Problem Set #2
IE 495
Due September 18

Written Problems

1. Show that a recursive algorithm with running time satisfying

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

   satisfies $T(n) \in \Theta(n^2)$ if $f(n) \in \Theta(n)$.

2. The sequence of *Fibonacci numbers*, f_1, f_2, f_3, ... is defined recursively as follows:

   $f_1 = f_2 = 1$
   $f_{n+2} = f_n + f_{n+1}$

   - Develop a nonrecursive linear−time algorithm to return the n^{th} Fibonacci number.
   - Show that the running time of the recursive algorithm based on the definition is $\omega(n)$.

3. In class, we saw how to implement insertion sort using arrays. This implementation runs in $\Theta(n^2)$ time in both the average and worst case. Suppose we use linked lists instead of arrays.
   - What are the worst case and average case running times for the search step?
   - What are the worst case and average case running times for the insert step?
   - What is the overall worst case and average case running times?
   - For a given set of input data, which implementation will be faster?

4. Prove Taylor's formula with remainder by induction.

$$f(x)=f(0)+f'(0)\,x+\cdots+\frac{f^{(n)}(0)\,x^n}{n!}+\int_0^x \frac{t^n}{n!}\,f^{(n+1)}(x-t)\,dt$$

Programming Problems

5. The *partition problem* is the following. Given a set S and a weight function $w: S \to \mathbb{N}$, partition the elements of S into two subsets T and R such that $\sum_{a\in T} w(a)=\sum_{a\in R} w(a)$. No polynomial time algorithm is known for the general form of this problem. Show that if the weights are bounded by a constant u, then the problem can be solved in polynomial time. Give pseudo−code for such an algorithm. Hint: Use recursion.

6. Graph Data Structures

In class, we discussed the concept of a *graph*, given by a set V of vertices and a set $E \subseteq V \times V$. This is an important concept that is not only central to many areas in the field of Operations Research, but will also be central to many algorithms we will discuss in the course. There are several common data structures for representing graphs. Two of them are discussed in AHU, Section 2.3.

In many practical applications, the vertices of the graph represent geographic locations (assume they all lie in a flat plane) while the edges represent links between these locations (roads, railways, etc.). In these applications, there are usually costs associated with the edges that can be thought of as distances or travel costs between the pairs of vertices. In some models, these distances are taken to be Euclidean, i.e. the distance between vertices $v_1 = (x, y)$ and $v_2 = (x', y')$ is given by

$$\sqrt{(x-x')^2+(y-y')^2} \quad .$$

In these models, the edge costs can be specified implicitly by simply giving the coordinates of each vertex. The edges of the graph along with their costs can then be calculated.

Write a program which does the following:

1. Reads in a list of coordinates from a file. The first line of the file should contain the number of vertices and each remaining line should contain the x and y coordinates for the corresponding vertex. Coordinates can be assumed to be integers.

2. Calculates the corresponding edge costs and then determines each node's ten nearest neighbors. Edge costs can be rounded to the nearest integer.

3. Creates an adjacency list representation of the graph containing only the edges corresponding to each vertex's nearest neighbors.

4. Prints out a list of the edges in the graph and their costs in lexicographic order.

You may use the built−in sorting functions if you wish. However, for the long run, I suggest you write your own sorting routine. This will be an advantage later in the course.