Solutions for Tutorial Sheet 1

August 26, 2013

1 Problem 3

A k-way merge can be done in the following ways.

A. Merge the k sequences pairwise in a binary tree structure. Time complexity is $O(n \log k)$, since at every level of the tree, the number of comparisons is $O(n)$.

B. Build a k-heap on the surviving smallest element of each of the k sequences. For each element output, we need to do extract-min and insert from the corresponding sequence (from where the smallest element was output). Time complexity is $O(n \log k)$, with $O(\log k)$ comparisons for each heap operation.

For k-way mergesort, the recurrence is

$$T(n) = kT(\tfrac{n}{k}) + O(n \log k)$$

The solution is $T(n) = O(n \log n)$, which is independent of k.

2 Problem 4

Use quicksort where you choose the median as splitter. This is also called partition sort.

Claim: If there are n_α elements with value α, one of the elements will be chosen as a splitter by depth $\log_2(n/n_\alpha)$ - 1 of the recursion tree of quicksort.

All the elements of n_α must lie in the same partition. At level i of the recursion, the size of a partition is at most $n/2^i$. So, $n_\alpha \leq \frac{n}{2 \cdot 2^i}$. Otherwise, value α will be majority and will be the median. So, $i - 1 \leq \log_2(n/n_\alpha)$, i.e. $i \leq \log_2(n/n_\alpha) - 1$.

Once α is chosen it will not participate in future comparisons, since only the elements strictly larger and strictly smaller are recursively shifted. So, the number of comparisons that each element with value α participates is $\log_2(n/n_\alpha)$.

3 Problem 5

$$A = A_1 A_2 A_3 A_4$$
$$B = B_1 B_2 B_3 B_4$$

Let $|A| = |B| = n = 4^k$. As is the case with divide by 2, we can write

$A \times B$
$= (A_1 2^{3l} + A_2 2^{2l} + A_3 2^l + A_4)(B_1 2^{3l} + B_2 2^{2l} + B_3 2^l + B_4)$
$= A_1 B_1 x^6 + (A_1 B_2 + A_2 B_1) x^5 + ... + A_4 B_4$, where $x = 2^{l.n/4}$

This is similar to polynomial multiplication and the coefficients do not depend on the value of x. Polynomial multiplication can be done using polynomial evaluation and interpolation in $O(d^2)$ "word" multiplication (say by Lagrange's formula) for a polynomial with degree d. Using the method of problem 10, we can write a recurrence

$$T(n) = O(d^{1.7})T(n/d) + O(n), d = 4$$

4 Problem 6

For this problem, we will write a recurrence without repeating the sampling (even when the splitter is not balanced). Let $T(n)$ be the running time for n elements (assuming that it is the worst case for all k). Let n' be the size of the recursive call. Then $T(n) = T(n') + O(n)$ where n' depends on the rank of the splitter. Taking expectation n both sides, we get

$$E[T(n)] = E[T(n')] + cn$$

where n' is a random variable that depends on the rank of the splitter which is itself a random variable, say $r \in_{\mathcal{U}} [1..n]$, i.e., uniformly distributed. Note that, $E[T(n')|r = i] \leq E[T(\max\{i, n-i\})]$ from the monotonicity of expected time complexity. Therefore using the law of conditional expectation, we can write

$$E[T(n)] \leq \frac{1}{n}\sum_{i=1}^{n} E[\max\{T(i), T(n-i)\}] + O(n),$$

Denoting $E[T(n)]$ by $\bar{T}(n)$, this can be rewritten as

$$\bar{T}(n) \leq \frac{2}{n}\sum_{i=1}^{n/2} \bar{T}(i + n/2) + O(n)$$

Verify that $\bar{T}(n) = \alpha n$ by induction for an appropriate $\alpha > 1$.

5 Problem 7

For any given n and k, we can calculate $i_1, i_2, ..., i_k$ such that $i_1 < i_2 < ... < i_k$ and $i_1, i_2, ..., i_k$ are ranks of elements that will induce the required partition. Choose $m = 2^l$ such that $m \leq k \leq 2m$. Now use the median algorithm recursively to find elements $z_1, z_2, ..., z_k$ with ranks $n/m, 2n/m, ..., kn/m$. This will take $O(nl) = O(n \log m) = O(n \log k)$ comparisons.

We can form the m partitions by binary search in $O(n \log m) = O(n \log k)$ time. Given ranks $i_1, i_2, ..., i_k$ we know exactly which parts contain the elements. Hence, in an additional $mO(n/m) = O(n)$ steps we can find the required elements.

6 Problem 8

If an element occurs more than $n/4$ times, then it must have any one or more of the following possible ranks: $n/4, n/2, 3n/4$. We can use $O(n)$ time selection to select the elements and verify their frequencies.

7 Problem 9

Let the alphabet size be n, so we denote the alphabet as $1, 2, \ldots, n$.

Let $a_1, a_2, \ldots, a_{\sqrt{n}}$ and $b_1, b_2, \ldots, b_{\sqrt{n}}$ be two randomly chosen sets of $\sqrt{n}$ numbers from $1, 2, \ldots, n$. Assume for simplicity that n is a perfect square.

Then, the strings to be sorted are $A_n = \{a_i b_j \mid 1 \leq i, j \leq \sqrt{n}\}$. Consider radix sort on the family of strings $\{A_n : n \text{ is a perfect square}\}$.

Observe that we have n strings in A_n.
LSB radix sort would take time $O(n+|\Sigma|) = O(n)$ for each of the two passes, and hence it'd take time $O(n)$.

MSB radix sort would create $\sqrt{n}$ buckets (one for each a_i) of size $\sqrt{n}$ each in the first pass. Hence, it'd take time $\Omega(|\Sigma|)$ in the second pass per bucket. Hence, it'd take time $\Omega(n\sqrt{n})$ in the second pass, which is asymptotically more than the time for LSB radix sort.

We can generalize this example for d-length strings, by taking $n^{\frac{1}{d}}$ numbers instead of $\sqrt{n}$. Since we'll have $n^{\frac{d-1}{d}}$ buckets at the final pass (as the number of buckets will be multiplied by $\frac{1}{d}$ at each of the d passes) each containing $n^{\frac{1}{d}}$ elements, we'll have a time-complexity of $\Omega(n^{\frac{d-1}{d}} * n) = \Omega(n^{2-\frac{1}{d}})$ ($\Omega(n)$ for sorting each bucket as before) for MSB radix sort, and $O(nd)$ for LSB.

8 Problem 10

$P_A(n)$
$= (a_{n-1}x^{\frac{n}{2}-1} + a_{n-2}x^{\frac{n}{2}-2} + ... + a_{\frac{n}{2}})x^{\frac{n}{2}} + a_{\frac{n}{2}-1}x^{\frac{n}{2}-1} + a_{\frac{n}{2}-2}x^{\frac{n}{2}-2} + ... + a_0$
$= P'_A(x)x^{n/2} + P''_A(x)$

$P_B(n)$
$= (b_{n-1}x^{\frac{n}{2}-1} + b_{n-2}x^{\frac{n}{2}-2} + ... + b_{\frac{n}{2}})x^{\frac{n}{2}} + b_{\frac{n}{2}-1}x^{\frac{n}{2}-1} + b_{\frac{n}{2}-2}x^{\frac{n}{2}-2} + ... + b_0$
$= P'_B(x)x^{n/2} + P''_B(x)$

$P_A(n)P_B(n)$
$= (P'_A(x)x^{n/2} + P''_A(x))(P'_B(x)x^{n/2} + P''_B(x))$
$= P'_A(x)P'_B(x)x^n + (P'_A(x)P''_B(x) + P''_A(x)P'_B(x))x^{n/2} + P''_A(x)P''_B(x)$

Using the same method as integer multiplication, we can rewrite the 4 product terms as 3 product terms of half the size (i.e. degree $\frac{n}{2} - 1$). This gives a running time of $O(n \log_2 3)$. Note that the values of the coefficients are no more than $n2^{O(log\ n)} = 2^{O(log\ n)}$, i.e. you need at most $O(\log n)$ bits. Hence, all the arithmetic can be done in $O(1)$ time.