**CS 310 Exam One Terms and Concepts**

Last Modified: 29 January 2022

You should know or be able to do:

* The powers of two from zero to 1,000. The names and approximate values of 210, 220, 230, and 240.
* Draw labeled diagrams of memory corresponding to C++ programs. Specifically, immediate variables, pointer variables, and reference variables.
* The three parameter passing modes used in C++, pass by value, pass by reference, and pass by constant reference. How they are diagrammed, when and why each is used.
* Draw diagrams of the memory spaces of called and calling scope when one function calls another.
* Draw diagrams of heap memory when dynamic allocation is used.
* Given a diagram of memory, write a program which matches the diagram.
* State and justify the input size and basic operations of an algorithm.
* Analyze an algorithm containing loops, conditionals, and function calls.
* For each rule in the style guide, what purpose that rule serves.

You should also be familiar with the LaTeX constructs in the templates I have shown you. In particular, be able to properly use the inline and display math modes, and the align\* environment.

Some problems that might be useful to consider are:

1. Using the definition, prove or disprove that if T ( n ) ≥ 4 n + 2 lg n − 5 , then T ( n ) ∈ Ω ( n )
2. For each of the following algorithms, indicate (i) an appropriate size metric for input, (ii) the main operation(s) that will determine its efficiency class, (iii) whether the algorithm has best and worst cases or not.
   1. computing the sum of *n* numbers
   2. computing *n*!
   3. finding the largest element in an unsorted list of *n* numbers
   4. the sieve of Eratosthenes
   5. the standard pencil-and-paper algorithm for multiplying two *n*-digit decimal integers
3. For adding two *n* × *n* matrices, what are the basic operations, how many times are they performed as a function of *n*, and how many times are they performed as a function of the total number of elements in the input matrices?
4. Repeat the preceding question for multiplying two *n* × *n* matrices.
5. Sort the following list functions in order from smallest rate of growth to largest rate of growth.
   1. 10*n*
   2. *n*!
   3. 2*n*
   4. 2*n*3.2
   5. 3*n*2.3
   6. 2*n*
   7. 2*n*√*n*
   8. *n*1.5
   9. 2*n*3 lg *n*
   10. 3*n*2 lg *n*
   11. lg *n*
6. For each of the following pairs of functions, indicate whether the first is a lower bound of the second, an upper bound, a tight bound, or none of these.
   1. *n*(*n* + 1) and 2000*n*2
   2. lg *n* and ln *n*
   3. 2*n*− 1 and 2*n*
   4. 100*n*2 and 0.01*n*3
   5. (*n* − 1)! and *n*!
7. Indicate whether each of the following is true or false.
   1. *n*(*n* + 1)/2 ∈ O(*n*3)
   2. *n*(*n* + 1)/2 ∈ O(*n*2)
   3. *n*(*n* + 1)/2 ∈ Θ(*n*3)
   4. *n*(*n* + 1)/2 ∈ Ω(*n*)
8. Analyze the following algorithms as completely as possible. This means to state what the input size is, what operations are being counted, how the number of operations varies with the input size, and finally give a big-Theta or a big-Oh and a big-Omega description of the operations as a function of input size. Also, give a value for sum after the code has run to completion.
   1. sum = 0;
   2. for (i = 0; i < n; i++)
   3. {
   4. for (j = 0; j < n; j++)
   5. {
   6. sum++;
   7. }
   8. }
   10. sum = 0;
   11. for (i = 0; i < n; i++)
   12. {
   13. for (j = 0; j < i; j++)
   14. {
   15. sum++;
   16. }
   17. }
   19. sum = 0;
   20. for (i = 0; i < n; i++)
   21. {
   22. for (j = 1; j < i \* i; j++)
   23. {
   24. if (j % i == 0)
   25. {
   26. for (k = 0; k < j; k++)
   27. {
   28. sum++;
   29. }
   30. }
   31. }
   32. }
9. For the following algorithm, state what it computes, state what the input size is, state what operations should be counted, give an exact formulation for the number of operations as a function of input size, and give the efficiency class or classes for the algorithm.
10. double foo(const vector<double>& array)
11. {
12. double minval = array.at(0);
13. double maxval = array.at(0);
14. for (size\_t index = 1; index < array.size(); index++)
15. {
16. if (array.at(index) < minval)
17. {
18. minval = array.at(index);
19. }
20. if (array.at(index) > maxval)
21. {
22. maxval = array.at(index);
23. }
24. }
25. return maxval - minval;
26. }

1. Do the same analysis for this algorithm:
2. bool foo(const vector<double>& array)
3. {
4. size\_t n = array.size();
5. bool condition = false;
6. size\_t outer = 0;
7. while (!condition && outer < n - 1)
8. {
9. size\_t inner = outer + 1;
10. while (!condition && inner < n)
11. {
12. if (array.at(outer) == array.at(inner))
13. {
14. condition = true;
15. }
16. inner++;
17. }
18. outer++;
19. }
20. return condition;
21. }

Things that will *not* be on the test include

* shell commands
* gnuplot commands