CS 430 Introduction to Algorithms	Fall Semester, 2016

Homework 3

Assigned: Sept. 28 | *Due: Oct. 17*

Please respect the following guidelines for writing pseudocode:

1. C instructions are fine. But do not write object-oriented additions. Do not declare or use any class. Declare only procedures (if necessary) and explain in words what each procedure does, and what is the use of each parameter.
2. One instruction per line
3. Match the brackets with a horizontal line
4. Number your lines
5. Write down if your array is indexed $0 \ldots n-1$ or $1 \ldots n$.

Problem 1 Suppose you are given two n-element sorted sequences A and B, each representing a set (none has duplicate entries). Describe an $O(n)$-time method for computing a sequence representing the set $A \setminus B$ (with no duplicates; note: **the difference** of sets A and B consists of those elements in A and not in B).

You do not have to argue correctness (but, obviously, your method must be correct), but must justify the running time.

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

Problem 3 Let A and B be arrays of n integers each (do not assume they are sorted). Given an integer x, describe a $O(n \log n)$-time algorithm for determining if there is an integer a in A and an integer b in B such that $x = a + b$.

Present pseudocode and analyze the running time.

Problem 4 Describe an O(n)-time algorithm that, given a set S of n distinct numbers and a positive integer $k \leq n$, determines the k numbers in S that are closest to the median of S.

Assume n is odd and the set S is given as an **unsorted** array of size n. You cannot assume the input array is sorted.

Example: if $S = \{1, 3, 5, 9, 13, 21, 101\}$ and $k = 4$, the solution is $\{3, 5, 9, 13\}$. That is, the median itself is included. The answer $\{5, 9, 13, 21\}$ is not correct since 3 is closer to the median (which is 9) than 21. The algorithm should write the output in a separate array, and the numbers **do not** have to be sorted.

You can use the selection algorithm as a subroutine. Precisely, assume that the following procedure is given: SELECT(A,p,q,i) returns (finds) the index j such that $A[j]$ is the i^{th} number among $A[p], A[p+1], \ldots, A[q]$. SELECT correctly runs in time $O(q-p)$ even if the elements of A are not distinct.

Partial credit will be given to correct algorithms, but with larger running time.

Problem 5 Characterize each of the following recurrence equations using the master method (assuming that $T(n) = c$ for $n < d$, for constants $d \geq 1$).

a. $T(n) = 2T(n/2) + (n \log n)^4$

b. $T(n) = 2T(n/2) + \log^2 n$

c. $T(n) = 9T(n/3) + n^2$

d. $T(n) = 9T(n/3) + n^3$

e. $T(n) = 7T(n/2) + n^2$

```
//                                 QUICKSORT

void swap ( int* ia, int i, int j )
{ // swap two elements of an array
     int tmp = ia[ i ];
     ia[ i ] = ia[ j ];
     ia[ j ] = tmp;
}

void qsort( int* ia, int low, int high )
{ // quick sort
  // stopping condition for recursion
     if ( low < high ) {
          int lo = low;
          int hi = high + 1;
          int elem = ia[ low ];

          for (;;) {
              while ( ia[ ++lo ] < elem ) ;
              while ( ia[ --hi ] > elem ) ;

              if ( lo < hi )
                 swap( ia, lo, hi );
              else break;
           }  // end, for(;;)

        swap( ia, low, hi );

        // Recursive calls:
        qsort( ia, low, hi - 1 );
        qsort( ia, hi + 1, high );
        } // end, if ( low < high )
}
```