**Lab 3: Sorting algorithms**

*All code should be properly documented, commented and have proper unit testing implemented in main(). To facilitate things you should incorporate the README file for each task as a comment in the beginning of the corresponding file.*

*You are not allowed to use Java library implementations of the data structures (queues, lists, STs, hashtables etc.) unless it is otherwise stated so in the assignment. The idea is that you should learn to implement the internals yourself!*

*NB: This lab also covers material introduced in ch. 3.4 and 3.5 in hashing and applications*

*Input for the programming assignments: You should use this text as input for your programming assignments: http://www.gutenberg.org/files/98/98-0.txt (Länkar till en externa sida.)Länkar till en externa sida. henceforth referred to as "the text".*

**Programming assignments:**

1. Write a simple filter to clean a text, i.e. to remove all characters that are not alphabetic, blank or newline - replacing every such character by a blank to keep the number of characters constant to the original text. Hint: this is easy to do in C using the "isalpha()" function (to find out more about it on a unix/linux system type: man isalpha as a command to the shell) This could be useful for the next assignments. A filter is a program which reads its input from stdin, processes it and the prints the result to stdout (i.e. reads and writes from/to the terminal).
2. Use the first thousand words from the text to measure the running time of the ordered array ST (also known as binary search symbol table, see algorithm 3.2 in the book (obs not chapter 3.2)). Use the FrequencyCounter from page 372 as test program (you may need to change how you read the words if you do not use the libraries from Sedgewick&Wayne). Show tables or graphs of your measurements.
3. Use the first thousand words from the text to measure the running time of the Binary Search Tree algorithm (Algorithm 3.3 in the book (obs not chapter 3.3)) (you need only implement the basic operations to put and get keys to/from the ST) Use the FrequencyCounter from page 372 as test program (you may need to change how you read the words if you do not use the libraries from Sedgewick&Wayne). Show tables or graphs of your measurements.
4. Compare and explain the differences in execution time of the two algorithms in assignments 2 and 3.
5. Write a program that shows how evenly the built-in hashcode() function for strings in Java distributes the hashcodes for the words found in the text. (Hint it may be hard to use the hashcodes directly without manipulating them...)
6. Write a program which allows the user to repeatedly ask questions (without re-reading the text or re-building the data structures used) how many times a word occurs in the text. Base the implementation on a hash table which uses separate chaining.
7. Write a program based on a Binary Search Tree which reads the first two hundred words from the text and allows the user to select to print these words in either alphabetic or reverse alphabetic order. The printing of the content of the tree should be done in a method while traversing the tree. The time complexity of the printing should be O(N) where N is the number of different words in the text and the memory complexity should be O(log(N)) assuming the tree is balanced.