# 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 we are 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 TA and/or Dr.Das 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 in the class 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 contains details about your project, such as main data structures, main components of the algorithm, design of the user-interface for input/output, 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 Dr.Das (cc TA) with "CSE5311-Project <Lastname>" in the subject.

**Project Topics:**

**1. Median finding, Order Statistics and Quick Sort:**

*Median Finding :*

Implement the following order statistics algorithms :

a. Median of Median with groups of 3,5 and 7.
b. Randomized median finding algorithm.

Make your code generic enough so that it can answer order statistics for any k. For eg k = n/2 gives the median and k=n gives the max.

*Quick sort :*

Implement quick sort where the pivot is chosen from the previous order statistics algorithms. Compare the performance of quicksort with median as pivot with the following practical implementations :

a. Randomized quicksort which chooses a random element in the array as the pivot.

b. (Simplified) real world quick sort with the following heuristics :

1. if L and R partitions are of unequal sizes, sort the smallest partition first.
2. if array size <= 7, use insertion sort
3. Use the following idea for getting the pivot.

```
middle = (start+end)/2
if array size > 40
        length = array size / 8
        pivot1 = median(start, start - length, start - (2*length))
        pivot2 = median(end, end+length, end + 2*length)
        pivot3 = median(middle,middle-length, middle+length)
        pivot = median(pivot1,pivot2,pivot3)
else
        pivot = median(start,middle,end)
```

Compare the performance of different implementations of quicksort. Test it with large arrays (eg 10000, 100000 or higher) with random elements from different distributions like uniform, normal etc.

## 2. BST and Heap : Huffman coding and decoding

Huffman Encoding is one of the simplest algorithms to compress data. Even though it is very old and simple , it is still widely used (eg : in few stages of JPEG, MPEG etc). In this project you will implement huffman encoding and decoding. You can read up in Wikipedia or any other tutorial.

Your system must accept a file and you need to form a binary (huffman) tree for the same. During the construction of huffman tree, use the priority queue to select nodes with smallest frequencies. Once you have constructed the tree, traverse the tree and create a dictionary of codewords (letter to code). Given any new sentences, your system must show how the sentence is converted to huffman code and then decoded back to original sentence.

Note that you must implement BST and Heap yourself and must not rely on any language libraries. You can use external libraries like GraphViz to display your huffman tree.

## 3. Red-Black Trees : Shared Memory de-duplication

Red black trees are one of the most important data structures used extensively in the Linux Kernel. For eg, recent kernels use Completely Fair Scheduler for process scheduling which depends on Red Black trees !

In this project, you will use Red black trees to simulate another of their popular applications – Shared memory de-dupulication. You can read up the algorithm used at http://www.ibm.com/developerworks/linux/library/l-kernel-shared-memory/index.html .

Your system must have the following operations :

a) **Load** : This will accept a list of <page id, hash of page content> records.

b) **Update :** This will accept another list of <page id , hash of page content>. If page id already loaded, then just update its hash. This is equivalent to page content changing. If page id is not yet loaded, then add it to the system.

c) **De-dupulicate :** In this operation, you must run the de-duplication algorithm explained in

the website and display if any memory is freed.

**4. Minimum Spanning Tree : Solving TSP for Metric Graphs using MST Heuristic**

Given an arbitrary metric graph, construct its Minimum spanning tree using Kruskal's algorithm. You can assume adjacency matrix representation of graphs. If you wish, you can reuse external libraries for heaps.

Now use the constructed MST to find an approximate estimate for the TSP problem. You can choose to implement any of the two approximation algorithms specified in Wikipedia's entry on TSP – One with approximation factor of 1.5 (Christofides) or 2. Compare it with the optimal answer . You can use some external library to find the optimal solution to the TSP problem.

**5. Network Flow : Task allocation using Bipartite Graph**

In this project, you are given two sets : set of employees and tasks. For each task , you are also given the list of employees who can complete the task. Model this scenario as a bipartite graph and allocate work in such a way that the job is completed as soon as possible. You must also utilize your workforce as much as possible. Solve this problem using Network flow . Implement both Ford-Fulkerson and Edmond-Karp. Your UI must be able visualize augmented path during each iteration .

**6. String Matching : Simple Plagiarism detection using String matching algorithms**

Plagiarism is a serious problem in research. In this project, you will implement a very simple plagiarism detector. Your input will be a corpus of existing documents and a potentially plagiarized document. Your output will be the set of documents from which the document was plagiarised from. Implement the following basic detectors and compare their performance :

a) **KMP :** Given a test file, treat each sentence in the file as a potential pattern. Search for the pattern in the existing documents and find the matches.

b) **LCSS** : Run the LCSS algorithm for each paragraph from the test file and existing corpora.

c) **Rabin-Karp fingerprints** : Implement Rabin-Karp algorithm for multipattern matching. You can use the language libraries for converting string to hash.

The definition of plagiarism and threshold are left to your discretion.

**7. Convex Hull : Triangulation using Convex Hull Algorithms**

a. Implement the following convex hull algorithms : Divide and Conquer, Graham Scan and Jarvis March.

b. Onion Convex Hulls : For a given set of points, you can create a set of concentric convex hulls. Let S be the set of original points. Now find the convex hull using the algorithms you implemented in the previous task. Let S' be the set of points from S that are not in the convex hull. Find the convex hull for S' . Repeat the process till no points remain within the convex hull

c. Implement Onion triangulations for any two consecutive convex hull using Rotating calipers method. For details refer http://cgm.cs.mcgill.ca/~orm/ontri.html .