CSE 5311: Advanced Algorithms
PROGRAMMING PROJECT TOPICS

Several programming projects are briefly described below. You are expected to select one of these topics for your project. However, in exceptional circumstances I am also willing to consider other programming proposals that you may have on your own, provided they have a very strong relation to the contents of this course. Once you have selected a topic, please make appointments with me and/or Arjun asap to discuss the requirements of your topic in more detail. Discuss the basic architecture of your system, such as main data structures, main components of the algorithm, design of the user-interface for input/output, etc. with the TA so as to get started on the right track.

Team Size: Max 2 members.

Final Project Demonstration:
Once the project is completed, the following is expected of you:
1. A demonstration of your project to me and/or Arjun, in which you show the various features of your system, such as its correctness, efficiency, etc. You should be prepared to answer detailed questions on the system design and implementation during this demo. We will also examine your code to check for code quality, code documentation, etc.
2. You should also hand in a completed project report, which is essentially a polished version of the project design document, but should also include some experimental results, e.g. charts of running time versus input size, etc.
3. You should also turn in your code and associated documentation (e.g. README files) so that everything can be backed up for future reference.
4. Email your code and all associated files to me (cc TA) with "CSE5311-Project <Lastname>" in the subject.

Project Topics:
1. **Shortest Paths and Minimum Spanning Trees:** In this project you will implement the Dijkstra's shortest path algorithm as well as Prim's minimum spanning tree algorithm (read up about Prim's algorithm). Assume adjacency list representations of the input graphs. Implement the fringe data structure using a binary heap and alternatively, as a linked list, and compare the two methods. Some animation which shows each major iteration of the algorithms will be very desirable. Have a user interface in which you can incrementally add edges to the graph and update the trees accordingly.
2. **Median finding, Order Statistics and Quick Sort:** In this project you will implement the medianfinding algorithms. The user should be able to select the "k", i.e., the rank of the number desired as output (k = n/2 is the median). You should also be able to select groups of 3, 5, 7 etc in the linear-time median finding algorithm and be able to compare the performance of each. Also read up and implement the randomized median finding algorithm and compare against

the linear-time one. Implement quick sort using both algorithms and compare the performances of these different versions.
3. **Convex Hull:** Implement several versions of the convex hull algorithm: (a) the divide and conquer version, (b) the Graham Scan version, (c) the version in which a preprocessing step removes all points in the extremal quadrilateral, and (d) a convex hull algorithm when the input is not a set of points, but is a non-convex polygon. Have a GUI where one can insert new points (for (d), the user can add a new vertex to the input polygon) and the application recalculates the hull. Compare the performances of all algorithms.
4. **Network Flow:** Implement the Ford-Fulkerson algorithm for computing network flow in bipartite graphs. While your algorithm should be able to handle large graphs, for demo purposes design a GUI in which any bipartite graph of up to 20 nodes can be specified. Some animation which shows each major iteration of the algorithm will get extra credit. Run experiments that measure the performance of the
algorithm over large inputs.
5. **String Matching algorithms:** Implement both the KMP pattern matching algorithm as well as the Longest Common Subsequence (or Edit Distance) algorithm. Your algorithm should be tested out on real datasets, e.g., text files, biological sequence data, times series databases, etc. Compare the performances of each algorithm against naive algorithms for the same problems.
6. **Matrix Multiplication:** Read up about how large matrices are multiplied in the text book. Implement the Soloway-Strassen matrix multiplication algorithm, as well as the naïve algorithm. Compare the running time of the two algorithms experimentally using synthetically generated large matrices
7. **Six Degrees of Kevin Bacon:** This task comprises of downloading data from IMDB (actors, movies, and co-actors) and making an interface for finding the shortest path between Kevin Bacon and another actor. Two actors are linked together if they have starred in a movie together. Thus, given an actors name the interface will return a sequence of other actors (related by common movies) that connect up to Kevin Bacon. The IMDB dataset are available as text files at http://imdb.com/interfaces#plain . You will need to download relevant files which will provide information needed to connect actors with movies. Consider only English Language movies for this case. For more information on this game see http://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon.
8. **Sudoku Solver:** Implement a brute force sudoku solver along with an optimized solver (Eg. The Backtracking Solver) for a nxn matrix. Compare performance over varied datasets.

Due Date: TBA