CS 375 Design and Analysis of Algorithms **Fall 2017**

PROJECT

I. INTRODUCTION

For this project, you will work in a project group with ***exactly two members***, all of whom must be enrolled in your section of the class. It is possible that some groups could have 3 members. Please assume this is not your group unless we contact you (Don't request a group of size 3; we'll decide that). You will select one or more algorithms to implement and test thoroughly, using your language of choice (C, C++, or Java). Importantly, you will design one or more questions that you hope to answer about your algorithm, and you will use your implementation and a well-designed set of experiments to answer your question(s). You will explain the problem your algorithm addresses, how your algorithms and data structures work, and what you learned about them from your experimentation. You will provide this explanation in an Oral Presentation (12-15 minutes) of your work to your classmates, at the end of the semester. Your Oral Presentation will be supported by well-designed and informative slides, whose design and content will serve as a component of your grade. Both group members must contribute equitably to the project design, programming, and presentation. You will demonstrate your program to the class as part of your presentation, taking no more than two of your allowed 12-15 minutes to do so. There will be 3 minutes of QA for each presentation beyond the 15 minutes maximum allocated for the presentation.

This assignment is designed to enable you to:

- Study and implement a new algorithm, or implement a familiar algorithm, to gain depth of knowledge in an area of the course that interests you most;
- Broaden the base of the problems and algorithms with which you are familiar, by listening to other Project Group presentations; and
- Practice the important skills of designing and delivering an Oral Presentation of your work;

II. GROUP AND TOPIC SELECTION

Each group should send a short email to the instructor with the names of the two group members and the proposed topic specifying the algorithm(s) to be implemented. You should also describe briefly what you hope to investigate about the algorithm(s) and how you intend to do so. Send this email to the instructor (at lyu@cs.binghamton.edu), with subject "**CS375 Project Proposal**", by ***start of class, Wednesday November 8***. All project topics will be approved and finalized by ***start of class, Friday November 10***.

When multiple groups bit on the same topic, priority of choice will be given to the group which proposes earlier. In some circumstances, we will allow two groups to work on the same topic independently.

Oral Presentations will be held during the classes on ***November 29, December 1***, ***4***, ***6***, and ***8***. The order of the presentations will be decided by the instructor. Additional due dates on presentation slides, schedules of final presentations, and other deliverables for the project will be released later.

III. ROJECTS

Each project should implement some algorithm(s) and investigate certain aspects and behavior of the algorithm(s). The algorithm could already have been covered in class, or it could be an algorithm that we did not cover, but is described in the textbook (including those in the Problems or Exercises sections). You may *not* choose an algorithm that you implemented for one of the programming assignments, obviously; there are no other restrictions on your selection of a project topic.

Some problem may have different algorithm solutions. For example, the maximum-subarray problem can be solved by divide and conquer and dynamic programming. The 0-1 knapsack problem can be solved by dynamic programming, backtracking, or branch-and-bound. You may implement two algorithms representing different solutions and study their behaviors under different input instances.

Some algorithms may have various parameters, alternative components, and implementation strategies. For example, there are different ways to perform PARTITION in QUICKSORT. You may decide to test an algorithm with different parameter values or components to see how the algorithm behaves in terms of running time. You may also try

different inputs (such as arrays with different lengths, different types, and different input permutations) to see how your algorithm behaves. Testing and experimenting with your algorithm are important requirements of this project.

Some algorithms can be fine-tuned to make them more efficient. For example, for dynamic programming algorithms, some entries in the table(s) need not be computed and sometimes the procedures for the 3rd and 4th steps can be combined.

You can also consider improving an algorithm as part of your investigation; in this case you should design experiments to test the extent of your improvement.

IV. RESENTATION

Every student in the class should attend all presentations and participate in the discussion and evaluation of a presentation. Each Project Group will give an In-Class Presentation at the end of the semester. Your presentation should cover the following aspects:

1. ***Algorithm Purpose***: Clearly describe the *problem* that your algorithm is designed to solve. Use an example to illustrate the problem, if possible.
2. ***Algorithm***: Describe the basic idea of the algorithm; show and describe pseudo-code. Your pseudo-code should be in the style of the text book, should be extremely neat and clear, and should use enough abstraction so that it fits on one slide.
3. ***Algorithm Analysis***: What are the interesting features of the algorithm? What is its time complexity, and why?
4. ***Implementation***: What language did you use for your implementation? Discuss key data structures, classes, and functions used in the implementation.
5. ***Problem Statement***: What question did you set out to answer about your algorithm? Why is this an interesting and important question? (Note that you may choose to include this part of your presentation earlier or later in your talk, depending on your project.)
6. ***Demo***: A short demonstration of your program. Have this ready to go; do not waste time "setting up." You may decide to provide static screen shots and sample output instead of running your program "live", to ensure a smooth presentation.
7. ***Experimental Plan***: Describe the dataset that you used to test the algorithm. How did you generate it? What characteristics does it have, and why? What did you decide to vary in the input set, and why?
8. ***Results***: What did you learn from testing your algorithm?
9. ***Limitations and Future Work***: What limitations does your project currently exhibit? If you had another month to work on the project, how would you improve your project? What additional tests would you run?
10. **Concluding remarks** and discuss role of each group member.

Managing your time to fit all of this within 12-15 minutes will be difficult. Limit the scope of detail that you present, and practice your presentation! We strongly suggest that you address each of these topics explicitly in the slides, in the order provided. In other words, you should have a slide titled "Problem Statement", another titled "Experimental Plan", etc. Some of the topics may be combined into one slide. Note that an average of 1-1.5 minute per topic above would produce a talk of appropriate length.

V. EVALUATION

The project and presentation counts for a total of 10% of the final grade, and will be evaluated based on the presentation in class by both peer (classmates) evaluation (50% weight) and instructor evaluation (50% weight). For each presentation, everyone in the audience will assign a score of 1-5 (representing poor, fair, good, very good, and excellent) for each of the following four categories, and then decide an overall score based on the average of the four scores. The final score of a group by peer evaluation will be decided by the average of all overall scores from the student audience excluding the highest three and lowest three overall scores (to avoid possible outliers in the scores).

1) ***Difficult level of the project:*** the difficulty of your algorithm's implementation, and the sophistication of the experimentation with the algorithm that you carried out. Please note that we are not looking for overly difficult and ambitious projects, but at the same time your algorithm and experiments cannot be too simple;

2) ***Knowledge of the problem and algorithm(s):*** how well you understand the problem and algorithm(s) as demonstrated from your presentation;

3) ***Effectiveness of the implementation and experiments:*** how well your code and test cases work, address your question, and draw a correct and useful conclusion;

4) ***Quality of the presentation:*** were the slides professional, helpful, and complete, according to project specs? How clearly and effectively did you communicate during your presentation, and how well did you answer the questions from the audience? How well did your presentation follow the time limit?

VI. CANDIDATE TOPICS

The list below provides some candidate topics you may choose from. The difficult level of these topics is acceptable (not too simple), but is relatively low. However, the difficulty level of the overall project can go up with increasing sophistication of the experimentation with the algorithm(s). You may also combine two related topics (for example, 9 and 10) together to increase the difficulty level of the project.

1. Sorting: implement and compare insertion sort, merge sort, quick sort, and a randomized version of quick sort (see Cormen book Chapter 7.3).
2. Maximum-subarray problem: implement and compare two algorithms: one using divide and conquer and one using dynamic programming.
3. Rod-cutting problem: implement and compare two algorithms using dynamic programming (bottom up) and recursive with memorization.
4. 0-1 Knapsack problem: implement and compare two algorithms using dynamic programming (bottom up) and recursive with memorization.
5. 0-1 Knapsack problem: implement and compare the brute-force algorithm using depth-first search and an algorithm using backtracking.
6. Weighted activity selection problem: implement an algorithm using dynamic programming.
7. Graph traversal: implement BFS to produce BFS tree for both directional and unidirectional graphs.
8. Graph traversal: implement DFS to produce DFS tree for both directional and unidirectional graphs.
9. Minimum spanning tree: implement the Kruskal's algorithm.
10. Minimum spanning tree: implement the Prim's algorithm.
11. Single source shortest path: implement the Dijkstra's algorithm.
12. All-pair shortest path problem: implement the Floyd-Warshall algorithm.
13. Traveling salesman problem: implement and compare the brute-force algorithm and the best-first search algorithm.