COMP 3761 Assignment 6

Due: Wednesday **Mar 11, 2015** at 6:30pm

1. Refer to textbook Page 186 Exercises 5.3 Problem 8
   (a) Exercises 5.3 Problem 8 part (a). [1]
   (b) Exercises 5.3 Problem 8 part (b). [1]
   (c) Exercises 5.3 Problem 8 part (c). Provide the pseudocode of your algorithm. [2]
   (d) Implement your algorithm that constructs a binary tree, given two lists that are generated by the inorder and postorder traversals of the tree. [5]

   Your program must read in the input values from the standard input (not command-line arguments) formatted as follows:

   i. first line input contains one positive integer number n, where n stands for the number of nodes in the binary tree. The tree nodes are labelled as $0, 1, \ldots, n-1$.
   ii. second line input contains n distinct integer numbers (in the range of $[0..\, n-1]$, separated by white spaces), where the list of numbers represent the order of the tree nodes visited by the inorder traversal of the binary tree.
   iii. third line input contains n distinct integer numbers (in the range of $[0..\, n-1]$, separated by white spaces), where the list of numbers represent the order of the tree nodes visited by the postorder traversal of the same constructed binary tree.

   Sample Input:
   ```
   10
   9 3 1 0 4 2 7 6 8 5
   9 1 4 0 3 6 7 5 8 2
   ```
   (e) Implement the preorder traversal algorithm for a binary tree. After the binary tree of n nodes is constructed by your program in (d), call the method of binary tree preorder traversal, and output the corresponding ordered list generated by the preorder traversal of the same binary tree. [1]

2. Implement the algorithm of HeapSort. Test your implementation with a randomly generated input array of size $n = 10000$. Your test program must verify that the input array is indeed sorted in the non-decreasing order after invoking the method of HeapSort.

   Note: Heapsort is an in-place sorting algorithm. That means, the heapsort (and all of its subroutines) must be performed in-place (using the original input data array). You are not allowed to create any new arrays to copy the input array elements in any of the helper methods you created in this question. [5]

3. The goal of this problem is to design and implement a "Median Maintenance" algorithm by using the heap data structures (min-heap and max-heap) .

   Download the input test data file *median.txt*

   The text file contains a list of the integers from 1 to 10000 in unsorted order. You should treat the data as a stream of numbers, arriving one by one.

   Let x_i denote the i-th number of the file, the k-th median m_k is defined as the median of the numbers $x_1, \ldots, x_k$. If k is odd, then m_k is $(k+1)/2$ th smallest number among $x_1, \ldots, x_k$; if k is even, then m_k is the $k/2$ th smallest number among $x_1, \ldots, x_k$.

   (a) Given an array $H[1..n]$ of n integers, implement a linear iterative method for checking whether a subarray $H[l..r]$ rooted at index l is a heap (max-heap or min-heap).

   In the sample signature below, the integers of start and end indicate the subarray indices l and r in $H[l..r]$, the flag isMax indicates whether the array H to be tested is for max-heap or min-heap (true for max-heap, false for min-heap). You may have your own method signatures, if you like. [2]

```
boolean isHeap(int[] H, int start, int end, Boolean isMax)
// returns true if the subarray H[start..end] is a heap
```

   (b) Given a subarray $H[l..r]$ (with $k = r - l + 1$ elements) represented as a heap (max-heap or min-heap, which is already verified as in part (a)), design and implement an algorithm that adds a new element (integer) into the heap. Your algorithm must maintain the increased subarray portion $H[l..r+1]$ as a heap in the same kind (max-heap or min-heap) and have the worst-case running time efficiency of $O(\log k)$. Return true if the extended subarray $H[l..r+1]$ is a heap, false otherwise.

   Note: You may assume that H has already allocated enough space for the additional elements. You may want to add boolean flag (isMax) to the sample method signature below, if you want this method to handle both max-heap and min-heap [2]

```
boolean addNode(int[] H, int start, int currentEnd,  int newItem)
```

   (c) Suppose a list of k integers in an array $H[1..n]$ of n integers (where $k \leq n$) stored in two heap structures, where half of k items in a min-heap and half of the k items in a max-heap. Design an algorithm to store the k integers in such a way that we can easily extract the median value of the k integers m_k. Your algorithm should correctly handle the cases of both k is odd and k is even. [2]

   (d) Design and implement an efficient algorithm that solves the "median maintenance" problem by iteratively adding a new integer into the existing heap data structures as in part (c) and updating the median values found so far in the extended heap data structures. Determine the run-time efficiency class of your solution. [2]

   (e) Write a test driver to test your solution in (d) using the input data from the file *median.txt* . Your program should compute the sum of the 10000 running medians from the streamed numbers and display only the last 4 digits of the sum. That is, you should compute $(m_1 + m_2 + m_3 + \ldots + m_{10000}) \bmod 10000$. [2]

IMPORTANT NOTES:

- You should use a single program (with one main method) to test all the problems in this assignment.
- Design your heap data structure carefully so that you may share some code in Problem 2 and Problem 3. Avoid duplicate code.
- As in the previous assignments, you can hard-code the given input file names, but do NOT use any absolute file path in the code. You can simply copy the files to the same directory where your java code is located.
- Print out the required program output only and submit the test results.