Homework 4 (80 points), Spring 2003

Q1: (10 points)

Given an input array $A = < 13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21 >$, what is the resulting sequence of numbers in A after making a call to `Partition(A, 1, 12)`.

- The resulting array is $A = < 13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21 >$.

Q2: Exercise 7.2-2 (10 points)

What is the running time of `Quicksort` when all elements of array A have the same value?

- If all elements are the same, the quick sort partition return index $q = r$. This means, the problem with size n is reduced to one subproblem with size $n-1$. So the recurrence is $T(n) = T(n-1) + n$. By iteration method, $T(n) = \Theta(n^2)$.

Q3: Exercise 7.2-3 (10 points)

Show that the running time of `QuickSort` is $\Theta(n^2)$ when the array A contains distinct elements and is sorted in decreasing order.

- In each partition, the pivot element is always the smallest element. Therefore, each partition will produce two subarrays. The first subarray contains only one element (the smallerest element) and this element is in its correct position. The second subarray contains the remaining elements.
  In this case, the running time recurrence will be $T(n) = T(n-1) + n$. Thus, $T(n) = \Theta(n^2)$.

Q4: Exercise 7.3-2 (10 points)

During the running of the procedure`Randomized-Quicksort`, how many calls are made to the random-number generator `Random` in the worst case? How about in the best case? Give your answer in terms of Θ-notation.

- Let $N(n)$ be the number of calls to `Random` when running `randomized-Quicksort` on an array with size n. Let's try to define $N(n)$ recursively. Two extreme cases to consider as follows.
  case 1: always partition into two subproblems with equal sizes.
  Then $N(n) = 2N(n/2) + 1$ and $N(n) = \Theta(n)$.
  case 2: Always partition into one subproblem with one less element.
  Then $N(n) = N(n-1) + 1$ and $N(n) = \Theta(n)$.

- Two extreme cases results in the same number of calls. This means there is no best and worst cases. All cases need $\Theta(n)$ calls to `Random`.

Q5: (10 points)

Please draw a decision tree for sorting an input array $A = < a_1, a_2, a_3 >$ using the `QuickSort` algorithm.

- Shown in figure 1.

Q6: Exercise 8.2-3 (10 points)

Suppose that the `for` loop header in line 9 of the `Counting-Sort` procedure is rewritten as

```
9  for j <-- 1 to length[A]
```

Show that the algorithm still works properly. Is the modified algorithm stable?

- We show this by an example. Suppose that there are two elements a_1 and a_2 with the same value and a_1 appears before a_2 in the input array A. In the original algorithm, if a_2 is put into the output array B at position x, i.e., put a_2 into $B[x]$, then a_1 will be put into $B[x-1]$. But in the modified algorithm, a_2 is in $B[x-1]$ and a_1 is in $B[x]$. They are reversed in the output. For the purpose of sorting, the reversing is ok because $a_1 = a_2$. So the modified algorithm works properly.
- From the above example, we know that the modified algorithm is not stable.

Q7: Exercise 8.2-4 (10 points)

Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and answers any query about how many of the n integers fall into a range $[a..b]$ in $O(1)$ time. Your algorithm should use $\theta(n+k)$ preprocessing time.

- Use line 1 to line 7 of the Counting-Sort algorithm (pp. 168) to preprocess the n integers in the range 0 to k.
  After the preprocessing, we have an array $C[0..k]$ in which each $C[i]$ contains the number of integers less than or equal to i. Obviously, this preprocess takes $\Theta(n+k)$ time. Let Query(a, b) be an algorithm to return the number of the n integers fall into a range [a..b] in $O(1)$ time. The pseudocode for Query(a, b) is as follows.

```
Query(a, b)
{
 if (b < a)
    then return error ''invalid range''
 if (a > 0)
    then low-index = a - 1
    else low-index = -1
 if (b < k)
    then high-index = b
    else high-index = k
 if low-index = -1
    then return C[high-index]
    else return C[high-index] - C[low-index]
}
```

Exercise 8.3-1 (10 points)

Using the figure 8.3 as a model, illustrate the operation of `Radix-Sort` on the following list of English words: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.

- Shown on figure 2.

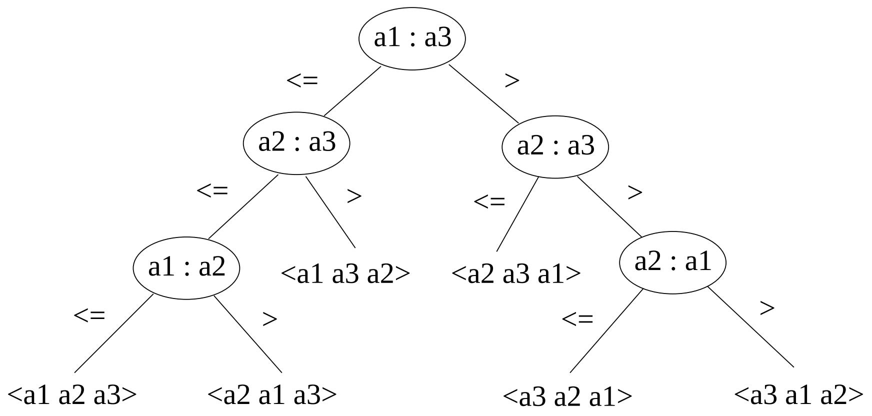

Figure 1: A decision tree for quick sort

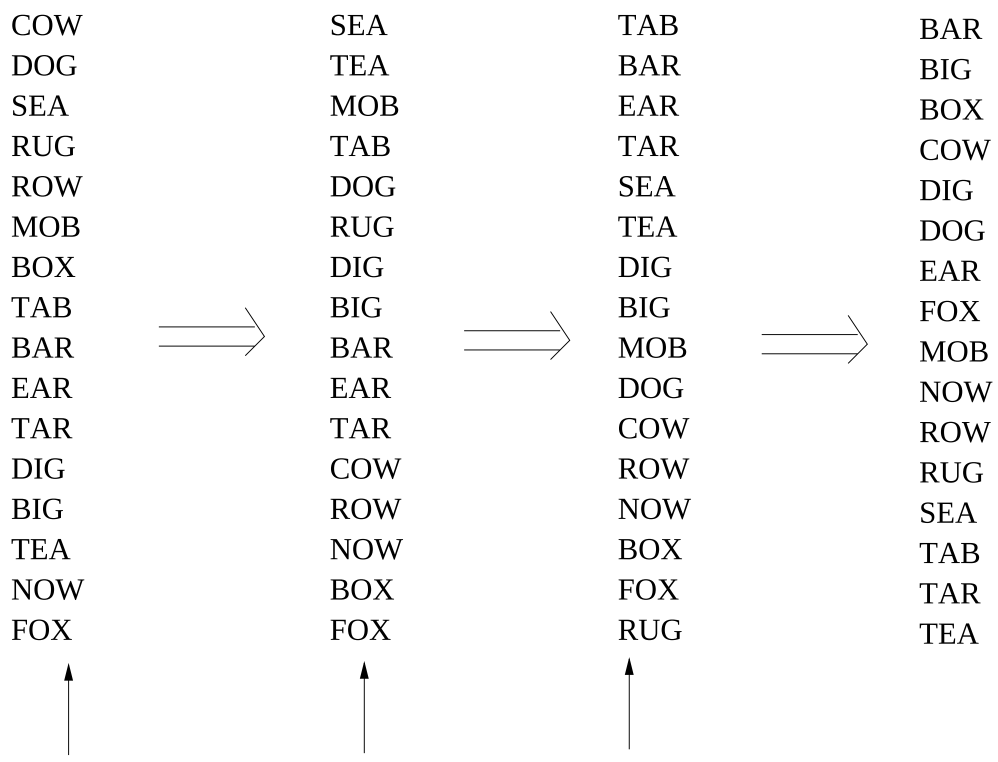

Figure 2: An example of Radix-Sort