CISC-365*
Test #1
October 3, 2008

Student Number (Required) _____

Name (Optional) SOLUTIONS

This is a closed book test. You may not refer to any resources.

This is a 50 minute test.

Please write your answers in ink. Pencil answers will be marked, but may not be reconsidered after the test papers have been returned.

The test will be marked out of 50.

Question 1	/10
Question 2	/24
Question 3	/16
TOTAL	/50

Question 1 (10 marks)

Use induction to show that n^2 is in $O(2^n)$.

Solution

We need to show that there exist constants N and c such that for all n >= N, $n^2 <= c*2^n$

Base Case: Consider n = 4. $n^2 = 16 = 2^n$

Inductive Assumption: Assume that for some k >= 4, $k^2 <= 2^k$

Inductive Step: Let n = k+1

$n^2 = (k+1)^2 = k^2 + 2k + 1$

> Claim: $2k + 1 <= k^2$
> Proof: If $2k + 1 > k^2$ then $1 > k(k-2)$. But since k >= 4, the right hand side is >= 8.
> Contradiction.

Therefore $n^2 = (k+1)^2 = k^2 + 2k + 1 <= 2k^2 <= 2(2^k) = 2^{k+1} = 2^n$

Thus for all n >= 4, $n^2 <= 2^n$

Letting N = 4 and c = 1, we see that the requirement is satisfied, so n^2 is in $O(2^n)$

Note: We cannot say "assume n^2 is in $O(2^n)$, and then prove $(n+1)^2$ is in $O(2^{n+1})$" because the assumption and the conclusion are actually identical. The statement "n^2 is in $O(2^n)$" is a statement about a relationship between the two functions, not about a particular value of n.

Question 2 (24 marks)

Recall the Subset Sum problem: Given a set of n integers **S** and a target integer **k**, does S contain a subset that sums to k? We know that this problem is NP-Complete. In this question you will be asked to classify three variations of this problem. Each part of this question is independent of the others. **For each part, either show that the problem is NP-Complete or explain how you can solve it in polynomial time. (Note: if you can show the problem is NP-Complete AND you can solve it polynomial time, congratulations!) Each part is worth 8 marks.**

(a) Approximate Subset Sum: Given a set of n integers S, a target integer k, and an integer c, does S contain a subset with sum in the range [k-c .. k+c]?

Solution: This is NP-Complete. We can see this immediately because it is a generalization of the Subset Sum problem (in which the constant c = 0).

(b) Within 10 Subset Sum: Given a set of n integers S and a target integer k, does S contain a subset with sum in the range [k-10 .. k+10]? (Hint: what happens if you multiply all the values in the set S, and k, by 100?)

Solution: This is NP-Complete, using a reduction from Subset Sum. Given an instance of Subset Sum, multiply all values in S, and k, by 100. Since the last two digits of all the numbers are now 00, the only way to get a sum within 10 of 100*k is to hit it exactly. Thus the answer to "Within 10" on the transformed problem is Yes if and only if the answer to the instance of Subset Sum is Yes.

(c) 100 Value Subset Sum: Given a set of 100 integers S and a target integer k, does S contain a subset that sums to k? (You won't need a full page to answer this question – the pagination just worked out this way.)

Solution: This is polynomial – in fact it is solvable in constant time. Since the size of the set is fixed at 100 (a constant), we can list all possible subsets in constant time (specifically, it takes $O(2^{100})$ time, which is constant). Checking each subset to see if it sums to k also takes constant time so the whole operation takes constant time.

Question 3 (16 marks)

Let A[0..n-1] be an array of integers with the following properties:

- all the integers in A are distinct (no repetitions)
- the integers increase in value from A[0] up to some A[m], then decrease in value to A[n-1]
- either the increasing side or the decreasing side (or both) may be empty – see the examples

As examples, A might look like {3 6 19 18 13 2 -5} or {7 5 1} or {2 8 13}

Create an algorithm to find the value of m, the index of the largest value in A. Note that this is not difficult to do in O(n) time: we can simply look at each value in A and remember the location of the largest. Your solution must run in O(log n) time.

(a) Express your algorithm in clear pseudocode (or in some standard programming language, if you prefer)

Solution: This is a modified version of binary search. Instead of checking to see if we have found a particular value, we check to see which direction the numbers are increasing.

```
FP(a,b) {
  if (a > b) ***ERROR***
  else if (a == b) return a
  else if (a+1 == b)
            if (A[a] > A[b]) return a
            else return b
  else { // at least three elements still in consideration
    m = (a+b)/2                                  // we know a < m < b
    if (A[m-1] > A[m]) return FP(a, m-1)         // the peak is in the left side
   else if (A[m+1] > A[m]) return FP(m+1,b)      // the peak is in the right side
   else return m                                 // A[m] is the peak
  }
}
```

(b) State and explain a recurrence relation that describes the running time of your algorithm.

Solution: Let n represent the number of elements still in consideration.

$T(n) = c$ for $n <= 2$
$T(n) = T(n/2) + c$ for $n > 2$

where c is a constant

This is correct since for the non-trivial cases, we divide the set of elements in half (which takes constant time) and recurse on one of the two halves.

(c) Solve the recurrence relation to show that your algorithm runs in O(log n) time.

Solution:

$T(n) = T(n/2) + c$
$= T(n/4) + c + c$
$= T(n/8) + c + c + c$
...
$= T(n/2^i) + i*c$

Using the standard assumption that $n = 2^x$ for some x, which is to say $x = \log n$, we get

$T(n) = T(n/2^x) + x*c$
$= T(1) + c*\log n$
$= c + c*\log n$

Thus T(n) is in O(log n)

Special Bonus Question: (0 marks)

What is the meaning of the figure above?