Applicable to:

* Students who have collected at least 60% (17 points) but less than 80% (22 points) of the possible points from the lab course ht21

Grading, due date and presentation schedule:

The solutions to the assignments should be uploaded in Canvas and presented in individual Zoom meetings. You should solve all problems to pass the lab course.

Students who have uploaded the solutions in Canvas will be able to book presentation times through a link which will appear here on nov 11th.

HINT: If you encounter problems downloading the large input files for the problems in your browser (i.e. leipzig1m.txt & NYC.txt) try to right-click on the link and select "Save link as ..." and save it directly to a file (which is simpler than copying from the browser ;-)

Requirements:

* All assignments should be solved
* All code should be well-commented
* All assignments should be uploaded into Canvas
* Oral presentation of the assignments where the student should be able to show the code executing - reserve a time slot here
* You are allowed to use standard ADTs from the Princeton website for assignments 2-5 limited to insertion-, quick- and mergesort; Symbol Tables; and breadth-first search, depth-first search algorithms for graphs

The student should have solved the assignments independently, be able to explain all assignments in detail and make additions/changes to the code at the presentation.

If there is a choice, you should select the most efficient algorithms/data structures covered in the course for each assignment.

Assignments:

***For all code assignments there should be exhaustive test code which takes it input from stdin. Your code should be reasonably efficient - it should not have big-Oh larger than what would be expected from good implementation***

1. Implement a double linked list with a sentinel element. The API should have methods to:  
   i) A method to create a new list instantiated with some type of elements (data stored in the list) defined at the time the list is created  
   ii) A method to insert an element at the beginning of the list   
   iii) A method to insert an element at the end of the list   
   iv) A method to remove and return the first element in the list  
   v) A method to remove and return the last element of the list  
   vi) A method to insert elements ordered in ascending order in the list  
   vii) You should calculate the Big-Oh complexities for insertion and removal of elements  
   Limitations: You are not allowed to use extra references such as first/last to keep track of where the list starts/ends. The total amount of lines of executable code (statements such as if, for, while, new, method calls and assignments) for the methods i - vi, should be 20-40 lines including lines of executable code in methods called. This requires careful design of the methods and "helper" methods.
2. i) Compare/evaluate the execution time of two versions of Quicksort with cutoff to Insertionsort. The versions should use shuffling of the input vs. median-of-three without shuffling. The input should be integer values. Use input that is sorted and random. Vary the input sizes from 100 to 1000000 elements with integer values uniformly distributed in the interval [0, 100000] and [0,100] respectively. Make sure your results are statistically significant. The program should take the number of integers N in the input as parameter by the command line arguments (argv).   
   ii) You should be able to show (think graphs) and explain your results.   
   For this experiment you need to use a random number generator and run experiments.   
     
   **On (pseudo) random number generators:** A random number generator is an algorithm that given a starting value called *seed* will generate a sequence of values which seems to exhibit good statistical properties, i.e. appear as "white noise". The sequence of numbers generated is completely deterministic and only depends on the *seed* that the random number generation has been initialized with. That is, given that we start with the same *seed* and run two sequences, the sequences of numbers generated with a specific random number generator will be identical. On a LINUX/UNIX machine you can read more about the standard library function for generating random numbers and initiating the sequence with a *seed* by typing the command: ***man random***
3. Implement a program which can sort input consisting of: <string, integer>  
   The input should be of the format:  
   6  
   B 47  
   K 97  
   A 12  
   B 567  
   A 32  
   K 4  
     
   The result of the sorting should be sorted by the string and then by the integer associated with the string:  
     
   A 12  
   A 32  
   B 47  
   B 567  
   K 4  
   K 97  
     
   The program should read the input from *stdin*with the number of tuples to sort (N) first followed by the tuples to sort. You should also write a program to generate the input. That program should take the number of tuples and a random number seed as inputs from the command line arguments (*argv*).
4. Implement a program which takes as input a text file and allows the user to repeatedly ask questions:   
   i) Which is the k:th most common word  
   ii) Which are the k:th to the k+n:th most common words  
   Use <https://introcs.cs.princeton.edu/java/data/leipzig/leipzig1m.txt> as input. The time to build the index must not exceed four minutes.
5. Implement a program which reads an ASCII text file whose name is given as an argument at the command line. The text file should contain a text written in English which may contain misspellings of words. The program should output a corrected version of the text, basing the corrections on the list of common misspellings found here: <https://en.wikipedia.org/wiki/Wikipedia:Lists_of_common_misspellings/For_machines>  
   The execution time complexity should be near linear compared to the time it takes to read the input files. That is, you need to show that your algorithm meets this constraint.  
   For this assignment you may use the abstract data types defined in the course directly (i.e. you need not implement them yourself)
6. Implement a program which allows the user to find the shortest path between two nodes in a directed graph with edge weights, possibly passing through a third node. I.e. the user should be able to ask questions like:  
   *Which is the shortest path from A to B passing through C?*   
   The program should output an ordered list of the nodes to traverse from A to B if such a path exists. If no such path exists then the program should output that no such path exists.  
   Use [NYC.txt](https://algs4.cs.princeton.edu/44sp/NYC.txt) as input when not executing tests (in the case that the tests should be executed you may use another input). This is the undirected road network of New York City. The graph contains 264346 vertices and 733846 edges. It is connected, contains parallel edges, but no self-loops. The edge weights are travel times and are strictly positive. You should also calculate/show the time complexity of your algorithm.  
   This assignment requires that you study material on edge weighted graphs in the book. (the algorithm is used in routing in the Internet, i.e. it is useful to know)