

The first two problems constitute a short homework, basically half of a usual homework, and problems 3 and 4 are **extra credit**. This homework is due in the (morning) class, Thursday, Oct. 2, and **I can't accept any late homeworks this time** because we will hand out model solutions in the discussion section that afternoon so you can use these to prepare for the first test.

Recall the bucketSort and radixSort algorithms presented in class on Sept. 18. I include them here for completeness because they are not in the book.

### bucketSort ( $S$ )

*Input:* sequence  $S$  of  $n$  items with keys in range  $[0, \dots, N - 1]$

*Output:* same sequence, sorted

1. Let  $B$  be an array of  $N$  sequences, each initially empty
2. **for** each item  $x$  in  $S$ , move  $x$  from  $S$  to end of  $B[\text{key}(x)]$ .
3. **for**  $i = 0$  to  $N - 1$  move sequence  $B[i]$  to end of  $S$ .

We proved in class that bucketSort is a correct stable sort taking time  $O(n + N)$ . Recall that **stable** means that for any two items  $a$  and  $b$  with the same key, if  $a$  comes before  $b$  in  $S$  then  $a$  comes before  $b$  in bucketSort( $S$ ).

### radixSort ( $S$ )

*Input:* sequence  $S$  of  $n$  items with keys  $d$ -tuples in range  $[0, \dots, N - 1]$

*Output:* same sequence, sorted

1. Let  $B$  be an array of  $N$  sequences, each initially empty.
2. **for**  $i = d$  downto 1 call bucketSort( $S$ ) using digit  $i$  as key.

We proved in class that radixSort is a correct sorting algorithm that takes time  $O(d(n + N))$ . Thus, if  $d$  is a fixed constant, e.g., 2 or 3, then radixSort sorts in linear time.

### Problems:

1. [25 pts.]
  - (a) Do problem 2.12, page 73, i.e., answer in big Oh notation by using the Master Theorem.
  - (b) Now also compute the exact answer, i.e., the exact number of lines printed, as a function of  $k$ , for inputs of the form  $n = 2^k$ .
  - (c) Prove by induction that your answer to (b) is correct.

2. [25 pts.] [Problem 2.13, page 73]: A binary tree is **full** if all of its vertices have either 0 or 2 children. Let  $B_n$  be the number of full binary trees with exactly  $n$  vertices.
- Draw out all full binary trees with 1, 3, 5, and 7 vertices, thus determining by hand  $B_1, B_3, B_5$ , and  $B_7$ . [It's important that you get the right answer here to be able to proceed, so I'll give away that  $B_1 = B_3 = 1$  and  $B_5 = 2$ .] Show that for all  $k \in \mathbf{Z}^+$ ,  $B_{2k} = 0$ .
  - For general  $k$ , derive a recurrence relation for  $B_{2k+1}$  in terms of  $B_1, B_3, \dots, B_{2k-1}$ . Argue why your recurrence equation is correct. You do not need to solve your recurrence equation.
  - Prove by induction on  $k$  that  $B_{2k+1} = 2^{\Omega(k)}$ . What this means is that there is a function  $f(k) = \Omega(k)$  such that for all  $k$ ,  $B_{2k+1} \geq 2^{f(k)}$ . From your recurrence equation in (b) you should come up with such a function  $f(k) = \Omega(k)$  and prove by induction on  $k$  that for all  $k$ ,  $B_{2k+1} \geq 2^{f(k)}$ .
3. [25 pts.] Do problem 2.17, page 74. As always in such problems you must argue clearly why your algorithm is correct and why its running time is what you claim it is.
4. [25 pts.] The running time for radixSort is analyzed above to be  $O(d(n + N))$ . This can be wasteful if  $d$  is large and the input consists of strings of symbols from the alphabet  $\{0, \dots, N - 1\}$  of maximum length  $d$ , but whose average length is much less than  $d$ . Give an algorithm that sorts a list of such strings in lexicographic, i.e., dictionary, order, e.g.,  $a < aab < ac < aca < b$ . Your algorithm should run in  $O(L)$  time, where  $L$  is the total length of all the strings. Thus this sort is linear in the total number of input characters.
- [Hint: start with radix sort. Observe that this sometimes wastes time by (a) manipulating  $B[i]$  when there are no characters  $i$  in a particular run; and, (b) examining and moving a string  $x$  in iteration  $i$  when  $x$  is too short to have the digit being considered at iteration  $i$ . The idea is that you can do an  $O(L)$  preprocessing phase which will precompute all the relevant information so the above two kinds of wasteful steps are eliminated. This preprocessing phase can be accomplished using a single Radix sort of the data
- $$\{(p, i, j) \mid \text{Symbol } i \text{ occurs in position } p \text{ of string } j\} \quad . \quad ]$$