CSE 307 – Assignment # 3

Assignment problem # 1 (marks: UG: 5, PG: 4)

Write an SML function **collectNodes(L)** which given a list of directed edges (e.g., **[[1,2],[3,4],[2,3],[5,6]]**) returns the list of nodes in those edges without duplicates (e.g., **[1,2,3,4,5,6]**). Example: **collectNodes([[1,2],[3,4],[2,3],[5,6]])** returns **[1,2,3,4,5,6]** (in any order).

Hint: you can implement **flatten** to obtain a list of nodes with duplicates and then remove the duplicates.

Assignment problem # 2 (marks: UG: 5, PG: 4)

Write a purely functional SML function **product(L)** that returns the product of the elements in the list L greater than 0. Invoked for an empty list, it should return 1.

Example call: **product[~1,~2,0,1,2,3];** returns **val it = 6 : int**.

Assignment problem # 3 (marks: UG: 5, PG: 4)

A widely-used data structure is the priority queue. A priority queue is an ordinary queue extended with an integer priority – this can be represented in SML as a **tuple (value,priority)**. When data values are added to a queue, the priority controls where the value is added. A value added with priority p is placed behind all entries with a priority ≤p and in front of all entries with a priority >p. Note that if all entries in a priority queue are given the same priority, then a priority queue acts like an ordinary queue in that new entries are placed behind current entries.

Write an SML function **insertSorted(ListOfTuples:(int * int) list, Value:int, Priority:int)** that inserts the **tuple (Value,Priority)** in its right place and returns the new priority queue.

Example: **insertSorted([(1,1),(2,2),(3,3)], 4, 2);** returns

[(1,1),(2,2),(4,2),(3,3)];

Assignment problem # 4 (marks: UG: 5, PG: 4)

Write a purely functional SML function **setEqual(L1,L2)** that returns true if two sets L1 and L2 represented as lists are equal.

e.g. **setEqual([1,3,2], [2,1,3]);** returns

val it = true : bool

Assignment problem # 5 (marks: UG: 5, PG: 4)

Assume we have a list L of integers. Define a function unitLists(L) that places each integer in L into its own sublist (of size one). That is, if L has n integers in it, unitLists produces a list of n sublists, each containing a single integer from L. For example,

unitLists([]) ⇒ []

unitLists([1,2,3,4]) ⇒ [[1],[2],[3],[4]]

Assignment problem # 6 (Mandatory for graduate students) (Marks: PG 5)

Each of the sublists produced by unitLists in problem 5 is trivially sorted. If we merge the first two sublists together into sorted order, we have a sorted sublist of size 2. If we then merge the third and fourth sublists, then the fifth and sixth sublists, etc., we end up with n/2 sorted sublists of size 2, rather than n sublists of size 1 (the final sublist may be of size 1 if it has no partner to pair with). If we iterate this merging process, we next get n/4 sorted sublists of size 4, then n/8 sorted sublists of size 8, etc. Finally, we produce a single sorted sublist of size n. This sorting logic is the basis of a merge sort.

Write an SML function **mergeSort(L)** that first divides L into unit sublists using unitLists, and then repeatedly merges adjacent sublists until a single sorted list is produced. You may create and use any additional functions you find useful or necessary.

Example: **mergeSort([6,3,7,1,2]);** returns **val it = [1,2,3,6,7] : int list.**