CS430 HW1:

1. 2.2-2:

***2.2-2***

Consider sorting n numbers stored in array A by first finding the smallest element of A and exchanging it with the element in AŒ1 . Then find the second smallest element of A, and exchange it with AŒ2 . Continue in this manner for the first n 1 elements of A. Write pseudocode for this algorithm, which is known as ***selection sort***. What loop invariant does this algorithm maintain? Why does it need to run for only the first n 1 elements, rather than for all n elements? Give the best-case and worst-case running times of selection sort in ‚-notation.

2. (2 points) Use mathematical induction to show that when n is an exact power of 3, the solution of the recurrence

is T(n) = n^2

3. (4 points) Problem 2-1: Insertion sort on small arrays in merge sort

***2-1 Insertion sort on small arrays in merge sort***

Although merge sort runs in ‚.nlgn/ worst-case time and insertion sort runs in ‚.n2/ worst-case time, the constant factors in insertion sort can make it faster in practice for small problem sizes on many machines. Thus, it makes sense to ***coarsen*** the leaves of the recursion by using insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge sort in which n=k sublists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.

1. ***Show that insertion sort can sort the n=k sublists, each of length k, in ‚.nk/ worst-case time.***
2. ***Show how to merge the sublists in ‚.n lg.n=k// worst-case time.***
3. ***Given that the modified algorithm runs in ‚.nk C n lg.n=k// worst-case time, what is the largest value of k as a function of n for which the modified algorithm has the same running time as standard merge sort, in terms of ‚-notation?***
4. ***How should we choose k in practice?***

4. (3 points) Consider the following program and recursive function. void main() {

int A[3]={1,2,3};

Z(A, A.length, 0); }

void Z(int A[], int n, int k) { if(k==n-1) {

for (int i=0; i<n; i++) cout << A[i] << " ";

cout << endl; }

else { for (int i=k; i<n; i++) {

swap(A[i], A[k]); Z(A, n, k+1); swap(A[i], A[k]);

} }

}

4a. Demonstrate the execution, show the output, and explain what the program accomplishes.

4b. Give a recurrence equation describing the worst-case behavior of the program.

4c. Solve the recurrence equation.

1. (6 points) Give big-O bounds for T(n) in each of the following recurrences. Use induction, iteration or Master Theorem. You may assume T(1)=1 in all cases.

5a. T(n) = T(n-1) + n^2

5b. T(n) = 5T(n/3) + n\*n^(1/2)

5c. T(n) = T(n/4) + T(n/2) + n^2

6. (6 points) Problem 4-2: Parameter-passing costs

***4-2 Parameter-passing costs***

Throughout this book, we assume that parameter passing during procedure calls takes constant time, even if an N -element array is being passed. This assumption is valid in most systems because a pointer to the array is passed, not the array itself. This problem examines the implications of three parameter-passing strategies:

1. An array is passed by pointer. Time D ‚.1/.
2. An array is passed by copying. Time D ‚.N /, where N is the size of the array.
3. An array is passed by copying only the subrange that might be accessed by the called procedure. Time D ‚.q p C 1/ if the subarray AŒp : : q is passed.
4. ***Considertherecursivebinarysearchalgorithmforfindinganumberinasorted array (see Exercise 2.3-5). Give recurrences for the worst-case running times of binary search when arrays are passed using each of the three methods above, and give good upper bounds on the solutions of the recurrences. Let N be the size of the original problem and n be the size of a subproblem.***
5. ***Redo part (a) for the MERGE-SORT algorithm from Section 2.3.1.***