Group 01 Members I
First1 Last1
First2 Last2
First3 Last3
First4 Last4

Computer Science 1 – Group Meeting Report

Group Number: Rec11-G01

Meeting Dates and Times:

- **During Recitation**:
  - September 12, 19, 26, October 3, 10 (9:30am-10:20am)
- **Outside Recitation**:
  - October 12, 2023 (12:00 pm – 3:30 pm)

Key Strategies to Prepare for Exam:

One of our key strategies to prepare for this exam was to practice past class quizzes and foundation exams. We figured that these problems would have a similar format to what would be on the upcoming exam, preparing us for how to approach these problems and what to expect. We also assessed our weakest areas of understanding so we could optimize our time by focusing most on those areas. For the areas that each member was weak on, we as a team formulated a comprehensive gameplan. If this game plan was applied to any topic it would yield an increased understanding of a wide range of topics.

The first and most obvious tactic that we constructed was going back to the basics. A lot of the problems found on the previous foundation exam assume that the student understands basic concepts that may not have been taught in the class. The first and most obvious assumption that the test makes is that each student has a profound understanding of the concepts of algebra and calculus. The best example I can think of is the simplification of 1/10! * 12!. If an individual taking this course has not dealt with factorials they would be lost. Yet, the exams expect that students know this and many more concepts like the back of their hand. So, when it comes to the math problems found on the practice exams it would be best for individuals in our group to go back and brush up on the intricacies of both algebra and calculus. When it comes to coding problems, this expectation of the basics is also expected. A great example is the run time of the strlen() function or the what strcmp() function returns. Though the course does not teach these

intricate details it is the expectation that the student knows basic coding knowledge. Without this basic knowledge a student will find it near impossible to understand the basic foundation of this class. A step higher than the basics is the foundations. To understand the foundations one needs to understand the basics but to complete intricate code without the foundations you will be also lost. So the next strategy is to look back at the foundations. Even the small things that may have slipped your mind like dynamic memory allocation. Even things like changing an array of letters in an array to their respective numbers. Looking at the core concepts of coding apart from syntax will solidify our understanding of the foundations of this course.

The next game plan was adapting and playing with these core concepts of the course. The problems found on the exams not only test you on the foundations, it tests your ability to adapt to each scenario. The only way to get the knowledge of adaptation is practice. Suggesting the conventional way of practicing in this course is just the first step. Continuing to do foundation exam practice won't give you the level of adaptation necessary to succeed. So we suggest coding up and creating programs that test your understanding of the fundamentals. Writing just one function won't help on the same level as writing a multitude of functions that build upon each other and truly test your understanding of algorithms, data structures and or the programming language of C.

Another strategy that would solidify our understanding is breaking the status quo. In essence this means attempting to disprove or break some of the concepts found in this course. Try doing a factorial problem iteratively. Try writing an infinite recursive statement and see why it's infinite. Being able to break the concepts that we are taught will allow you to understand why these concepts were taught in the first place. This also helps with debugging and problem solving skills. Which is a big half of this course.

Lastly, all these strategies were recommended under our time managing strategy. Time management is a critical aspect of any plan of action. Creating a structured timeline involves setting deadlines for each goal, breaking down long-term objectives into smaller, manageable tasks, and allocating specific time frames for each activity. Adhering to this timeline ensures that there is no procrastination and that steady progress is made toward the ultimate goal. This ultimate goal is succeeding in this course by passing the exam.

Group Activities Overview:

We attended two different study sessions: During recitation and another outside of class hours.

During Recitation:

We tackled the following given problems during recitation; linked lists, recurrence relations, recursion, and the sample exam questions. Each member worked on the problems individually, then once every member was finished, we then compared our solutions and discussed our logic and reasonings. We also discovered and found useful how topics from earlier in the semester can be carried over or be rewritten to be much more simplified and efficient. For example, we found that many functions can be written with fewer lines and more efficiently by implementing recursion. While recursion may have been difficult for many of the members, it was good practice to rethink how a given function, like in problem 1 of the recursion recitation sheet, could be rewritten recursively as practicing these skills and process of thinking could carry over towards not just a better exam grade, but towards future programming assignments.

It is also notable that while we learned the concept of linked lists prior to recursion, it was good practice to first learn how to write linked list functions without recursion, then later on learn how to write it recursively. Broadening the different ways to write the same function and recognizing more efficient code is a very useful skill to have that will carry over throughout the semester.

Outside Recitation:

Outside of recitation, we worked through problems within previous course quiz material, foundation exams, and the exam review sheet. Because the timing of the meeting was after part A of the exam, we prioritized a focus towards the topics that were going to be expected on exam B. We communicated areas of difficulty with one another, shifting our focus towards summations, recursion, recurrence relations and algorithm analysis. Taking a similar approach to how we studied during recitation, we went over many problems, then discussed our answers and areas of confusion. If one member was confused about a certain approach or answer, the other members would then explain and resolve the confusion. Collectively, recursion and summations were the most difficult. There were different approaches towards solving the same problem, although it was notable that being able to discuss with one another what exactly the problem is asking was incredibly useful, as sometimes these problems need to be thoroughly understood and broken down before attempting to write anything down. This happened most commonly with many recursion problems we went through.

Summations was the next topic with the most difficulty, and the members of this group had a differing range of mathematical abilities. Members who were most suited in math were the most helpful at explaining summation and recurrence problems. There were instances where recurrence relations utilized a summation when iterating through them, so developing a deeper understanding of summations was important for deepening our ability in recurrence relations.

Lastly, after developing a better understanding of summations and recurrence relations,

we were then able to discuss algorithm analysis before ending off our meeting. We went over binary search, and how we can write it in terms of T(n), which would then assist in getting the runtime in Big-O notation. We were able to identify what notable lines of code were performing and identify that it was running in T(n/2) times.

Detailed Group Work:

The following are links to the material and problems we worked through:

- Quiz 3 - Part A (Spring 2022)
    - Problems: 1, 2, and 3
- Quiz 3 - Part B (Spring 2022)
    - Problems: 1, 2 and 3
- Quiz 2 - Part A (Spring 2022)
    - Problems: 1, 3 and 5
- Quiz 2 - Part B (Spring 2022)
    - Problems: 1, 3, and 5
- Foundation Exam (January 2019)
    - Page 11: Algorithm Analysis
- Class Notes (October 5, 2023)
    - Focus: Reviewing master theorem
- Exam 1 Review

Summary:

The focus of our October 12th meeting was preparing for part 2 of the exam. We first looked at two versions of quiz 3 from spring 2022. Specifically we solved three questions from each that pertained to predicting runtime of algorithms where the Big O value is known, solving the runtime of recurrence relations using the iteration technique, and evaluating summations.

From quiz 2 A and B, we went over the questions that involved writing recursive algorithms, and we also took a look at the questions regarding the odometer code, and the order in which said algorithm outputs numbers.

We finally took a look at a past foundation exam to review a problem that involves reading code and determining the Big O runtime. For this we looked at question 1 from the Algorithms and Analysis Tools Exam from 2019. We also checked the class notes on runtime analysis to go over the master theorem, so that we might be better equipped to check our work on

the exam.

We used the Exam Review notes from the course website to help inform our decisions on what might be helpful to review. In retrospect, the problems we solved during our meeting were quite relevant to the exam.

Notable Discoveries:

When reviewing quiz 3 2022, we made sure to note the important summation identities, specifically the following:

$$\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1} \qquad \sum_{i=0}^{n} i = \frac{n(n+1)}{2} \qquad \sum_{i=1}^{n} x = n \cdot x$$

In general, quiz 3 allowed us to brush up on our algebra, review important properties of summations (such as rules for expanding summations as well as multiplying summations by constants), and improve our understanding of big O properties (for instance: $O(n^{\log 5}) < O(n)$ because $\log 5 < 1$).

Our review of quiz 2 may not have led to any noteworthy "discoveries" but it was quite fruitful. Quickly formulating recursive functions proved difficult for every member of the group, and working together to determine how to form base cases and recursive calls for the various problems was very helpful. We did conclude that it might be helpful to copy the recursive brute-force algorithms on our formula sheet for the exam, which in hindsight was a very good idea.

When looking over the algorithm analysis problem from the foundation exam, we noted that it may prove useful to memorize (and understand) the runtime of a few prominent algorithms, namely: loops, binary search, and standard functions such as strlen(). We also noted the importance of understanding how nesting said functions within one another can affect runtime. For instance, a for loop that is followed by a binary search will run in O(n) + O(log n) = O(n) time, while a for loop with a nested binary search will run in O(n log n) time.

We learned about more summation identities and deepened our understanding of code written in terms of T(n), specifically using Binary Search as an example. We also recognized how useful Master Theorem can be for recurrence problems. Additionally, Benjamin assisted the group towards understanding multiple concepts and problems, one being permutation. He explained in detail why in specifically problem 5 in the previous Spring 2022 Quiz 2-B concerning permutations, he explained how the problem was solved and why the left side of the permutation remained unchanged until "4".

Member Reflections:

First1 Last1:

For me the study session gave me a great insight on two important problems that I was having trouble with. Those two weak points would be writing recursive and recurrence relations. The biggest issue I faced was simply recursion. Though I understood the principle of recursion and its definition. I still faced a great deal of trouble when writing recursive functions. I could not imagine or articulate any solutions for the recursive questions that we tackled, as a group. The questions to me seemed so complicated. I stumbled for minutes trying to think of a game plan to tackle some of these questions, to no avail. It was frustrating as my team members were able to analyze and come up with a solution relatively quickly. Even when a solution was not given, my group members were still able to write something down while I was stuck brainstorming. At this point I deferred to my group mates and questioned their tactics when it came to writing a recursive statement. This is when I learned the pattern to these recursive questions. Though each solution is relative to the problem after the completion of many recursive problems I was able to spot a pattern. The base statement is what I assumed the end of the function should check and calculate for. In a permutation function the base case checks if the permutation is complete. Now that I got the base case down, working on the return function or the breakdown of the recursion was my next obstacle. Though it's all relative, my groupmate gave me insight on how most recursive statements are broken down. With this help I felt more confident about solving problems that contain recursion.

Recurrence relations was a section I believe I was proficient in but a small mistake was causing me to get the wrong answer to some of the recurrence relations. This small mistake was easily changed but if I didn't get the guidance from my group I believe I would have never found it. The small but crucial thing I was having was when I performed the iteration I did not divide by all functions of N. Instead I was dividing all Ns. Now this seems like the same thing but I was getting completely different results due to this mistake. For example if I was performing a recurrence relation that looked like this: 2T(n/2)+(n)^2. I would divide every n by two to get 2[2t(n/4)+(n^2)/2]+n^2. This is fundamentally wrong. n^2 is a function of n and I was dividing n^2 by 2 when I was supposed to just be dividing the n inside the function n^2 to get (n/2)^2. This caused me to get every single recurrence relation problem wrong that contained this variable. When my groupmate looked over it, he was able to help me find my mistake and all in all improved my understanding of every recurrence relation problem I face.

First2 Last2:

During the study groups, and most notably the most recent one, I focused mostly on algorithmic topics like summations, recurrence relations, and finding the runtimes for algorithms, mostly because math is not my strong suit. I also worked on recursion problems, as they also tend to be difficult for me. The other group members were very helpful, as they helped me work through problems and corrected my logic. I was shown useful summation identities and reminded of the Master Theorem that may help me for certain recurrence problems. Overall, I found that I best study by working through many problems, learning from the mistakes I make. This group was incredibly helpful because I received explanations as to why an answer is the way it is and the

logic behind the reasoning, which would have taken me much longer to discover on my own.

I also noticed that I struggle with fully understanding what a problem is wanting to perform. Not having a full understanding led me to making many mistakes that could have been prevented if I spent more time to really break down and fully understand the problem. I would make mistakes writing the base case because I would miss what the problem asked. Rereading and really thinking about the logic before writing code helped me make fewer mistakes. It takes me a lot of time to do function writing problems because I often have difficulty thinking about the logic behind it. While I am doing more review and practice of these problems specifically at home, like how I mentioned earlier, being in a group setting really helped me save a lot of time because members explained things that would have taken me longer to do myself.

First3 Last3:

I think our greatest strength was our optimal use of time. We went through a good number of problems, and while many would cover the same topics, the format of each problem was distinct. I felt we prepared for pretty much every type of question that could have been asked of us, and that we didn't waste time studying irrelevant topics. This was even more so apparent in hindsight, as every question that ended up on the exam had a very similar analog in the problems we reviewed. As previously mentioned, we even made a point to include the odometer algorithm in our notes.

It is hard to put my finger on any one thing we did that I found particularly helpful. For the most part I think I needed to sharpen my skills across the board, and we addressed that with the diversity of problems we tackled. It was very helpful to have people who could point out my errors in logic whenever I failed to find the right answer. Specific errors I recall being assisted with included the correct ordering of Big O terms as well as various arithmetic errors when using the iteration technique.

In general, I found the members of the group very helpful and easy to work with. Everyone was patient, diligent, and easy to communicate with. I perceived that the time we spent studying together was far more productive than any time I would have spent on my own, and I think the others felt the same. I think my biggest takeaway on how to better prepare for exams in the future is more study groups.

First4 Last4:

During our study session, I spent a fair amount of time reviewing recursion and recurrence relations, as these are topics that are somewhat more familiar to me. However, I made sure to put extra effort into understanding summations, as this was a topic I have struggled with in the past.

Reviewing summations helped to refresh my memory of the topic, as I was able to remember and apply the summation identities I had learned during Professor Guha's lectures. During my practice there were many instances where my knowledge of the topic was tested. One of these

instances revolved around a question asking to determine which summation expression of the given two resulted in a greater sum:

$$\sum_{i=1}^{n} i2\text{^}i \text{ or } \left(\sum_{i=1}^{n} i\right)*\left(\sum_{i=1}^{n} 2\text{^}i\right).$$

Although at first glance I had assumed that the former summation equation would result in the larger sum, some assistance from my groupmate had revealed the truth to me through the illustration of the equation's final product. The first expression would have resulted in a summation that appears as:

$$1*2\text{^}1 + 2*2\text{^}2 + \ldots + n*2\text{^}n$$

Whereas the latter expression resulted in:

$$2\text{^}1 * (1+2+...+n) + 2\text{^}2 * (1+2+...+n) + \ldots + 2\text{^}n * (1+2+...+n)$$

As one can see, the latter expression has a much larger sum than the former due to it essentially being a summation of summations.

In total, working with my group mates was overall a very pleasant experience. Everyone in my group was respectful, diligent, intelligent, helpful, and committed to the task at hand. Whenever we had questions, if one of us was informed on the topic, they gladly showed us how to find the answer. If none of us knew what was correct, we worked together to find the answer in our notes, example codes, and past quizzes or exams. Ultimately, I find working with my group to be a considerably more efficient form of studying when compared to individual study.