CSE326 Homework #2

Due: Wednesday, July 6

1. Suppose we have an unsorted array $A[1 \ldots n]$ of integers with possible duplicates. Design a version of Quicksort that instead of partioning into two sets, one whose elements are less than or equal to the pivot and a second whose elements are greater than or equal to the pivot, the new algorithm partitions into three sets, one whose elements are strictly less than the pivot, a second whose elements are sticktly more than the pivot, and a third whose elements are equal to the pivot. Your algorithm should be in-place. One idea is that in the partitioning phase as we move the two pointers i and j toward each other we maintain the invariant that the array looks like:

   [elements equal to pivot] [elements less than pivot] [unknown elements] [elements greater than pivot] [elements equal to pivot]

   When there are no unknown elements left then the elements can be rearranged to be of this form:

   [elements less than pivot][elements equal to pivot][elements greater than pivot]

   (a) Design the Quicksort and Partition algorithms that implements this idea.

   (b) Show that your Quicksort algorithm runs in worst case time O(dn) where d is the number of distinct keys in the array.

2. There are two strategies for handling small arrays in Quicksort. The first strategy is the one described in class. Apply Quicksort recursively until the array are smaller than a CUTOFF size, then call Insertionsort to sort the small array. A second strategy is to apply Quicksort recursively unitl the array is smaller than the CUTOFF, then return. In this strategy after Quicksort(A[],1,n) is completed, the array A[1,n] is almost sorted. Now call Insertionsort(A[],1,n) to finish the job. Since A[1..n] is almost sorted then Insertionsort should do a good job.

   (a) Explain why the number of comparisons executed by Insertionsort in the two strategies is approximately the same. Which strategy uses slightly more and why?

(b) Explain why the second strategy executes fewer instructions than the first.

(c) Explain why the second strategy has more cache misses than the first when n is very large.

(d) If 8 integers fit in a memory block and n is very large, then about how many more cache misses are there in the second strategy as compared to the first strategy. Explain your answer.

3. Suppose you are given as input n positive integers and a number k. Write an algorithm to determine if there are any four of them, repetitions allowed, that sum to k. Your algorithm should run in time $O(n^2 \log n)$. Partial credit will be given if your algorithm is correct but takes longer than $O(n^2 \log n)$. As an example, if $n = 7$, the input numbers are 6, 1, 7, 12, 5, 2, 14 and $k = 15$, the answer should be yes because $6+5+2+2 = 15$.

   Hint #1: First solve the simpler problem that determines if there are any two numbers that sum to k.

   Hint #2: Sum of four numbers is the sum of two pairs of numbers.