it. Clearly the smaller of M_A and M_B has at least k elements greater than it. So the median cannot be less than it. Similarly the larger of M_A and M_B has at least k elements less than or equal to it. So the median cannot be greater than the element next to it in the same array. Also since in the recursive call we have ignored ⌊k/2⌋ elements definitely smaller than the k-th smallest element, we must now find the ⌈k/2⌉-th smallest element. This completes the proof.

1.3 Running time

Let T(n) be the total number of calls when foo() is called with initial k = n. Then:

T(2) = 1 T(3) = 2 T(4) = 2

In general, T(n) = T(⌈n/2⌉) + 1

Claim. For $2^{k-1} < n \leq 2^k$, T(n) = k.

Proof. By induction on k.

True for k = 1, 2.

If $2^{k-1} < n \leq 2^k$, then $2^{k-2} < \lceil n/2 \rceil \leq 2^{k-1}$. So, by inductive hypothesis, T($\lceil n/2 \rceil$) = k-1. T(n) = T($\lceil n/2 \rceil$) + 1 = k.

So log(n) ≤ T(n) ≤ (log(n) + 1). Hence T(n) $\in \Theta(log(n))$).

2 Quicksort Pivot Methods

To simplify the discussion suppose we only sort permutations of $[n]$. Assume uniform distribution of the input sequences. Consider the method of choosing as pivot the larger of a_1 or a_2.

A. Determine the expected value of the length of the left sublist.

B. Solve the recurrence for the average case running time.

2.1 Solution

Also, let us assume that the two recursive calls issued by `qsort` are to `qsort(l,p-1)` and `qsort(p,r)`. Clearly, this can only slow down the algorithm. Let X_n be the random variable: length of left sublist. Set

$$p_i = Prob[X_n = i].$$

Then for $i = 1, \ldots, n-1$ we have $p_i = 2i/n(n-1)$, whence $p_i + p_{n-i} = 2/(n-1)$.

$$E[X] = \sum_{i<n} i p_i = 2/n(n-1) \sum_{i<n} i^2 = (2n-1)/3 \to 2/3n.$$

Thus in average case splitting is sufficiently close to middle to get logarithmic depth of recursion.

Now let $t(n)$ be the average running time of Quicksort on a list of length n. Then

$$t(n) = \begin{cases} c & \text{if } n \leq 1, \\ cn + \sum_{i<n}(t(i) + t(n-i))p_i & \text{otherwise.} \end{cases}$$

Hence we have

$$\begin{aligned} t(n) &= cn + \sum_{i<n} \big(t(i) + t(n-i)\big) p_i \\ &= cn + 1/2 \sum_{i<n} \big(t(i) + t(n-i)\big) p_i + \big(t(n-i) + t(i)\big) p_{n-i} \\ &= cn + 1/2 \sum_{i<n} t(i)\big(p_i + p_{n-i}\big) + t(n-i)\big(p_i + p_{n-i}\big) \\ &= cn + 1/(n-1) \sum_{i<n} t(i) + t(n-i) \\ &= cn + 2/(n-1) \sum_{i<n} t(i). \end{aligned}$$

and thus

$$\begin{aligned} (n-1)t(n) &= cn(n-1) + 2\sum_{i<n} t(i) \\ n \cdot t(n+1) &= cn(n+1) + 2\sum_{i\leq n} t(i) \\ n \cdot t(n+1) &= 2cn + (n+1)t(n) \\ t(n) &= 2c + n/(n-1) \cdot t(n-1). \end{aligned}$$

Now recall from above the recurrence with:

$a(n) = n/(n-1)$, $b(n) = n-1$ and $f(n) = 2c$.

Note that this equation has homogeneous solution $h(n) = n$. Thus $t(n) = h(n) \sum_{i<k(n)} f(b^i(n))/h(b^i(n))$.

Hence $t(n) = n \cdot \sum_{i<n} 2c/(n-i) = 2cn \cdot \sum_{i<n} i^{-1} = 2cn \cdot H_{n-1} = \Theta(n \log n)$.

3 Lower Bounds

Consider a sorted sequence $a_1, \ldots, a_n$ whose elements are taken from some ordered set A, and let $x \in A$.

Determine lower bounds for the following problems.

- Assume x appears in a. Determine its position in the sequence.
- Assume x does not appear in a. Determine the interval in the sequence that contains x. That is, return i when $a_i < x < a_{i+1}$, 0 when when $x < a_1$, and n when $a_n < x$.
- Lastly, make no assumptions about x and determine either its position, or the interval that contains x.

3.1 Solution

We have a binary tree with m leaves, requiring in the worst case at least $\lceil \log_2 m \rceil$ comparisons:

- $m = n$.
- $m = n + 1$.
- $m = 2n + 1$.