EECS 477 - Sample Midterm Exam

Name -

UMich ID # -

DO NOT OPEN THE EXAM BOOKLET UNTIL YOU ARE INSTRUCTED TO BEGIN!

Honor Code: I have neither given nor received any help on this exam.

Signature: ____

You must work by yourself. This is a closed book exam. You are allowed to use one 4" x 6" index card with anything you like written on it. No other material may be consulted during the exam. You may not use any scratch paper other than what is provided in the exam booklet.

You do not need to solve all problem to receive a good grade. If you find a problem particularly challenging, try to solve easier problems first.

Please read all directions and questions carefully.

You have 1 hour and 20 minutes to complete this exam.

Good luck.

1. **Asymptotic Notation** (20 points)

For the following problems, if you use results proved in the textbook or in class, you need to restate them. You are not allowed to use any results about asymptotic complexity that were not given in the textbook or in class.

a) Let $f(n) = 6n^3(2^{n-1}) + (2\log n)(4^{\lg n})$ and $g(n) = n^2(3^{n+1}) + \frac{1}{2}(\log n)^2$

Is $f(n) = O(g(n))$? Justify your answer
Is $f(n) = \Theta(g(n))$? Justify your answer

b) Let $f(n) = 2^{2n} + \frac{1}{2}n^3(2^{n+1}) + 5n(\log n)$
and $g(n) = 3n(2^{n+3}) + 2n^2(3^n) + \frac{1}{2}n^2$

Is $f(n) = O(g(n))$? Justify your answer
Is $f(n) = \Theta(g(n))$? Justify your answer

2. **Subset Sum** (20 points)

The C++ code below prints out the sum of the values in each subset of a given set.

```
#include <iostream.h>

void subsetSum(int *set, int setSize, int sum)
{
    if(setSize == 0)
    {
        cout << sum << endl;
        return;
    }

    subsetSum(set+1, setSize-1, sum);
    subsetSum(set+1, setSize-1, sum+set[0]);
}
```

Give the recurrence for this subset sum algorithm and the asymptotic bound for this recurrence. You can solve the recurrence using any of the methods explained in class or in the textbook. If you use the Master Theorem, you need to state the theorem and clearly explain how it applies to this particular recurrence.

Note: Problem 2 on the actual midterm exam will be more involved than this, and you are advised to go over the analysis of divide-and-conquer algorithms in Chapter 7 when preparing for the exam.

3. **Fill in the blanks / Multiple choice** (20 points)

To answer a multiple choice question, circle one answer.

____________________ algorithm is an efficient technique for calculating greatest common divisors.

Instances where algorithms have the same running time are called ____________________.

Given that $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, we know that $f(n) =$ ____________________.

____________________ analysis involves taking the average over a series of calls to an operation.

The ____________________ can be applied to recurrences that have a form corresponding to $T(n) = aT(n/b) + f(n)$.

The ____________________ of a tree is the number of edges in the longest path from the root to a leaf.

Chaining is a technique for dealing with ____________________ in a hash table.

When a new value is inserted into a heap and the heap property is not satisfied, the new value must be ____________________ .

____________________ algorithm is used to find the shortest path from a given source vertex to each of the other vertices in the graph.

Is $T(n) = T(n/2) + n * T(n/3)$ a linear homogeneous recurrence?

Yes No

4. **Algorithm Analysis** (20 points)

Consider an algorithm that compares the string "GANDALFGANDALF..." of length $N = 7k$ (k repetitions of "GANDALF") to an arbitrary input string. The algorithm scans characters in both strings and stops when the corresponding characters in the strings are different OR they are both end-of-line characters.

a) Prove by induction that this algorithm terminates in finite time and correctly compares the two strings.

Be sure to note that this algorithm works with null-terminated strings. Emphasis should be placed on formulating a good claim that can be proven by induction and would imply the correctness of the algorithm.

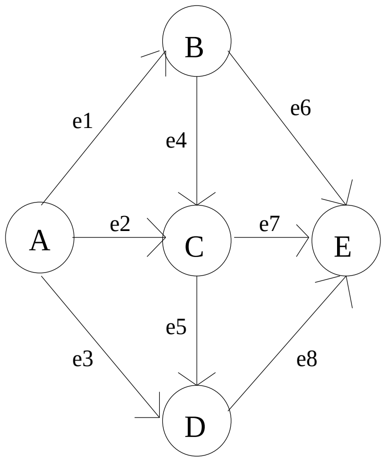
Hint: Use generalized induction with two parameters (the lengths of the strings), but note that the lengths are not computed. Try to come up with a simple argument (or even an explanation) why this algorithm always returns a correct answer, and then formalize it into a proof by induction.

b) What are the best-case, worst-case, and average-case (expected) complexities of this algorithm? For average-case analysis, assume that every character is equally likely in each position of the input string. Substantiate your answers (you will need a summation for average-case analysis).

c) Would it be a good idea (in terms of asymptotic complexities) to first compute and compare the lengths of the two strings?

5. **Shortest Path** (20 points)

Replace the edge weights in the graph below with the digits from your student ID. For example, if your student ID # is 8765-4321, then the corresponding edge weights would be e1=8, e2=7, e3=6, e4=5, e5=4, e6=3, e7=2, and e8=1.



a) Apply Dijkstra's algorithm to this graph and determine the length of the shortest path from source vertex A to each of the other vertices in the graph. You must show each step in the evolution of the algorithm.

Note: Students with really simple student IDs are encouraged to practice on graphs with arbitrarily-assigned weights.

b) Consider a directed, weighted graph $G = (V, E)$ in which all edge weights are nonnegative. If the number of edges that don't have a weight of 1 is bounded by some constant c, what is the running time of Dijkstra's algorithm? Is there a faster algorithm for such a case? If so, explain the algorithm and give its running time.

6. **BONUS: Gray Codes** (20 points)

NO PARTIAL CREDIT!

Prove or disprove whether the C++ code below will correctly print all subsets of a given set in a Gray Code ordering.

```
#include <iostream.h>
#include <math.h>

// subsets of set[] will be printed in a Gray Code ordering
void printGrayCodedSubsets(int set[], char *grayCodes[], int
length, int size)
{
    char *temp;

    for(int i=0; i<size; i++)
    {
        temp = grayCodes[i];

        cout << "{ ";
        for(int j=0; j<length; j++)
            if(temp[j] == '1')
                cout << set[j] << " ";
        cout << "}\n";
    }
}

// fills char* array grayCodes[] with Gray Codes
void calculateGrayCodes(char *grayCodes[], int length, int size)
{
    if(length == 0)
    {
        grayCodes[0] = "";
        return;
    }

    int i, j, k;
    int tempSize = size/2;

    char *tempCodes[tempSize];
    for(i=0; i<tempSize; i++)
        tempCodes[i] = new char[length-1];

    calculateGrayCodes(tempCodes, length-1, tempSize);

    for(i=0; i<tempSize; i++)
    {
        grayCodes[i][0] = '0';
        for(k=0; k<length-1; k++)
            grayCodes[i][k+1] = tempCodes[i][k];
    }

    for(i=tempSize, j=0; i<2*tempSize; i++, j++)
    {
        grayCodes[i][0] = '1';
        for(k=0; k<length-1; k++)
            grayCodes[i][k+1] = tempCodes[j][k];
    }
}
```

```
void main(void)
{
    const int n = 3;

    int someSet[n] = {3,2,1};

    int size = (int) pow(2,n);

    char *grayCodes[size];

    for(int i=0; i<size; i++)
        grayCodes[i] = new char[n];

    calculateGrayCodes(grayCodes, n, size);

    printGrayCodedSubsets(someSet, grayCodes, n, size);

    cout << endl << endl;
}
```