CMSC 420
Fall 1993
Exam 3

1. [25 points] Sorting

a)[8] Explain the differences between radix sort and radix exchange sort.

b) [8] What is the worst case computational complexity of sorting by distribution counting for a set of n keys, each of which is an m bit number?

c) [9] Under what conditions, then, would sorting by distribution counting be preferable to an O(nlogn) sorting algorithm like quicksort?

2. [30] String matching

a)[15] Develop the state transition function for the string matching automaton for the pattern P = cacab. Draw the corresponding finite state machine. You can leave out all transitions back to the initial state - any transition not drawn will be assumed to go back to the initial state.

b)[5] List the sequence of states that the machine would go through in processing the string T = cababcacacacabab. Process the entire string T!

c [8] Show the result of applying the Rabin Karp algorithm, doing arithmetic modulo 7, in matching the string 132 against the pattern 7261321526723.

d) [2] In this discussion of Rabin Karp, we remarked that if we are matching patterns from alphabets containing d symbols, then we want to choose a modulus q such that dq is a close as possible to the size of the largest integer representable in a single word. Since d is given, this involves making q as large as possible. Why would we want q to be large rather than small?

3. [30] Storage allocation

a) [15] Draw a picture of memory, indicating which parts are free (and where the free block **boundaries** are) and in use, when the buddy system processes the following sequence of requests. There are 8 requests, and you should show memory after every other request (i.e., after the 2'nd, 4'th, 6'th and 8'th). Assume there are initially 32 words of free memory.

a <-- allocate 7
b <-- allocate 2

c <-- allocate 5
d <-- allocate 1
free b
e <-- allocate 3
free a
free d

b) [10] One drawback of **compaction** (compression) algorithms is that they may perform a large number of memory transfers in order to reclaim only a small amount of free storage. For example, suppose that when compaction is initiated, there is a very small free block at the top of memory, but then a large block immediately following it that is in use. In order to reclaim that free block, we must move all of the words in the following large block up just a few words in memory (and then perhaps move more contiguous blocks in use up only a small distance in memory).

An alternative algorithm might search the list of blocks in use following the free block, trying to find one that would fit into the free block (perhaps with a small amount of internal fragmentation). If successful, we move that block up to occupy the old free positions, and can move down in memory to the next free block and try to fill it with another subsequent block in use. If we fail, then we perform ordinary memory compaction from that point forward, to finally reclaim a large block of free storage at the end of the heap.

Criticize this algorithm from the point of view of a) computational efficiency and b) memory fragmentation, comparing it to the compaction algorithm presented in class.

c) [5] In the boundary tag method, we store the size of a block at the end of each block, duplicating the information stored at the beginning of the block. Is there any practical advantage to storing the **size** of the block rather than the **start address** of the block?

4. [15] Heaps

a) [10] Show the heap that results from the following sequence of insertions into an initially empty heap :10, 15, 22, 4, 11, 23, 19, 14. Show the heap after the insertion of **each** element.

b) [5] Show the result of deleting the key stored at the root of the resulting heap.