Homework #4: Sorting models, hashing
Due Date: Tuesday, 14 May 2002 — **two weeks later this time**

Reading: Chapters 9, 12 in CLR, 8, 11 in CLRS.
Points: This homework will be worth 20% more points than the usual one-week homeworks.

Recall that *exercises* are for you to work out on your own; *problems* are to be handed in.

Exercise 4-1. Do Exercise 9.1–6 on page 175 of CLR, 8.1–4 on page 168 of CLRS.

Exercise 4-2. Do Exercises 9.3–3 and 9.3–4 on page 180 of CLR, 8.3–3 and 8.3–4 on page 173 of CLRS.

Exercise 4-3. Do Exercise 12.1–4 on page 221 of CLR, 11.1–4 on page 223 of CLRS.

Exercise 4-4. Do Exercise 12.3–2 on page 232 of CLR, 11.3–2 on page 236 of CLRS.

Problem 4-1. "Real" Cost of Sorting. [50 points]

As part of a homework assignment for your architecture class you have to select the "best" sorting algorithm to use on your classes' example computer. The relevant parts of the computer's architecture are the cache and the main memory. In particular, the machine's memory can be viewed as a large array partitioned into a number of *cache lines*, each containing s consecutive locations of the memory array. In other words the first cache line consists of the first s memory locations, the second cache line consists of the next s memory locations etc. The cache memory is a small fast memory that at any point in time contains a $\Theta(1)$ number of these cache lines. The cache lines that are not in the cache are maintained in main memory. Whenever a user program accesses a memory location in a cache line that is in the cache, the access occurs with no main memory activity. Such a memory access is considered to be fast. If the memory location is not in the cache, however, the cache line on which it resides replaces some other cache line currently in the cache. In other words, the cache line that is being replaced is copied out to main memory and the desired cache line is copied into the cache. This event is called a *cache miss*. Since a cache miss involves a reference to main memory, this type of memory access is considered to be slow.

You decide to select your sorting algorithm based on the running time analysis given of the various sorting algorithms discussed in CS161. However, after a while you realize that the analysis given in CS161 does not account for the fact that the main memory operations required by a cache miss can take orders of magnitude longer than memory

operations that only involve the cache. In order to account for the cost of the different memory operations you decide to analyze the sorting algorithms based on the number of cache misses they may require.

First you extend O-notation to two-variable functions, (so as to free yourself from having to give exact answers when counting main memory operations).

Specifically:

$$O(f(x,y)) = \{g(x,y) : \text{there exist positive constants } x_0,\ y_0, \text{ and } c \text{ such that } |g(x,y)| \leq cf(x,y) \text{ for all } x \geq x_0 \text{ and } y \geq y_0\}.$$

In answering the following questions, assume that the cache holds $\Theta(1)$ cache lines and that the n numbers to be sorted are stored in an array of contiguous memory locations. Your answers should generally be worst-case analyses expressed in terms of n, the number of values to be sorted, and s, the number of values that fit in a cache line. Justify your answers by showing your analyses.

(a) Show that insertion sort has $O(n^2/s)$ cache misses. Assuming that your code knows s, suggest a modification to insertion sort that achieves $O(n^2/s^2)$ cache misses.

(b) Show that the number of cache misses generated in merging two sorted arrays of m numbers apiece is $O(m/s)$. How many cache misses are required by merge sort when accessing the input array?

(c) How many cache misses are generated when *Heapify* is called on a node of height h? What is the total cost (in terms of cache misses) of *BuildHeap*? How many cache misses does heapsort generate? (Use the array implementation for heaps given in the book.)

(d) How many cache misses does the *Partition* procedure in the book require for access to the input array? Make an intelligent guess as to the average number of cache misses that quicksort requires. You need only provide an intuitive justification for your guess.

(e) Which sorting algorithm would you recommend to your boss and why?

Problem 4-2. Radix-exchange sort. [30 points]

Consider a sorting algorithm analogous to QUICK-SORT in which the partitioning is done on the basis of the most significant bit of the elements: if that bit is 1, the element goes into the right hand part of the table; if that bit is 0, the element goes into the left hand part of the table. Subsequent partition is based on the next significant bit of the elements and so on. This method is called RADIX-EXCHANGE sorting.

(a) Analyze the worst case running time of this algorithm on input of size n.

(b) Analyze the average case running time on input of size n. Assume that the elements to be sorted are in the set $S = \{0, 1, 2, \ldots, n = 2^t - 1\}$ and that any input sequence is equally likely. Is the average case behavior different from the worst case one? How important is it for the running time of our algorithm to keep the partition balanced at each step of the recursion?

(c) Which algorithm would you use to sort elements in S: RADIX-EXCHANGE or the RADIX-SORT presented in class?

Problem 4-3. Hashing To Disk Pages [40 points]

In an application that requires a very large hash table, it may be impractical to store the hash table in primary memory. In particular, one might choose to store the hash table on disk, with one disk page playing the role of one slot in the hash table. (For a review of disk storage, read pages 382–384 in CLR, 434–437 in CLRS.)

We presume that a disk page is large enough to hold many records. For example, a typical disk page holds 2^{12} bytes of data, and a record may contain only 2^5 bytes. We shall store the records that hash to a single page in a linear order on the page. If a page overflows because it contains too many records, the excess records are stored in an overflow area somewhere else on disk.

A SEARCH consists of hashing to the correct disk page and then linearly searching through the records on that page for one with the query key. If the page is full, then the overflow area must be searched in addition. Since the time to access a disk page is typically at least 10 milliseconds, the cost of the linear search on the page is neglible. Thus, we shall focus the number of disk accesses as our cost measure. For the SEARCH operation, the cost is 1 if we find the record in its "proper" page, but it may be considerably greater if we must in addition search the overflow area. Consequently, the focus of this problem is to ensure that the proper pages seldom overflow, while using as little extra space as possible.

We shall assume for the rest of this problem that we are hashing n keys to disk pages, where each disk page holds r records. We would like to know the number m of disk pages so that with high confidence, we can search for any of the n keys with a single disk access. Moreover, we would like $m = O(n/r)$ so that at most a constant fraction of the storage is wasted.

We shall make the assumption of simple, uniform hashing (see page 224 of CLR, 226 in CLRS). Also, Inequalities (6.9) and (6.22) in CLR or (C.5) and (C.18) in CLRS will be useful for solving this problem.

(a) For some disk page k chosen arbitrarily, argue that the probability that page k overflows is at most $\binom{n}{r}(1/m)^r$. Please be explicit about why this bound is an overestimate.

(b) Argue that the probability is at most $m\binom{n}{r}(1/m)^r$ that any of the m pages overflow.

(c) Argue that the probability is at most $m(en/mr)^r$ that any of the m pages overflow.

(d) Given any constant $\epsilon \geq ne/r2^{r-1}$, show that by choosing $m = 2en/r$, the probability is at most ϵ that any of the m pages overflow.

(e) Calculate a bound on the probability ϵ when hashing $n = 2^{20}$ records to $m = 2en/r$ pages, where each page contains $r = 2^7$ records. Is the constraint $\epsilon \geq ne/r2^{r-1}$ liable to be critical in practice?

(f) What is the largest load factor $\alpha = n/m$ that you can obtain with $n = 2^{20}$ and $r = 2^7$ so that the probability of a page overflowing is at most 1 in a billion? (*Hint:* Use the results of part (b) directly.)