22C:31 Homework 3

Due in class on Tuesday, March 23rd

This homework will be graded out of 60 points and it is worth 6% of your grade. The Teaching Assistant will grade some 6 out of the 8 problems, with each problem worth 10 points.

1. Here is pseudocode for Prim's algorithm for the *Minimum Spanning Tree* problem that is quite close to code in a high level programming language.

```
primMST(Graph G){

   //Defining Data structures and initializing them to be empty
   VertexSet S, MinHeap H, EdgeSet T

   // getVertex() returns an arbitrary vertex in the graph
   current = G.getVertex()

   // Initialize S to an arbitrary vertex
   S.insert(current)

   // Initialize H to contain all vertices, except current
   vertices = G.allVertices()
   for i = 1 to vertices.size() do
      if(vertices[i] != current)
         H.insert(vertices[i], null, LARGENUMBER)

   // Main loop that repeatedly and greedily selects cheapest
   // edges between S and the rest of the vertices
   while(S.size() != G.size())do{
      L = G.getNeighbors(current)

      // Process each neighbor of current to determine if its entry
      // in the heap can be decreased
      for i = 1 to L.size() do
         H.decreaseKey(L[i].neighbor, current, L[i].edgeWeight)

      // Pick a vertex in the heap that has the cheapest edge to S
      heapItem = H.delete()

      // Update S and T
      S.insert(heapItem.vertex())
      T.insert(heapItem.vertex(), heapItem.neighbor())
   }
}
```

   Most of the code is fairly self-explanatory. Here are some additional notes that will help you understand the code.

   - `G.getNeighbors(v)`: returns an array with all the neighbors of `v`. Each item in the array contains two fields: (i) `neighbor` u, and (ii) `edgeWeight` w. Here u is a neighbor of v and w is the weight of edge $\{v, u\}$. The size of the array is equal to the number of neighbors of v.

- `MinHeap H`: Each item in `H` contains 3 fields: (i) `vertex`: v, (ii) `neighbor`: u, and (iii) `edgeWeight`: w. The heap is organized by `edgeWeight`, i.e., an item with smallest edge weight is at the top of the heap, etc. The vertex v is outside S, and the edge $\{v, u\}$ is a lightest edge from v to a vertex in S, and w is the weight of this edge.
- `H.decreaseKey(v, u, w)`: This `minHeap` operation compares edge weight w with the `edgeWeight` currently associated with v in the heap. If w is smaller, then the `edgeWeight` field (associated with vertex v) is decreased to w and the `neighbor` field (associated with vertex v) is updated to u.

It is easy to implement a graph data structure (for example using adjacency lists) that maintains a graph with n vertices and m edges in which `getVertex()` takes $O(1)$ time, `allVertices()` takes $O(n)$ time, and `getNeighbors(v)` takes $O(degree(v))$ times. Based on this information and what you know about the efficiency of heap operations, carefully analyze the running time of the above pseudocode. Specifically, label the lines of the code $1, 2, 3, \ldots$ and for each line i specify (i) the running time of one execution of line i, (ii) the total number of times line i might be executed, and (iii) the total running time of line i. Your final answer should specify the running time of the entire algorithm, as a function of m and n.
Note: I did an "informal" version of this analysis in class.

2. Here are problems on running time analysis, similar to the one on Exam 1.

   (a) Consider the following code fragment.

   ```
   while(n ≥ 1)do
     m ← n
     while (m ≥ 1) do
       for i = 1 to m do
         print("hello world")
       m ← m/2
     n ← n/2
   ```

   Express the running time of this code fragment as a function of n. Specifically, come up with a function $f(n)$ such that the running time of the function is $\Theta(f(n))$. Prove your claim by showing that the running time of the code fragment is both $O(f(n))$ and $\Omega(f(n))$.

   (b) Now modify the code fragment so that the for-loop goes from 1 through n, instead of from 1 through n. Redo (a) for this modified code.

3. Solve the following recurrences and obtain an asymptotic upper bound on $T(n)$:

   (a) $T(n) = 4T\left(\frac{n}{2}\right) + n$.
   (b) $T(n) = T(n-1) + n$.
   (c) $T(n) = 3T\left(\frac{n}{4}\right) + n^2$.
   (d) $T(n) = T(\sqrt{n}) + 1$.

   In all cases, assume that $T(n) = b$ for all $n \leq a$ for some convenient constants a and b.

4. Problem 26, Page 202.

5. Problem 28, Page 203.

6. Problem 31, Page 204.

7. Problem 4, Page 247.

8. Problem 7, Page 248-249.