CSE 3101 Design and Analysis of Algorithms
Practice Test for Unit 0
Relevant Math

Jeff Edmonds

First learn the steps. Then try them on your own. If you get stuck only look at a little of the answer and then try to continue on your own.

Chapter 22: Existential and Universal Quantifiers

1. For each prove whether true or not when each variable is a real value. Be sure to play the correct game as to who is providing what value.

   1) $\forall x\ \exists y\ x+y=5$ 2) $\exists y\ \forall x\ x+y=5$
   3) $\forall x\ \exists y\ x \cdot y=5$ 4) $\forall x\ \exists y\ x \cdot y=0$
   5) $[\forall x\ \exists y\ P(x,y)] \Rightarrow [\exists y\ \forall x\ P(x,y)]$
   6) $[\forall x\ \exists y\ P(x,y)] \Leftarrow [\exists y\ \forall x\ P(x,y)]$
   8) $\exists a\ \forall x\ \exists y\ [x=0 \text{ or } x \cdot y=5]$

2. The game $Ping$ has two rounds. Player-A goes first. Let m_1^A denote his first move. Player-B goes next. Let m_1^B denote his move. Then player-A goes m_2^A and player-B goes m_2^B. The relation $AWins(m_1^A, m_1^B, m_2^A, m_2^B)$ is true iff player-A wins with these moves.

   (a) Use universal and existential quantifiers to express the fact that player-A has a strategy in which he wins no matter what player-B does. Use $m_1^A, m_1^B, m_2^A, m_2^B$ as variables.

   (b) What steps are required in the Prover/Adversary technique to prove this statement?

   (c) What is the negation of the above statement in standard form?

   (d) What steps are required in the Prover/Adversary technique to prove this negated statement?

3. Let $Works(P, A, I)$ to true if algorithm A halts and correctly solves problem P on input instance I. Let $P = Halting$ be the Halting problem which takes a Java program I as input and tells you whether or not it halts on the empty string. Let $P = Sorting$ be the sorting problem which takes a list of numbers I as input and sorts them. For each part, explain the meaning of what you are doing and why you don't do it another way.

   Extra:
   Let $A_{insertionsort}$ be the sorting algorithm which we learned in class.
   Let A_{yes} be the algorithm that on input I ignores the input and simply halts and says "yes".
   Let A_∞ be the algorithm that on input I ignores the input and simply runs for ever.

   (a) Recall that a problem is *computable* if and only if there is an algorithm that halts and returns the correct solution on every valid input. Express in first order logic that $Sorting$ is computable.

   (b) Express in first order logic that $Halting$ is not computable.

   (c) Express in first order logic that there are uncomputable problems.

   (d) What does the following mean and either prove or disprove it: $\forall I, \exists A, Works(Halting, A, I)$. (Not simply by saying the same in words "For all I, exists A, $Works(Halting, A, I)$")

   (e) What does the following mean and either prove or disprove it $\forall A, \exists P, \forall I, Works(P, A, I)$. Hint: An algorithm A on an input I can either halt and give the correct answer, halt and give the wrong answer, or run for ever.

4. Quantifiers:

(a) $\exists y \ \forall x \ x \cdot y = 0$
Prove whether true or not when each variable is a real value. Be sure to play the correct game as to who is providing what value.

(b) $\exists y \ \forall x \ x \cdot y = 5$

(c) A computational problem is a function P from inputs I to required output $P(I)$. An algorithm is a function A from inputs I to actual output $A(I)$. $Time(A, I)$ gives the running time for algorithm A on input I. Let $T(n)$ be some function. Use universal and existential quantifiers to express "Problem P is computable in time $T(n)$". (No BigOh)

(d) Express "Problem P is not computable in time $T(n)$".

Chapter 23: Time Complexity

5. This problem compares the running times of the following to algorithms for multiplying.

algorithm $KindergartenAdd(a, b)$

⟨*pre−cond*⟩: a and b are integers.

⟨*post−cond*⟩: Outputs $a \times b$.

```
begin
    c = 0
    loop i = 1 ... a
        c = c + b
    end loop
    return(c)
end algorithm
```

algorithm $GradeSchoolAdd(a, b)$

⟨*pre−cond*⟩: a and b are integers.

⟨*post−cond*⟩: Outputs $a \times b$.

begin
Let $a_{s-1} \dots a_3 a_2 a_1 a_0$ be the decimal digits of a, so that $a = \sum_{i=0}^{s-1} a_i \cdot 10^i$.
Let $b_{t-1} \dots b_3 b_2 b_1 b_0$ be the decimal digits of b, so that $b = \sum_{j=0}^{t-1} b_j \cdot 10^j$.
$c = 0$
loop $i = 0 \dots s-1$
loop $j = 0 \dots t-1$
$c = c + a_i \cdot b_j \cdot 10^{i+j}$.
end loop
end loop
return(c)
end algorithm

For each of these algorithms, do the following questions.

(a) Suppose that each addition to c requires time 10 seconds and every other operation (for example multiplying two single digits 9×8 and shifting by zero) is free. What is the running time of each of these algorithms either as a function of a and b or as a function of s and t? Give everything for this entire question exactly, i.e. not bigOh.

(b) Let $a = 9,168,391$ and $b = 502$. (Without handing it in trace the algorithm.) With 10 seconds per addition, how much time (seconds,minutes,...) does the computation require?

(c) The formal "size" of an input instance is the number of bits n to write it down. What is n as a function of our instance $\langle a, b \rangle$?

(d) Suppose your job is to choose the worst case instance $\langle a, b \rangle$ (i.e. that which maximizes the running time), but you are limited that you can only use n bits to represent your instance. Do you set a big and b small, a small and b big, or a and b to be the same size? Give the worst case a and b or s and t as a function of n.

(e) The "Running Time" of an algorithm is formally defined to be a function $T(n)$ from n to the time required for the computation on the worst case instance of size n. Give $T(n)$ for each of these algorithms. Is this polynomial time?

6. The input is an n bit integer x. The output is $\pi(x)$ which is the x^{th} digit of real number $\pi = 3.14\ldots$ written in decimal. What is the time complexity of the following algorithms?

   (a) It could take $Time\ 1/\epsilon$ to get π to accuracy within ϵ.

   (b) It could take $\Theta(1)$ time to get the next bit of accuracy of π.

   (c) I think I heard that they can now find the x^{th} bit without first finding all the earlier bits.

7. Time Complexity: Consider the two simple algorithms, summation and factoring.

   Summation: The task is to sum the N entries of an array, that is, $A(1) + A(2) + A(3) + \ldots + A(N)$.

   Factoring: The task is to find divisors of an integer N. For example, on input $N = 5917$ we output that $N = 97 \times 61$. (This problem is central to cryptography.) The algorithm checks whether N is divisible by 2, by 3, by 4, ... by N.

   Give the time complexity of each.

Chapter 24: Logarithms and Exponentials

8. Simplify the following exponentials: $a^3 \times a^5$, $3^a \times 5^a$, $3^a + 5^a$, $2^{6log_4 n+7}$, $n^{\frac{3}{log_2 n}}$.

9. Logarithms and Exponentials: Simplify the following exponentials: $a^3 \times a^5$, $3^a \times 5^a$, $3^a + 5^a$, $2^{6log_4 n+7}$, $n^{\frac{3}{log_2 n}}$.

Chapter 25: Asymptotic Growth

10. For each of the following functions sort its terms by growth rate. Get the big picture growth by Classifying it into $2^{\Theta(n)}$, $n^{\Theta(1)}$, $\log^{\Theta(1)}(n)$, $\Theta(1)$ or into a similar and appropriate class. Also give its Theta approximation.

    (a) $f(n) = 5n^3 - 17n^2 + 4$

    (b) $f(n) = 5n^3 \log n + 8n^3$

    (c) $f(n) = 2^{2^{5n}}$

    (d) $f(n) = 7^{3 \log_2 n}$

    (e) $f(n) = \{$ 1 if n is odd, 2 if n is even $\}$

    (f) $f(n) = 2 \cdot 2^n \cdot n^2 \log_2 n - 7n^8 + 7\frac{3^n}{n^2}$

    (g) $f(n) = 100n^{100} + 3^{4n} + \log^{1,000}(n) + 4^{3n}$

    (h) $f(n) = 6\frac{n^4}{\log^3 n} + 8n^{100}2^{-5n} + 17$

    (i) $f(n) = \frac{1}{n^2} + \frac{5 \log n}{n}$

    (j) $f(n) = 7\sqrt[5]{n} + 6\sqrt[3]{n}$

    (k) $f(n) = \frac{6n^{5.2}+7n^{7.5}}{2n^{3.1}+7n^{2.4}}$

    (l) $f(n) = -2n$

    (m) $f(n) = 5n^{\log^3 n}$

11. Suppose that $y = \Theta(\log x)$. Which of the following are true: $x = \Theta(2^y)$ and $x = 2^{\Theta(y)}$? Why?

Chapter 26: Adding-Made-Easy Approximations

12. Give the Θ approximation of the following sums. Indicate which rule you use and show your work.

(a) $\sum_{i=0}^{n} 7i^3 - 300i^2 + 16$

(b) $\sum_{i=0}^{n} i^8 + \frac{2^{3i}}{i^2}$

(c) $\sum_{i=0}^{n} \frac{1}{i^{1.1}}$

(d) $\sum_{i=0}^{n} \frac{1}{i^{0.9}}$

(e) $\sum_{i=0}^{n} 7\frac{i^{3.72}}{\log^2 i} - 300i^2 \log^9 i$

(f) $\sum_{i=1}^{n} \frac{\log^e i}{i}$

(g) $\sum_{i=1}^{\log n} n \cdot i^2$

(h) $\sum_{i=0}^{n} \sum_{j=0}^{m} \frac{j}{i}$

(i) $\sum_{i=1}^{n} \sum_{j=1}^{i} j^i$

(j) $\sum_{i=1}^{n} \sum_{j=1}^{i^2} ij \log(i)$

13. Zeno's classic "paradox" is that Achilles is traveling $1\frac{km}{hr}$ and has $1km$ to travel. First he must cover half his distance, then half of his remaining distance, then half of this remaining distance, ... He never arrives there. "The Story of Philosophy" by Bryan Magee states "People have found it terribly disconcerting. There must be a fault in the logic, they have said. But no one has yet been fully successful in demonstrating what it is." Resolve this ancient paradox by adding up the time required for each step.

14. Consider the following functions.

- **Classification:** Provide a Θ-approximation, if possible classifying the function into $\Theta(1)$, $2^{\Theta(n)}$, $\log^{\Theta(1)}(n)$, or $n^{\Theta(1)}$. What is this class called?
- **Summation:** Approximate the sum $\Theta(\sum_{i=1}^{n} f(i))$. Which result did you use and how did you use it? What type of sum is it?

(a) $f(n) = 5 \cdot n^{5.5} + n^{2.5} \log^{100}(n)$

(b) $f(n) = 5 \cdot 3^{4n} + n^{100}$

(c) $f(n) = \frac{5}{n^{1.01}} + \frac{7}{n^{0.99}}$

Chapter 27: Recurrence Relations

15. Give solutions for the following examples:

(a) $T(n) = 2T(\frac{n}{2}) + n$

(b) $T(n) = 2T(\frac{n}{2}) + 1$

(c) $T(n) = 4T(\frac{n}{2}) + \Theta(\frac{n^3}{\log^3 n})$

(d) $T(n) = 32T(\frac{n}{4}) + \Theta(\log n)$

(e) $T(n) = 27T(\frac{n}{3}) + \Theta(n^3 \log^4 n)$

(f) $T(n) = 8T(\frac{n}{4}) + \Theta((\frac{n}{\log n})^{1.5})$

(g) $T(n) = 4T(\frac{n}{2}) + \Theta(\frac{n^2}{\log n})$

16. Solve the famous Fibonacci recurrence relation $Fib(0) = 0$, $Fib(1) = 1$, and $Fib(n) = Fib(n-1) + Fib(n-2)$ by plugging in $Fib(n) = \alpha^n$ and solving for α.

17. Solve the following by unwinding them.

(a) $T(n) = T(n-1) + n$

(b) $T(n) = 2 \cdot T(n-1) + 1$

18. Recurrence Relations: Strassen's algorithm for matrix multiplication is given two $n \times n$ matrices A and B. In $\mathcal{O}(n^2)$ time, it finds seven pairs of $\frac{n}{2} \times \frac{n}{2}$ matrices. It recursively multiplies each pair. Then in time $\mathcal{O}(n^2)$, it combines the answers to the seven multiplication into the answer for $A \times B$. Let $T(n)$ be the total time for the algorithm given two $n \times n$ matrices. Give the **recurrence relation** for $T(n)$. Solve for it.