Name ______________________________ ID No. ______________

There are 200 points possible.

1. (30 pts) Suppose you have a computer that requires, on average, one second to solve problem instances of size $n = 100$. Assuming memory and storage are not a problem, how long would it take, on average, to solve a problem ten times as large ($n = 1,000$) in each of the following situations? (Recall that $A(n)$ is the average-case time for a problem of size n.)

   (a) $A(n) = Cn$ for some constant C.

   Ans. $C = A(100)/100$ and $A(1,000) = 1,000C = 10\,A(100) = 10$ seconds.

   (b) $A(n) = Cn^2$ for some constant C.

   Ans. $C = A(100)/(100)^2$ and $A(1,000) = C(1,000)^2 = 100\,A(100) = 100$ seconds.

   (c) $A(n) = C\,2^n$ for some constant C.

   Ans. $C = A(100)/2^{100} = 2^{-100}$ and $A(1,000) = 2^{-100}2^{1,000}$ seconds. This is $2^{900} \approx 10^{270}$ seconds. For comparison, the age of the universe is estimated to be around 10^{18} seconds.

2. (30 pts) We have two algorithms for a problem.

   - The average run time for Algorithm A is **much better** than the average run time for Algorithm B.
   - The worst-case run time for Algorithm A is **much worse** than the worst-case run time for Algorithm B.
   - The two algorithms require the same amount of storage and are equally difficult to program correctly.

   The following examples should be rather specific, as in "reordering student names according to GPAs" **but not** "sorting a list."

   (a) Give an example of a situation where **Algorithm A** should be used rather than Algorithm B. Explain BRIEFLY why it should be used.

   Ans. Any situation in which the algorithm is run over and over on problems that appear to be random, and all that matters is the total running time. You could come up with many situations. One example is sorting students in classes according to their grades to produce rankings for all classes. Average time matters since that will give us a good estimate for how much computer time will be required.

   (b) Give an example of a situation where **Algorithm B** should be used rather than Algorithm A. Explain BRIEFLY why it should be used.

   Ans. Any situation where it is critical to get the answer within a certain time every time the algorithm is run. Again, you could come up with many situations. One example is controlling a robot that is moving in a dangerous environment. (It must avoid walking off a cliff, falling, etc.) Average time doesn't matter since one decision that takes too long could result in disaster for the robot.

3. (30 pts.) I have found two divide and conquer algorithms for a problem I want to solve. I tell you that all running times increase with n, the problem size, and also:
   - the average time for Algorithm 1 satisfies $A_1(n) = 2A_1(n/2) + 3n$ when n is a power of 2 and
   - the worst time for Algorithm 2 satisfies $W_2(n) = 5W_2(n/3) + n$ when n is a power of 3.

   (a) Determine the complexity categories of A_1 and W_2.

   Ans. Using Theorem B.6 we have

$$\begin{aligned} A_1(n) &\in \Theta(n \lg n) \qquad \text{with} \quad a = 2, \quad b = 2, \quad k = 1; \\ W_2(n) &\in \Theta(n^{\log_3 5}) \qquad \text{with} \quad a = 5, \quad b = 3, \quad k = 1. \end{aligned}$$

   (b) I ask you which algorithm is better for large problems. What is your answer? Why?

   Ans. Since $\log_3 5 > 1$, $n \lg n \in o(n^{\log_3 5})$ and so, by (a) the average-case time for Algorithm 1 is much better than the worst-case time for Algorithm 2. This is not enough information to decide; for example, the average-case time for Algorithm 2 may be much less than $A_1(n)$.

   (c) A few minutes later, I return and apologize because **I gave you the wrong equations**. I had reversed average case and worst case. The correct recursions are

$$W_1(n) = 2W_1(n/2) + 3n \qquad \text{and} \qquad A_2(n) = 5A_2(n/3) + n.$$

   What is your answer to (b) now? Why?

   Ans. These are the same equations except worst and average have been interchanged. Now the average-case time for Algorithm 2 is worse than the worst-case time for Algorithm 1. Thus Algorithm 1 is better regardless of whether we are interested in worst or average times.

4. (20 pts.) Complete the following sentences with a word or brief phrase.

   (a) If it is possible to design a divide and conquer algorithm for a problem, ONE important factor in whether or not the running time will grow at a reasonable rate as the problem size grows is

   Ans. (1) the number of subproblems one problem is divided into; (2) the size of the subproblems; (3) how much time is required to combine the subproblem results to solve the original problem

   (b) Suppose it is possible to design a backtracking algorithm for a problem. There are usually various choices to be made when setting up the algorithm. ONE choice that can significantly affect the running time is

   Ans. (1) the order in which things are considered (e.g., order of objects in the 0-1 knapsack problem); (2) the order in which decisions are made about each thing (e.g., include before leave out in the 0-1 knapsack problem); (3) what is taken into account in the `promising( )` function.

5. (30 pts) Consider the following algorithm:

```
TRANS(lo, hi) {
    if (1 == hi-lo) return;
    mid = (hi+lo)/2;
    TRANS(lo, mid);
    TRANS(mid, hi);
    for (i=lo; i<mid; i=i+1) {
        t = w[i] + w[i+mid];
        w[i+mid] = w[i] - w[i+mid];
        w[i] = t;
    }
}
```

The algorithm is used when `n` is a power of 2. One invokes the algorithm by `TRANS(0,n)`. It uses an `n`-long external array of numbers `w`. Assume that executing one step of the `for` loop is a basic operation.

(a) What algorithm category(e.g., backtracking) does it belong in and why?

Ans. Divide and conquer because either (a) it is top down or (b) it divides the problem into smaller problems of the same time and combines results.

(b) Using induction on m prove that the number of basic operations in `TRANS(0,m)` is the same as the number in `TRANS(j,m+j)` for all j. (You may assume that m is a power of 2.)

Ans. Since `TRANS(j,j+1)` takes zero basic operations for all j, this does $m = 0$.
Now for induction. Suppose $m > 0$. We have `mid =(j+m+j)/2 = j+m/2`.
• First the recursive part of the code: By induction, `TRANS(j,j+m/2)` and `TRANS(0,m/2)` require the same number of basic operations. By similar reasoning, so do `TRANS(j+m/2,j+m)` and `TRANS(0,m/2)`, and also `TRANS(m/2,m)` and `TRANS(0,m/2)`. Thus so do `TRANS(j+m/2,j+m)` and `TRANS(m/2,m)`.
• Now the nonrecursive part of the code: In `TRANS(0,m)` and `TRANS(j,m+j)` the `for` loop is executed m times.
• Combining these observations, we see that the number of basic operations is the same.

(c) Write a recursion for the every-case time complexity of the algorithm. *Do NOT solve the recursion.*
HINT: Use the result from (b). You can do this even if you have not done (b).

Ans. Let $T(n)$ be the number of basic operations when n is a power of 2. We have $T(n) = 2T(n/2) + n/2$.

6. (60 pts.) Indicate whether true or false. Beware of guessing:

correct answer +4pts. incorrect answer −2pts. no answer 0pts

(a) **T** If $f(n) \in \Theta(g(n))$, then $g(n) \in \Theta(f(n))$.

(b) **F** If $f(n) \in o(g(n))$, then $g(n) \in o(f(n))$.

(c) **T** If $f(n) \in o(g(n))$, then $g(n) \notin o(f(n))$.

(d) **F** If $f(n) \in O(g(n))$, then $g(n) \notin O(f(n))$.

(e) **F** Divide and conquer algorithms use a bottom up approach.

(f) **T** If a divide an conquer algorithm requires recomputing the same quantity many times, it is a good idea to look for a dynamic programming algorithm.

(g) **T** Quicksort has a good average run time and a poor worst-case run time.

(h) **F** Although it requires more complicated data structures, Prim's algorithm for a minimum spanning tree is better than Kruskal's when the graph has a large number of vertices.

(i) **T** Monte Carlo algorithms can be used to estimate the run times for some backtracking programs.

(j) **F** The complexity category of a backtracking program such as n-queens can be determined by a Monte Carlo algorithm.

(k) **F** If you can devise a simple backtracking algorithm for a problem, you should use it since no other algorithm is likely to be faster.

(l) **T** It is impossible to design a sorting algorithm based on comparison of keys whose worst-case run time is in $\Theta(n)$.

(m) **T** It is impossible to design a search algorithm based on comparison of keys of items in a sorted list such that the worst-case run time requires at most $\log_{10} n$ comparisons for large n.

(n) **F** For most problems, it is fairly easy to obtain lower bounds for run-time complexity that are close to the times of the best known algorithms for the problems.

(o) **T** For many problems, the best known algorithms require keeping track of data that was not asked for in the problem.