Analysis of Algorithms –
CSE 4081/CSE 5211 – Final Exam

William David Shoaff email:wds@cs.fit.edu

Spring 2022 Posted Friday, April 29 and due Monday, May 2 at 3:00 P.M

1 Final Information

1.1 Resources

Most questions and answers can be found in my handouts and the (excellent) Cormen et. al. textbook. Of course, there are other ways to find answers. Some of which are acceptable. Other ways are not academically acceptable. Keep in mind the purpose of a final exam is to demonstrate your knowledge. Send me an email (wds@cs.fit.edu) should you have a question, a concern, or a doubt. I will try to give a timely reply.

2 Input and Output Size

1. The *size* of *input* often determines running time of a algorithm. The most commly used measure of size is the *Uniform Cost Model*: Each input value has size 1.

2. But, consider the question: Is natural number n *prime* or composite? There is only 1 input number n, and its *size* is more reasonably measured by the number of bits needed to n. This called the *Logarithmic Cost Model*.

3. The *size* of *output* often is seldom mentioned. It seems to me there should more discussion of it. But, often the problem/algorithm only requires a simple "yes or no" answer. On the other hand, I imagine you have waited hours for some large output to complete.

   I don't have a question here. Perhaps you do.

3 Loops, Counting , and Summations

There are two basic loop types: *fors* and *whiles*.

1. (10 points) The syntax for *for* loops (usually) describe how many times statements in its scope executes. An analysis skill is to be able to use summation notation to count the number of times instructions executes in a *for* loop.

   Here is an example from linear algebra: Given two vectors

   $$\vec{A} = \langle a_1, a_2, \ldots, a_n \rangle \text{ and} \vec{B} = \langle b_1, b_2, \ldots, b_n \rangle$$

   Their *inner (some call it a* dot*) product* is

   $$\langle a|b \rangle = \sum_{k=1}^{k=n} a_k \cdot b_k$$

   Write this sum as a *for* loop and state its big-O time complexity.

2. (5 points) *For* loops can be nested giving rise to a sequence of summations with one (perhaps) dependent on previous sums.

   Given two matrices $n \times n$ matrices A and B The standard algorithm to compute their product $C = A \times B$ is to compute the *inner product* of each row of A_i with each column of B_j.

   Write this description as nested *for* loops, express the operational complexities as summations and simplify the results.

   What is the big-O time complexity of this standard matrix multiplication algorithm?

4 Summations

3. (5 points) Geometric summations and their variations often occur because of the nature of recursion. What is a simple expression for the sum

   $$\sum_{i=0}^{i=n-1} 2^i?$$

4. (5 points) Differentiation and integration under a summation is formally useful.

   What are expressions for this derivative and integral of

   $$(\sum_{i=0}^{i=n-1} 2^i)$$

5 Recursion

In 1967, John Lennon and Paul McCartney wrote "All you need is love." In 1936, Alonzo Church wrote "All you need is recursion."

Given problem derive and solve a recurrence that can be solved to compute the time complexity of the algorithm. or vice-versa given a recurrence describe the problem(s) it solves.

5. (5 points) Binary Search What recursion describes *binary search* and what is its solution?

6. (10 points) A Common Recurrence is:

$$T(n) = 2T(n/2) + n$$

What is its solution and what are some algorithms it describes?

7. (10 points) Strassen's Matrix Multiplication The recursion for Strassen's (1966) matrix multiplication algorithm is

$$T(n) = 7T(n/2) + cn^2, \qquad \text{with initial condition } T(1) = 1$$

Assume that all matrices have sizes $2^{n-1} \times 2^{n-1}$ (so you can always partition them into quarters.

Solve the Strassen's recurrence to derive the running time of Strassen's matrix multiplication algorithm. There have been other developments since 1966. Summarize these.

I like unrolling as a solution technique. Unrolling the recurrence gives these first few substitutions:

$$\begin{aligned} T(n) &= 7T(n/2) + cn^2 \\ &= 7[7T(n/4)c(n/2)^2] + cn^2 \\ &= 7^2T(n/4) + 7c(n/2)^2 + cn^2 \\ &= 7^3T(n/8) + 7^2c(n/4)^2 + 7c(n/2)^2 + cn^2 \\ &\vdots \end{aligned}$$

The unrolling stops when $2^k = n$ or $n = \lg(k)$ and $T(n/2^k) = T(1) = 1$.

Finish this up to derive a simple expression for the time complexity of Strassen's matrix multiplication algorithm.

6 Algorithmic Paradigms

8. (10 points) List and describe at least 5 algorithmic paradigms that can be used to solve problems.

7 Applications

9. (5 points) An interesting question is: Which questions/problems have algorithms that can be applied to compute solutions?

   We know there are questions with "yes or no" answers for which there is no algorithm.

10. (5 points)
    1. Name and describe some of these undecidable questions.
    2. I like string problems and their algorithms (I know little about "String theory" but believe it is an interesting attempt to describe our universe (whatever that is) take time and look up string theory)
    3. Name some questions about strings and describe algorithms that answer these questions.

8 Computational Complexity

1. Adopting a *Model of Computation* is required to discuss some of these topics. I believe the *Turing machine* is the most cited model. Describe the components of a Turing machine and how the work together to solve problems.
2. Some questions require a "yes or no" answer. Some of these questions can be answered correctly by algorithm. Give examples of such decidable problems and describe the flow of their algorithms.
3. Other questions place constrains on parameters to filter (prioritize) potential solutions, and then select a best answer depending on some criterion.

8.1 Complexity classes

Language is important here. When I say "problem" I mean a thing that can be described by formal/mathematical notation of (strings, symbols, *etc.*) Each with well-defined meanings.

11. (5 points) Describe the class P of problems

12. (5 points) Describe the class NP of problems

13. (5 points) There is a nice characterization of NP problems: Please describe this characterization about given a solution.

14. (5 points) Reductions of instances: This is problem solving technique of reducing a new problem to an already solved problems. What time restrictions should be placed take on the run-time of a reduction?

15. (5 points) An $NP-$complete problem, let's call it Y,

16. (5 points) And then, there *recursively enumerable languages* & *recursive languages* languages. describe these language classes.