Habib University
Summer Tehqiq Research Project 1

Developing a Course on Competitive Programming

Research Report

Supervisor: Dr.Waqar Saleem

Iqra Ahmed - ia07674
Areesha Amir - aa07613
Ali Muhammad Asad - aa07190

Contents

1 Introduction

Programming competitions have become increasingly popular over the years, with many participants, both professionals and amateurs, competing for recognition and prizes. Competitive programming competitions [39] involve solving algorithmic problems within a specified amount of time, typically a few hours. These competitions are designed to test the participants' problem-solving and programming skills under time pressure.

Competitive programming has become a global phenomenon, with contests held regularly in various countries. Often conducted as a team sport, competitive programming is a valuable tool for improving programming skills, particularly among undergraduate students in computer science programs. Competing requires a high level of knowledge of data structures, algorithms, and problem-solving skills, which are fundamental to computer science education. It provides an engaging and challenging environment for students to develop their programming skills and apply them to real-world problems. Technology companies like Google, Microsoft, Amazon, Apple, and Facebook, as well as local companies like Folio3, securiti.ai, Creative Chaos, and Bazaar, famously test potential employees for exactly these skills [23], often in settings similar to competitive programming. Some of these companies run their own competitions [7, 24, 11, 1, 2] in order to promote and identify talented programmers.

Habib University's emphasis on critical thinking, creativity, and problem-solving has naturally led DSSE students to partake in competitive programming. Specifically, they competed in International Collegiate Programming Contest (ICPC) [28], which is the largest and most prestigious computer programming contest for students at the undergraduate level. Colleges are represented by teams of three students each under the tutelage of a coach who is typically a faculty member. The annual competition proceeds in rounds—national, regional, and a world final. DSSE students have qualified for and appeared in the top 10 of the regional round for three consecutive years [10, 16]. Recognizing the value of competitive programming in a computer science education, several universities offer formal instruction in it.

With this research, we aim to develop a Computer Science elective course on competitive programming for the students at Habib University and propose metrics to measure its effectiveness. The course will be offered to students with existing exposure to programming, data structures, algorithms, and programming languages. The course will provide students with the opportunity to enhance their algorithmic problem-solving skills, collaborate with peers, and participate in programming contests, thereby preparing them for real-world programming challenges. It will also allow students a creative outlet for their studies in abstract computer science topics like data structures and algorithms. Through inclusive activities, it will indirectly serve to popularize competitive programming on campus and prepare teams for ICPC. This in turn will make our students more attractive to industry and to graduate schools.

The following questions guide the design process of our course:

1. What topics are essential for a competitive programming course, and how should their order be kept to ensure a progressive increase in difficulty and learning?

2. How do instructors facilitate discussions and problem-solving sessions among students in a competitive programming class?

3. What are the prevalent online platforms where competitive programming students can participate in coding contests and access problem archives?

4. How were the assessments developed and integrated within the course?

5. What are some ways to measure the effectiveness of a course that can be utilized for our course?

The design of this course has been conducted with our constraints in mind. Our students and faculty have minimal exposure to competitive programming. There is a shortage of available faculty to deliver such a course. Our student body hails from diverse backgrounds and many of them lose a good portion of their day commuting to and from campus, and attending to their domestic responsibilities. A significant part of their curriculum consists of courses that are not directly related to computer science. The number of computer labs is not keeping pace with the number of students on campus. The infrastructure— electricity, internet—outside campus cannot be relied on completely. Keeping these constraints in view we come up with a course design to suit our students that can be implemented efficiently with our limited resources.

2 Literature Review

Courses on competitive programming have started to appear at some universities [26, 35, 33, 37, 31, 34, 9, 17, 36, 30]. The instructors are usually faculty members who have themselves been participants in ICPC or its school-level counterpart, International Olympiad in Informatics (IOI) [19], or are long-time coaches for these events [14, 21, 38, 25, 8, 4, 15]. Some of the enrolled students [32] similarly have experience in IOI, ICPC, International Mathematics Olympiad (IMO) [18], or many of the online competitive programming platforms [13, 27, 6, 20, 12, 29, 5]. Some instructors and their courses have amassed significant reputations. The instructors are invited by other universities to offer a version of the course [40], or students travel from other countries just to take the course [22].

The pedagogical value of these courses in their respective curricula is to develop the students' algorithmic, programming, and problem-solving skills. But they also serve to distinguish talented programmers and identify motivated students. These students can then be trained more intensively to prepare them to represent the university at ICPC. Having a dedicated module also enables the formation of strong teams by grouping together the students that possess skills that complement each other. Further, it helps students develop and polish useful skills that can be learned only in a competitive environment. Having a dedicated module can also be useful in acquiring relevant resources for the purpose of preparing students for such competitions [54].

A typical ICPC-style programming competition starts by assigning a common set of problems to all the participating teams. They then have a specified amount of time to solve the problems and submit their solutions in real time over a network connection to an automated checking system. The system returns a result of correct or incorrect and the team can choose to correct their solution and resubmit or to attempt and submit the solution to another problem. Each team is scored based on the number of problems it solves correctly, and the time and number of attempts taken to submit the correct solutions. Teams that solve more problems quicker and in a smaller number of attempts are ranked higher.

The problems present scenarios that can be mapped to one or more algorithmic techniques. There may be more than one way to solve a problem and students are required to pick the solution that is the most efficient. Other teams are simultaneously and independently solving the same problems. The scores are updated in real-time as the submissions happen and a live scoreboard with the score of all teams is centrally displayed so as to be visible to all the participating teams. The environment is generally tense and highly competitive.

Collectively, courses on competitive programming achieve multiple outcomes. By teaching algorithmic techniques and concepts, they provide the students with the necessary intellectual background. Through curated problems on specific topics, they train the students in those topics. Through individual assessments, they ensure that each student has adequate practice and stands to become a valuable team member.

Frequent team competitions in a course serve to develop team dynamics and expose students to more realistic problems which do not rely on a single technique. They also acclimatize students to the high-pressure environment of these contests. Extensive practice ensures that the students are familiar with a rich, diverse, and voluminous set of problems so that they may relate any new problem to one that they have previously solved.

Online platforms to deliver problems and assess solutions are imperative for these courses. These serve the dual role of question banks for practice and assessments, and of automated judges for the students' solutions during assessments and mock contests. Some courses have developed their own platforms while others use existing competitive programming platforms, of which there are several that are freely available online.

The field of algorithms is deep and vast. A semester-long course covers but a small portion.

Some universities, therefore, offer a sequence of courses on competitive programming that build upon the usual introductory programming course [33].

The implementation of such a course also comes with its own set of challenges. The research paper by faculty members at NUS elaborates on some of the challenges that come with implementing a dedicated module. They also suggest some ways on how these problems can be tackled along with ways these were addressed during the implementation of the course at their university [54]. Four key challenge areas were identified. Below is a list of the probable challenges along with a possible workaround/solution as was implemented at NUS.

1. Manpower: First and foremost to running the course effectively is the need for experienced instructors and teaching assistants. For NUS, this wasn't much of an issue as they had several Ph.D. students and Research Assistants at the university that were ex-ICPC World Finalists. Further, the faculty consisted of several devoted professors/lecturers that shared domain-specific knowledge.

2. Students: One of the aims of having a dedicated module is to identify gifted programmers. However, grade-conscious students may refrain from enrolling in such a course. Advertising the course well at the start of the academic year can help tackle this. Further, it is likely that a cohort would comprise students with varying levels of prior experience. Teaching all of these together is then a little tricky. A two-tiered approach can help with this challenge as well as encourage talented students with little prior exposure to still take the course.

3. Assessment: Producing original problems for the course is yet another challenge. This is necessary as using only problems from online databases can result in incorrect scores on assessments. To tackle this, teaching staff may design original problems from scratch, or they may re-utilize existing problems through paraphrasing. Another workaround to this can be to assign students from each cohort an assignment to design problems, which can then be utilized to test their juniors.

4. Course Materials: There is a need for a course-specific textbook to assist in learning. With the ever-evolving nature of problems at these competitions, it may be useful to use real competition problems alongside existing textbooks to ensure the best preparation for the students.

The logical progression following the development of a course involves ensuring its efficacy and alignment with its intended purpose. Although there exists a gap in research concerning quantitative methods for accomplishing this, a range of qualitative approaches have been identified and assessed [48, 49, 50, 51, 52, 53]. Among the various qualitative strategies accessible, a consensus regarding the optimal approach remains difficult to capture. While some proponents advocate that significant insights can be drawn from student surveys and questionnaires, others contend that the most precise and reliable responses are obtained through direct student interviews. Certain experts advocate for a blended methodology, advocating the amalgamation of diverse data sources. This approach encompasses student appraisals, peer evaluations, self-assessment, student interviews, alumni feedback, measurements of learning outcomes, as well as classroom dynamics such as student attitudes and engagement.

One paper, in particular, offers a different approach. This approach involves implementation of Edumetric tests [52]. These tests consist primarily of questions that a substantial number of students answer incorrectly before undertaking the course, but subsequently answer correctly after completing it. These tests are administered both at the commencement and conclusion of each academic term, with none of the test questions being explicitly referenced during class instruction. Incorporating such assessments aids in gauging the extent of learning achieved as

a result of undertaking the course. When employed in combination with data obtained from student surveys, these tests can enhance the precision of evaluation scores.

3 Research Methodology

The research was conducted in four key stages: course/literature review, testing and identification of resources, course design, and development of effectiveness metrics for our course. Below we detail the methodology employed through each stage:

1. **Course/literature review:**

   We started with the identification of relevant courses on competitive programming through a thorough online search. The courses were then reviewed to identify good practices for teaching a specialized course in competitive programming to undergraduate students. Relevant literature was also consulted to understand the difficulties that come with teaching a specialized course and the ways to overcome them.

2. **Testing and resource identification:**

   The reference material collected through Step 1 was assessed to derive relevant information for the design of our course. The reference material comprised mainly of three types: existing courses, platforms for hosting programming contests, course-specific textbooks, and research papers.

   The syllabi of existing courses on competitive programming were carefully reviewed to identify patterns in the course content (such as commonly taught topics, the order in which these topics were organized), teaching methodologies employed, the nature and frequency of assessments, and the grade distribution for the course. Parts of these courses along with their associated assessments were also attempted to assist in the process.

   The identified platforms were tested through attempting problems and contests on them and their general feasibility for a course was assessed.

   Relevant textbooks were reviewed to assess them on exposition, coverage, and accessibility so one specific textbook can be identified for use in our course.

   Research papers were reviewed to gain insights from the experience of those who have previously designed and implemented such a course.

3. **Course design:**

   Based on findings from the first two steps, we finalized a list of topics to be covered in the proposed course, accompanying teaching and assessment material (in-class worksheets and problem sets), a lecture plan, and an assessment plan for the course as well as tentative grade distribution. Finally, the course syllabus was designed, a copy of which is attached in the appendix.

4. **Effectiveness metrics:**

   Relevant literature was surveyed to identify tools to measure the effectiveness of our course. The pros and cons of these different methods were analysed to determine their feasibility alongside our course.

4 Results/Findings and Discussion

In this section, we address the results and findings of our research and the key research questions outlined in the study. Each subsection corresponds to a different question, providing insights into our approach, analysis, and outcomes. By systematically exploring the essential components of a competitive programming course, along with considerations for course materials, assessments, and effectiveness measurement, we aim to contribute to the field of algorithmic education and curriculum design.

Through a comprehensive comparative analysis of existing competitive programming courses, we have formulated evidence-based responses to our research questions. The findings shed light on the optimal structuring of topics, the selection of course materials, and the formulation of effective assessments. Furthermore, we discuss potential methodologies for measuring the effectiveness of our course, ensuring a well-rounded exploration of the subject.

4.1 What topics are essential for a competitive programming course, and how should their order be kept to ensure a progressive increase in difficulty and learning?

Competitive Programming equips students with algorithmic-problem solving and critical thinking skills crucial for excelling in programming content and real-world applications. The course curriculum should cover a wide array of topics, carefully sequenced to facilitate a gradual increase in complexity, enabling students to master foundational concepts before tackling advanced challenges. Since one of the aims of this research is to prepare students for programming competitions as well, therefore, the topics were chosen while keeping the International Olympiad in Informatics (IOI) Syllabus 2023 [56] in mind, and also in renowned competitive programming competitions such as International Collegiate Programming Contest (ICPC) [55]. In addition, a thorough analysis of different courses being offered at various universities was also done.

From our findings and analysis, we were able to produce a matrix with a list of topics on the rows side, and corresponding courses on the columns side:

Courses / Topics	Competitive Learning Baylor University	Competitive Programming National University of Singapore	Competitive Programming University of Porto	Introduction to Programming Contests Stanford University	Problem Solving Programming Strategies TAMU	Competitive Algorithmic Programming Illinois	coding competitions: secrets of champions ITMO (edx)	Introduction to Programming Contests California	ACM Programming Contest Prep Washington	Competitive Programming Texas at Austin	A Competitive Programming Course Reykjavik	Contest Problem Solving UCF
Intro	1	1	1	1	1	1	1	1	1	1	1	1
Ad Hoc Simulation		1				1		1				
Mathematics and number theory	1	1		1		1			1	1	1	1
Data Structures and Sublinear Complexity Data Structures (map, set, priority queue)	1	1	1	1	1	1	1		1		1	
Binary Search, searching, divide and conquer, and sorting	1	1	1		1	1	1		1	1		1
Recursion and backtracking									1			
Brute Force												1
Greedy Algorithms		1			1	1				1	1	1
Dynamic Programming	1	1	1	1	1	1		1	1	1	1	1
Combinatorics		1		1	1	1		1	1			
Graphs and Graph algorithms (including BFS DFS)	1	1	1	1	1	1	1	1	1	1	1	1
Shortest path algorithms	1	1		1	1	1	1	1	1	1	1	1
Network flow problems				1	1	1		1			1	1
Strings	1	1	1	1	1	1			1	1	1	
Computational Geometry and Geomtric Algorithms	1	1	1	1	1			1			1	1
Trees	1	1								1		1

Figure 1: Course Topic Matrix for Essential and Commonly Included Topics

A "1" shows that a certain topic was taught or included in a certain course and a blank or empty space shows that a certain topic wasn't taught or introduced in a certain course. The matrix, however, does not show all the 17 courses that we analyzed due to space constraints, however, it does show all the relevant topics from our findings.

From this matrix, we were able to further analyze for each topic the number of courses that included that certain topic. Thus, for each topic, we had a quantitative analysis of the number of courses that included that topic to be taught in their course offering. Through this analysis, we were able to further refine our search for the essential topics for a competitive programming course as the frequency of a topic would imply its necessity and significance in a competitive programming course. The results can be shown in the chart below:

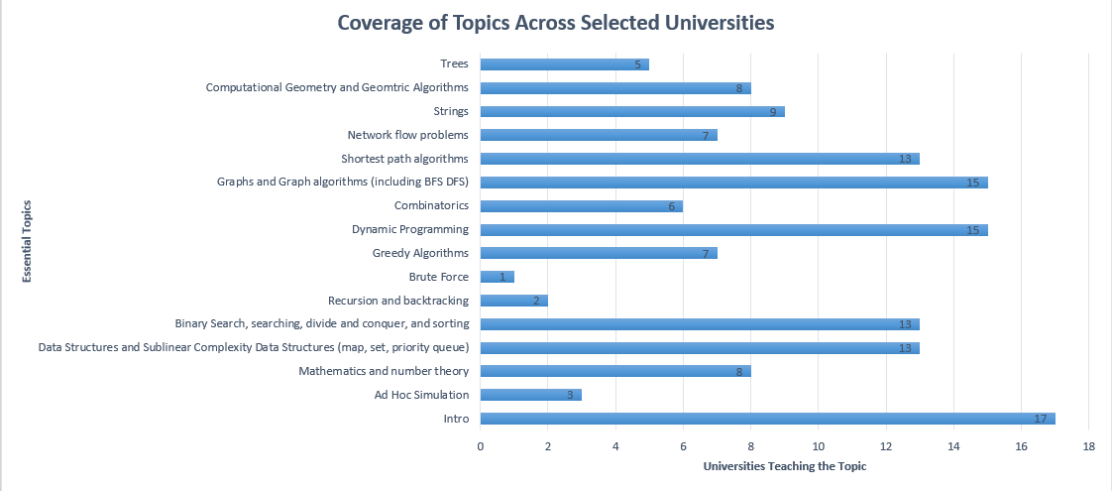


Figure 2: Frequency of Topics in Courses

Through these findings, and in compliance with the IOI Syllabus, and ACM-ICPC curriculum, we were able to select the essential courses for a competitive programming course. Moreover, we individually analyzed the course schedules as well to better understand the structure of the course schedules, for an understanding of their kept order, connections between each topic to the next, and a progressive increase in difficulty and learning. From the list of most essential courses, we found that the schedule could be divided into the following groups:

1. **Introduction, Data Structures, Searching, and Sorting**:
   This group would contain the introductory parts of the course and equip students with the essentials that would be needed in all later parts of the course. This includes basic input-output techniques for efficient input and output, Ad Hoc techniques and simulation to devise basic strategies to solve various problems, libraries in different programming languages, elementary data structures for simpler problems and storing data and moving onto sub-linear and complex data structures for improving efficiency and optimization. Lastly, this group would include searching and sorting algorithms, especially the divide and conquer approach for efficient searching of data/results, and optimized sorting of the data.

2. **Graphs, Trees and related topics:**
   This group would most often have a span of three to four weeks in the courses, starting at basic graphs and graph traversals, basic graph techniques such as BFS (Breadth-First-Search) and DFS (Depth-First-Search), moving onto intermediate graph algorithms, shortest path algorithms, trees, and ending at real-world applications for graph problems most commonly known and introduced as Network Flow Problems. Although not all universities that introduced graphs had network flow problems or trees.

3. **Greedy Algorithms, and optimization:**
   This group would most often contain only Greedy Algorithms and Dynamic Programming. Some universities would offer it after the introductory group but before starting graphs, and some would start with these topics once the graphs would finish, but still, in most cases, they were offered one after the other.

4. **Other Topics:**
   This group mostly contains independent courses that can be introduced in any order after the introductory courses. This group mostly includes topics like Mathematics, Number Theory, Combinatorics, Recursion, Backtracking, and Brute Force (although most universities do not include Recursion, Backtracking, or Brute Force as a separate topic, rather incorporating other topics such as BFS DFS, and Greedy Algorithms), Computational Geometry and Geometric Algorithms, and Strings (in most cases Strings was one of the last topics or introduced after graphs and trees due to string problems containing suffix tries and suffix arrays).

The above grouping of topics was found to be the most effective as the first group starts with all the necessary introductory topics, equipping students with the necessary skills and understanding of techniques and concepts that will thoroughly be used throughout the course. Ad Hoc Techniques and Simulation develop problem-solving, analytical, and critical thinking skills. Data Structures build a solid foundation for more advanced concepts that would require efficient data storing, data access, and manipulation techniques for improving efficiency and optimization. Efficient searching and sorting methods including the divide and conquer approach reduce time in searching and sorting problems. Therefore, by the end of the first group, students would be equipped with the foundations to tackle the remaining topics. Groups 2 and 3 can sometimes exchange places, although we found that keeping Group 3 before Group 2 was more effective as some problems need a greedy or brute force approach which involves making optimal choices. Dynamic Programming would then build on the foundations of greedy strategies, enabling systematic optimization of various solutions mainly by breaking down the problem into sub-parts - usually in a recursive manner, storing the results, and sub-problems optimized to find the overall solution. These techniques would also be used in some graph algorithms as well, hence is more effective to comprehensively introduce this before Group 2. Group 3 would follow by introducing the students to graphs, graph traversal, DFS, BFS, shortest paths, some trees, and real-world scenarios - Network Flow problems. Keeping these groups first would set up the foundational skills as well for many geometrical, computational, mathematical, probabilistic, combinatorial, and string problems as well, hence Group 4 is kept last.

This carefully planned sequence of topics ensures is best to progressively build a student's algorithmic and problem-solving skills while students can gradually internalize the concepts and keep a modest difficulty level. By following this sequence, students can develop a strong foundation and gradually advance their skills, preparing them for various programming competitions and real-world algorithmic challenges.

Following this above sequence of topic groupings, grouping orders, and the essential topics to be included in a competitive programming course following the IOI Syllabus, ICPC Curriculum, and the analysis of the courses offered by different universities, we were able to come up with our own tentative weekly breakdown of topics that can be shown in the table below:

Week	Topics
1	Introduction: Input/Output Techniques + Ad Hoc Simulation
2	Elementary Data Structures, and Libraries in Python and C++
3	Data Structures and sub-linear complexity data structures
4	Searching and Sorting + Problem Solving Paradigm: Divide and Conquer
5	Greedy Algorithms
6	Dynamic Programming
7	Midterm week
8	Graphs (including unweighted), graph traversal, and graph algorithms including BFS and DFS
9	Intermediate graph algorithms + Trees
10	Shortest path algorithms
11	Network Flow
12	Computational Geometry and Geometry Algorithms
13	Strings, string matching, suffix tree, prefix tree
14	Mathematics and Number theory
15	Combinatorics

Table 1: Essential Topics in the Course

4.2 How do instructors facilitate discussions and problem-solving sessions among students in a competitive programming class?

The Competitive Programming Courses that we researched were seen to always be inclined towards the traditional lectures using the slide-based approach where the students get oral lessons during the lectures and get to work on the competitive problems themselves in their home-based assessments. We; however, decided to test out a new approach in the course by flipping the classroom to see if it assists students in equipping the essential skillsets required to master competitive programming more effectively.

Competitive programming is an intellectually stimulating activity that requires rigorous practice and if all the assessments are done outside class, the students might not be able to tackle the challenges fully just from the oral presentations given in class unless they engage in practicing their learning in class. The lectures that are kept around slide-based learning limits students' engagement and are not as effective for students that have different learning pace. Moreover, such traditional lectures go against efficient time utilization since the students will only encounter teaching material in class rather than active learning and such materials are already available to the students so the experience might be nothing new.

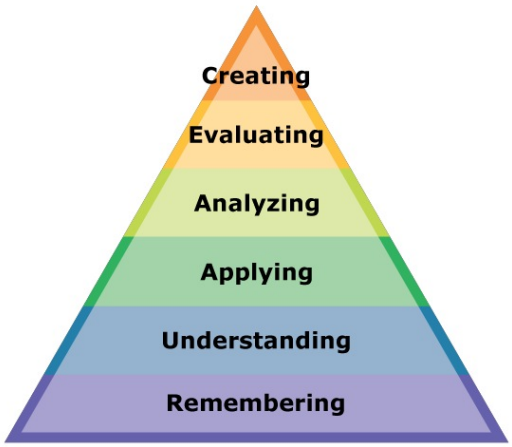


Figure 3: Bloom's Revised Taxanomy
[43]

Bloom's revised taxonomy means that the students are doing the lower levels of cognitive work and are focusing on the higher forms of cognitive work in class, with the support of their peers and instructors. Similarly, in our course design students deal with knowledge and comprehension before class through book readings and other available resources. Whereas they perform application and evaluation during class lectures in the form of worksheets and solving mini-contests. This model contrasts with the traditional model in which "first exposure" via lecture was seen in competitive programming courses offered around the world. CSE109: Introduction to Programming Contests, CS 491: Competitive Algorithmic Programming, and CC3032: Competitive Programming are a few examples of courses that relied on the theoretical exposition of the topics during class lectures. Some of these courses like CSE109 made attendance in lectures optional which might indicate that the lectures might not hold significant information or engagement that students cannot find elsewhere [45].

Eric Mazur, a physicist and educator at Harvard University along with another physicist and professor at Swarthmore College tested out the inverted classroom approach using the Peer Instruction model that required students to first gain exposure prior to class. In class, the students dealt with conceptual questions which would then shape the productive discussions. Mazur and colleagues then published results suggesting their method results in significant learning gains when compared to traditional instruction [43].

Others like Carl Wieman and Richard Hake have also published evidence that flipping a classroom can produce significant learning gains [43]. Although there is no literature on applying such a teaching strategy in a competitive programming course, Richard Hake's study on 2084 students on a course involving algorithmic problem solving proved that interactive engagement methods exhibited learning gains almost two standard deviations higher than traditional courses [42]. Therefore, we decided to incorporate flipped learning within our course. This could not only increase the efficient practice of the student's competitive programming skills but would also utilize the lecture time to the fullest by promoting active learning within the classroom.

The introduction of "The Flipped Classroom Model" in this course is what made this course design stand out the most since it is the first time such a technique is being introduced in a competitive programming course compared to all the courses we were able to find online. The flipped learning method will also allow the instructor to provide close-side guidance to all

the students by identifying where the students struggle while solving a competitive programming problem. Furthermore, the instructor will also be able to distinguish the common problems faced by students on a particular topic or technique which they can then address easily, enriching the learning experience for everyone. This method also makes the students more responsible for their learning where they will be allowed to collaborate more effectively. Lastly, it will ensure the same learning experience for every student regardless of the skills they are equipped with before the course [41].

4.3 What are the prevalent online platforms where competitive programming students can participate in coding contests and access problem archives?

Competitive programming enthusiasts have access to various online platforms that host coding contests and offer an extensive problems archive. These platforms cater to a diverse range of skill levels, interests, topics, and difficulty levels, fostering a global community of programmers. To analyze these platforms, and identify the platforms to use in our course, we did a comparative analysis of the different platforms used in different courses by different universities. Out of those, the most prevalent online platforms along with the total number of universities using them (number of universities that we analyzed) can be shown below:

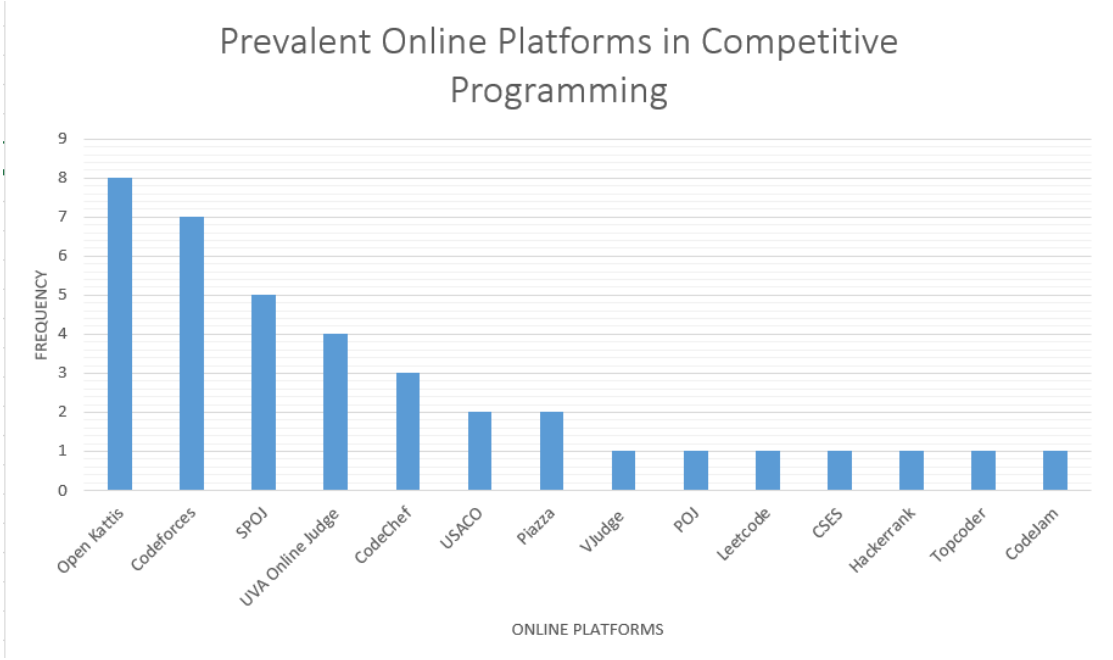


Figure 4: Online Platforms Used by Universities for Competitive Programming

These were the most prevalent online platforms being used in one way or the other. Universities had either set up their own codomain which had their problem sets and assignments, or they would set up timed coding contests on certain platforms, or they included a GitHub repository where the problems could be found linking to either of these platforms. It was observed that many of the universities did not limit themselves to any one platform either, rather, included

two or more platforms for different purposes, such as Codeforces for contests or mids and finals, or Open Kattis for assignments or problem sets, etc. Since Open Kattis and Codeforces were clearly the most used platforms, we mostly explored and analyzed these platforms for our course.

1. **Open Kattis**
   Kattis was the most widely used platform amongst the universities we analyzed. Open Kattis is a widely recognized online platform that serves as a collaborative coding arena for competitive programmers. It provides a nurturing environment, with a very clean, modern, and easy-to-understand interface especially for beginners while supporting many languages. It has an archive of over 2500 problems each with its own difficulty level, covering a wide range of topics that are used in competitive programming contests.
   It also has the option for buying a sub-domain for a university on a per-student-per-course basis. The sub-domain comes with the option of having our own Kattis Page limited to the students that enroll in our course at the university, not only creating our own problem sets and coding contests but also creating and uploading our own problems with their set test cases and solutions. In addition, the student submissions can also be viewed and checked for code logic, plagiarism, etc, while having the option of monitoring student performance.

2. **Codeforces**
   Codeforces was the second most widely used platform amongst the universities we analyzed. Codeforces is also a widely recognized online platform catering to both novices and experienced participants up to global competitions. It also has a clean and modern interface with an archive of over 7000 problems each with its own difficulty level, covering a wide range of topics used in competitive programming contests worldwide. Moreover, Codeforces is the platform used by ICPC making it especially suited for competitive programming contests. It also provides educators to host free virtual contests for their students, fostering a learning-oriented and timed competitive environment.

Research on Kattis indicates that students gain a theoretical advantage when utilizing the platform beyond class hours. Courses utilizing Kattis as an assessment tool received positive feedback from over 80% of students. Moreover, Kattis aligns with test-driven education, bringing the course pedagogy closer to this approach [47]. We leveraged these platform benefits in crafting problem-based assessments for independent study.
Keeping our objectives in mind, we decided to use both these platforms in our course due to their vast problems archive, easy user interface, community involvement, and the option to hold our own contests. However, due to financial constraints, we were unable to purchase our own sub-domain for Open Kattis, therefore, we decided to use both these platforms in different ways. The weekly problem sets were compiled using the problems available on Open Kattis archives, hence, would provide a good interface and practice platform for practice and getting used to competitive programming environments. Codeforces would then be used for the mini-contests as it is best suited to hold an actual timed competitive programming environment. The mid and final contests would also be held on Codeforces.

4.4 How were the assessments developed and integrated within the course?

Since our course design no longer catered to traditional classroom lectures, we required resources that would utilize class time in the most efficient way possible. Furthermore, we also required assessment methods outside the class hours that would prove effective in the student's learning.

The goal of our course is to equip students with the skillsets that will not only help them excel in coding competitions but also strengthen their career prospects. The first step towards

solving any competitive programming problem is always to read the problem statement carefully. It is not until the very end that one implements the code to solve the problem. We wanted to make sure that our course can train the students well enough so that they can efficiently grasp the abstraction of the problem and design an algorithm for it effectively without just jumping right into coding with no direction rather just trial and error. Furthermore, to ensure they can implement their algorithm into code effectively within the restricted time frame and debug it, we needed assessments that catered to practicing.

For this purpose, the lecture component of the course was divided into two parts. There were two lectures assigned for the course for every week of the semester. The first lecture comprised active learning using worksheets. These worksheets were designed for students to understand the problems fully and come up with the most effective algorithm paradigm to cater to the problem. A worksheet was then designed for every week dealing with a specific topic from the list of the course concepts identified previously. Every worksheet had around 3-4 competitive programming problems from different competitive platforms available online. These problems were carefully selected from SPOJ, POJ, UVA, Kattis, Codeforces, CodeChef, and ACM ICPC based on the topic they were dealing with. The problems tested the students on predicting the output of some test cases, handling the constraints, developing their own test cases, and then providing descriptions of the algorithms they were able to devise.

Now, the course required some class-based practice to see if the algorithms students are coming up with work in the long run, and that too in a short time frame. To cater to this need, the second lecture of the week was designed to keep mini-contests. These contests would be a smaller version of the actual coding competitions. They would not only provide efficient practice for the students but also familiarize them with the environment of a coding contest and its daunting yet thrilling aspect of competition and its time frame. The problems would be carefully selected from the platform of CodeForces which was selected to hold competitions. The first mini-contest would be only to familiarize the students with the platform and how to go about it whereas the real contests would start from the second week of the course. In this way, not only will the students be able to get used to the user interface of the platform but can now be assessed on their learnings the week after they have been introduced to the topics.

To foster continuous learning beyond classroom hours, a good academic strategy is needed to provide students with relevant practice material. In line with this objective, we designed a structured system of weekly problem sets, aligned with the topic covered in the week. The problem sets serve as practice material for students to further improve their skills and apply the concepts learned during the week. Students will be granted an entire week to solve the problems up until the release of the next problem set the following week. The problem sets were carefully designed from the problems database on Open Kattis, synchronized with the week's content, and the topic the problem caters to at varying difficulty levels. Therefore, each set encompasses a range of problems (ranging from 7-10 problems) that challenge students at varying levels of difficulty; easy, easy-medium, medium, medium-hard, and hard levels. This diverse problem distribution ensures that students engage with a variety of problems each with their own strategy and difficulty.

Furthermore, students will also engage in the activity of problem assessment where they will develop their very own competitive programming problems; this assessment will reflect their learning and understanding of a competitive programming problem. It will also allow them to test their ideas and play with creativity, enhancing their learning experience. The problem will then be assessed on the basis of the problem's interestingness writing quality, topics it targets, its difficulty level, etc. This can also serve as a mini project the students get to work on since the nature of the course does not allow broader project-based learning.

A competitive programming course generally holds assessments that are designed to gauge

students understanding and proficiency in solving algorithmic problems and coding challenges efficiently. The assessments of our course are designed in such a way that they not only comprehensively evaluate the student's algorithmic problem-solving and coding skills but also provide them with the opportunity for sufficient practice. Our course also provides students with innovative design challenges that will allow them to create their own problem from scratch! Furthermore, we have also kept bonus points that will not only encourage them to improve their coding skills but also give them rewarding grade points.

Our course assessments are divided as follows:

- Problem-Solving Assessments: A weekly problem set has been created catering to the topic of the week.
- In-class Worksheets: A weekly worksheet is created to solve in the first lecture of the week.
- Mini Contests: The mini contests are created on codeforces based on the topic taught in the previous week.
- Midterm and Final Contest: These will serve as competitions including problems from various topics.
- Problem Design Assessment: Students will design their own problems in this activity which will then be tested for interestingness, writing quality, and the topics it targets.
- External Contest Participation Points: Codeforces is known for holding all kinds of coding competitions, students can not only participate in them to improve their skills through practice but also earn points!
- Codeforces Rank Bonus Points: Codeforces gives rankings to its users based on how many problems they have solved; this does not only help them in their external participation but also earns them bonus points!

Based on these assessments we assigned grading weightage to every assessment.

Input	Output
Problem Sets (best 15)	30 %
In-class contest	20 %
Midterm Contest	15 %
Final Contest	20 %
Problem Setting	10 %
External Contest Participation	05%
Bonus: Codeforces Rank	

Table 2: Course Grading Policy

4.5 What are some ways to measure the effectiveness of a course that can be utilized for our course?

Among the diverse techniques outlined in the literature review, the university currently employs student evaluation surveys for every course. While incorporating Edumetric tests and alternative approaches could potentially enhance our course evaluation methods, their immediate integration hasn't been deliberated upon, primarily due to the project's current scope. Nevertheless, the incorporation of these methods could be taken into consideration in future iterations of the course with further work and research to assess their efficacy and adaptability.

5 Conclusion

In conclusion, the significance of competitive programming as a growing global trend within the realm of computer science education has been underscored. This has seen increasing participation from both professionals and amateurs, with programming competitions challenging participants' algorithmic prowess and coding abilities under time constraints. The prominence of such competitions is evident in their adoption by prominent technology companies for employee assessment.

This research endeavors to address an imperative within Habib University's curriculum by proposing the creation of a competitive programming elective course tailored to students with a foundation in programming, data structures, algorithms, and programming languages. The envisioned course aims to bolster students' algorithmic problem-solving capacities, foster collaborative teamwork, and enable participation in programming contests. By immersing students in practical programming challenges, the course intends to serve as a channel for applying theoretical concepts from abstract computer science domains, thus rendering it a platform for innovation and creativity.

The formulation of this course (syllabus attached in appendix) is guided by key inquiries encompassing the course's essential topics, pedagogical approaches to encourage active participation, platforms for online coding contests and problem repositories, the integration of assessments, and measures for gauging the course's efficacy. Recognizing the unique constraints of the university environment, where limited resources and diverse student backgrounds prevail, the course design has been meticulously crafted to ensure viability and effectiveness within these parameters.

Below we list and analyze the strengths, limitations, and probable future directions for our research:

5.1 Strengths:

1. Effective Assessment Tools: After extensive research and rigorous testing, we pinpointed two optimal platforms for course assessments. This enhancement ensures assessments are not only more effective but also comprehensible for students.

2. Pedagogical Approach: The adoption of the flipped classroom model empowered students with increased in-class practice opportunities, facilitated by mini contests and worksheets.

3. Course Design: Our course design process entailed meticulous analysis of a spectrum of competitive programming courses. This enabled us to discern fundamental concepts and topics for seamless integration.

4. Problem Curation: Algorithmic challenges were systematically curated from a diverse range of leading online platforms and judges. This strategic approach not only offered students diversity but also streamlined problem access, eliminating the need to navigate multiple platforms. Furthermore, our problem sets were thoughtfully designed to encompass a wide array of challenges across various course topics.

5.2 Limitations:

1. Case Studies: Although we accessed numerous online competitive programming courses, our research predominantly relied on the resources publicly available through these courses. At times, comprehensive insights into their strategies were elusive due to limited resource visibility.

2. Scarce Research on Effectiveness: The absence of substantial research evaluating the effectiveness of competitive programming courses prompted a shift in our focus. We redirected our efforts to explore broader methods of assessing course effectiveness.

3. Course Evaluation Constraints: To gauge the course's effectiveness, ideally, data from at least one course iteration should be accessible, correlating with students' overall performance in the course. Furthermore, there was no pilot testing done either.

4. Contests and Participation: Initially, the course aimed to facilitate external student participation in contests. However, logistical challenges and timing conflicts necessitated shifting contests to class hours, thereby restricting opportunities for wider student engagement.

5.3 Future Directions:

1. Feedback and Iteration: We can gather feedback from students and instructors who participate in the designed course and use this feedback to iterate and refine the course design based on real-world experiences in future offerings. Further, Edumetric tests may be designed and tested to analyze their feasibility for our course.

2. Comparative Analysis: We can compare the outcomes of the designed course with traditional programming courses or other innovative pedagogical approaches by making a course offering and studying students' progress and performance. This could help identify the strengths and weaknesses of the competitive programming course in relation to different teaching methods.

3. Skill Transfer: We can investigate how the skills developed in the competitive programming course translate into employability, technical interview success, and career advancement in fields beyond competitive programming, such as software development or data science. Furthermore, we can also study and research the soft skills students may acquire through the course.

4. Longitudinal Research: We can conduct longitudinal studies to assess the long-term impact of the designed course on students' performance in competitive programming, retention of skills, and real-world application of problem-solving abilities.

5. Teaching Strategy Experimentation: We can experiment with different strategies for teaching the course, for example implementing blended lab and theory sessions. Notably, educators from K. J. Somaiya College of Engineering propose intriguing teaching methods that could be considered for investigation [47].

In conclusion, this research aspires to advance not only the computer science education at Habib University but also the broader discourse on competitive programming pedagogy. By promoting hands-on learning, collaborative problem-solving, and practical skill development, the proposed course seeks to empower students to excel in real-world programming scenarios, contribute to industry demands, and embark on further academic pursuits with a competitive edge.

6 References

[1] 10Pearls. 10Pearls University holds WTQ 2022. url: https://10pearls.com/10pearlsuniversity-holds-wtq-2022/.

[2] 10Pearls. 10Pearls University Hosts CodeFest Competition. url: https://10pearls.com/10pearls-university-holds-codefest-2017/.

[3] Open AI. ChatGPT. url: https://chat.openai.com/chat.

[4] Mattox Beckman. Mattox Beckman. url: https://mattox.netlify.app.

[5] CodeChef. url: https://www.codechef.com.

[6] Codeforces. url: https://codeforces.com.

[7] Google. Google's Coding Competitions. url: https://codingcompetitions.withgoogle.com.

[8] Bjarki Ag´ust Guomundsson. ´ Bjarki Ag´ust Guomundsson ´. url: https://algo.is.

[9] Bjarki Ag´ust Guomundsson. ´ T-414-AFLV: A Competitive Programming Course ´. url: https://algo.is/competitive-programming-course/.

[10] Habib Univeristy, Facebook. 2021. url: https://www.facebook.com/HabibUniversity/posts/4210743022314945.

[11] HackerEarth. Amazon - Events — Top tech companies of 2023. url: https://www.hackerearth.com/companies/amazon/events/.

[12] HackerRank. url: https://www.hackerrank.com.

[13] Steven Halim. World of Seven - Steven Halim's Personal Website. url: https://www.comp.nus.edu.sg/~stevenha/.

[14] Greg Hamerly. Greg Hamerly. url: https://cs.baylor.edu/~hamerly/.

[15] ICPC Pakistan, Facebook. 2022. url: https://www.facebook.com/permalink.php?story_fbid=297527922485884&id=141825621389449.

[16] University of Illinois Urbana-Champaign. CS 491 Competitive Algorithmic Programming. url: https://courses.engr.illinois.edu/cs491cap/fa2019/.

[17] International Mathematical Olympiad. url: https://www.imo-official.org.

[18] International Olympiad in Informatics. url: https://ioinformatics.org.

[19] LeetCode. url: https://leetcode.com.

[20] Ninghui Li. Ninghui Li's homepage. url: https://www.cs.purdue.edu/homes/ninghui/.

[21] Jay Lim. Getting into Competitive Programming. url: https://imjching.com/writings/2019/06/21/getting-into-competitive-programming/.

[22] G.L. McDowell. Cracking the Coding Interview: 150 Programming Interview Questions and Solutions. CareerCup, LLC, 2011. isbn: 9781466208681. url: https://books.google.com.pk/books?id=anhAXwAACAAJ.

[23] Meta. Meta Coding Competitions. url: https://www.facebook.com/codingcompetitions.

[24] Jaehyun Park. Jaehyun Park. url: https://jaehyunp.github.io/profile/.

[25] National Univeristy of Singapore. CS3233 - Competitive Programming. url: https://www.comp.nus.edu.sg/~stevenha/cs3233.html.

[26] The Hitchhiker's Guide to the Programming Contests. url: https://comscigate.com/Books/contests/icpc.pdf.

[27] The ICPC International Collegiate Programming Contest. url: https://icpc.global.

[28] Topcoder. url: https://www.topcoder.com.

[29] Baylor University. CSI 4144: Competitive Learning, Spring 2022. url: https://cs.baylor.edu/~hamerly/courses/4144_22s/.

[30] Carnegie Mellon University. 15-295: Competition Programming and Problem Solving. url: https://contest.cs.cmu.edu/295/.

[31] Purdue University. Competitive Programmers Union. url: https://boilerlink.purdue.edu/organization/cpu.

[32] Purdue University. Competitive Programming Courses at Purdue Computer Science. url: https://www.cs.purdue.edu/homes/ninghui/courses/cpx_index.html.

[33] Stanford University. CS 97SI: Introduction to Programming Contests. url: https://web.stanford.edu/class/cs97si/.

[34] Stony Brook University. CSE 392 - Programming Challenges. url: https://www.cs.stonybrook.edu/about-us.

[35] Texas A&M University. Problem Solving Programming Strategies – CSCE430/2021Spring. url: https://tamu.kattis.com/courses/CSCE430/2021Spring.

[36] UT Austin. CS104c: Competitive Programming. url: https://www.cs.utexas.edu/users/downing/cs104c/.

[37] Etienne Vouga. Etienne Vouga - UT Austin Graphics - Home. url: https://www.cs.utexas.edu/users/evouga/.

[38] Wikipedia contributors. Competitive programming — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Competitive_programming.2022.

[39] YouTube. COMP300E - Programming Challenges - 2009 HKUST. url: https://www.youtube.com/playlist?list=PL07B3F10B48592010.

[40] Flipped classrooms Derek Bok Center, Harvard University. Available at: https://bokcenter.harvard.edu/flipped-classrooms.

[41] Brame, C. (2013). Flipping the classroom. Vanderbilt University Center for Teaching. Available at: http://cft.vanderbilt.edu/guides-sub-pages/flipping-the-classroom/.

[42] Crouch CH and Mazur E (2001). Peer instruction: Ten years of experience and results. *American Journal of Physics* 69: 970-977.

[43] Anderson LW and Krathwohl D (2001). A taxonomy for learning, teaching, and assessing: a revision of Bloom's taxonomy of educational objectives. New York: Longman.

[44] *2022-Spring-CSE109-introduction to programming contests* (2022) Jingbo Shang. Available at: https://shangjingbo1226.github.io/teaching/2022-spring-CSE109 (2022, March 29).

[45] Beckman, M. Getting started (live sections), Welcome to CS 491 CAP! Available at: https://courses.engr.illinois.edu/cs491cap/fa2019/.

[46] E. Enström, G. Kreitz, F. Niemelä, P. Söderman and V. Kann, "Five years with Kattis — Using an automated assessment system in teaching," 2011 Frontiers in Education Conference (FIE), Rapid City, SD, USA, 2011, pp. T3J-1-T3J-6, doi: 10.1109/FIE.2011.6142931.

[47] Sakhapara, A. and Pawade, D. (2018) 'Novel teaching strategies for lab-centric courses: Case Study of Programming course', 2018 IEEE Tenth International Conference on Technology for Education (T4E) [Preprint]. doi:10.1109/t4e.2018.00038.

[48] Akahori, Kanji. "Revised Design-Based Research Methodology for College Course Improvement and Application to Education Courses in Japan." Educational Technology, vol. 51, no. 6, 2011, pp. 26–33. JSTOR, http://www.jstor.org/stable/44429968.

[49] Francis S. Murphy (2005) Quality management: an 'essential attributes' approach. A case study towards a sustainable model of course effectiveness evaluation, Research in Post-Compulsory Education, 10:2, 227-244, DOI: 10.1080/13596740500200203

[50] Guannan Wang & Aimee Williamson (2022) Course evaluation scores: valid measures for teaching effectiveness or rewards for lenient grading?, Teaching in Higher Education, 27:3, 297-318, DOI: 10.1080/13562517.2020.1722992

[51] James W. Marlin Jr. & James F. Niss (1980) End-of-Course Evaluations as Indicators of Student Learning and Instructor Effectiveness, The Journal of Economic Education, 11:2, 16-27, DOI: 10.1080/00220485.1980.10844950

[52] Gerald M. Henson (1978) A Method to Evaluate Teaching Effectiveness in an Introductory American Government Course, Teaching Political Science, 5:2, 155-167, DOI: 10.1080/00922013.1978.11000117

[53] Wambsganss, T. et al. (2022) 'Designing conversational evaluation tools', Proceedings of the ACM on Human-Computer Interaction, 6(CSCW2), pp. 1–27. doi:10.1145/3555619.

[54] Halim, S. and Halim, F. Competitive Programming in National University of Singapore, available at: http://citeseerx.ist.psu.edu/documentrepid=rep1&type=pdf&doi=3b5e09c5da225e120e4a3b22a6df6c14ef474a08

[55] ICPC. ICPC Curriculum https://u.icpc.global/curriculum/

[56] IOI. IOI Syllabus https://ioinformatics.org/files/ioi-syllabus-2023.pdf

7 Reflections

Iqra Ahmed

Working on this research project alongside Areesha Amir and Ali Muhammad Asad under the supervision of Dr. Waqar Saleem has been an insightful experience. As first-time researchers, every one of us was faced with challenges and learned a great deal about what the research process actually entails. For me, this ignites a greater appreciation for all the academic work out there and the effort and time that goes into every single publication.

During our research, I was tasked primarily with reviewing relevant literature for course design and then subsequently determining the efficacy of our course. This process for me was initially very daunting and time-consuming. This was especially due to the lack of topic-specific papers, which meant that we needed to review papers from a broader spectrum of topics that hinted even slightly toward competitive programming. The limited amount of extra information that I was able to derive after long reading periods, perhaps added to the frustration. However, as I read more papers, I was able to sense patterns which allowed me to go over them a little faster. Some papers I was able to simply put aside because a glimpse at the abstract reflected a resemblance with a previously read paper. This thorough reading process indirectly helped me understand more properly how academic papers are written, and the extent to which the abstracts are relevant and helpful in understanding the body of the text. This understanding will be useful when reading more papers in the future or working on a project/research similar to this.

Further, participating in the discussions held within our meetings, such as those centered around evaluating the advantages and disadvantages of various platforms and the allocation of grade weightages, enabled us to exercise and improve our analytical capabilities. These discussions provided an opportunity to delve deeply into the nuances of decision-making, critically assessing the pros and cons of each option, and honing the skill of identifying optimal solutions based on multiple considerations. Moreover, the collaborative nature of our team dynamic served as a fertile ground for the enhancement of my collaborative capabilities. Working alongside my team members exposed me to diverse perspectives and methodologies, necessitating effective communication, adaptability, and a willingness to synthesize ideas cohesively. No doubt, the project also demanded careful time management to meet the various milestones and deadlines effectively. Through this, the experience reinforced the value of setting priorities, devising efficient schedules, and consistently adhering to them.

All in all, I believe it was 10 weeks well spent, with learning that extends well beyond the confines of a single project.

Areesha Amir

The idea of course development on a competitive programming course from scratch felt daunting initially, but the research journey quickly sparked innovative ideas and strategies. Being a student, allowed me to empathize with how certain teaching strategies and assessments are perceived in programming-heavy courses.

This research was particularly noteworthy since it marked the first time a competitive programming course was tailored for Habib University undergraduates. As expected, this research was not without its challenges. The meticulous selection of each design element for the course required extensive effort, and the research that underpinned the design drew inspiration from a wide array of 16+ courses spanning the globe.

Incorporating the flipped classroom model called for the creation of 15 meticulously designed worksheets and their corresponding solutions, each aligned with the weekly topic. This task proved the most formidable; devising pertinent test cases and ensuring problem relevance within the worksheet framework posed significant challenges. I ingeniously added unique twists to some problems to deter students from readily locating solutions online.

The most engaging facet of this research was crafting a course that aligned optimally with students' needs, leveraging insights from my own educational journey. Collaborating with Dr. Waqar Saleem was immensely enlightening. His input and ideas provided invaluable guidance and played a significant role in my understanding of course design.

This experience amalgamated innovation, growth through challenges, and an inherent sense of enjoyment. This research not only equipped me with practical skills in curriculum development but also enriched my perspective on effective pedagogy.

Ali Muhammad Asad

At first, I was skeptical of taking up this research since I was looking forward to a more CS practical and technical research project instead of a pedagogical one since that aligned with my interests more. However, taking up this research and being involved in designing an advanced-level course from scratch in 10 weeks only was a much better experience than I had anticipated. I got to be involved in the thought process that goes behind designing a course from scratch arising from a problem or need, formulating a rough plan to follow through while keeping the different stages in mind, researching existing courses and literature regarding the course, and then tackling the different problems arising in between such as platforms being used, platform availabilities, books used, etc, and then designing problem sets (and also solving them along the way), it was all a good learning experience. There were some stages where I was a little stuck especially in the initial stages while finding relevant competitive programming courses as that was tough to find and filter courses that had relevant material and content while being available, however, I found that the research proposal document provided by our supervisor was more than helpful as a starting point for the research. Moreover, Bard by Google was more beneficial at providing links to such courses. Apart from that, there weren't many tough or challenging points during the research, as the found literature and material during the initial stages, and the weekly meetings with our supervisor were more than enough to provide us with a clear outline. In addition, all three members worked on our parts diligently which enabled smooth progress. A good hidden outcome of this was that one of my parts was to go over the different languages and online coding platforms used, and then develop problem sets using those problem archives for our own course. In doing so, I was able to solve about 50% - 60% of those problems which provided me with a much deeper understanding of the type of problems encountered in competitive programming competitions, and also kept me in practice with the languages I knew myself. Also, I was able to learn more about those languages as well than I previously knew, indirectly preparing me for coding and CS-related job interview questions as well. So all in all, it was a really great learning experience, and although initially I was not interested in pedagogy, now it is definitely something that I can consider pursuing.

A Appendix

Habib University
shaping futures

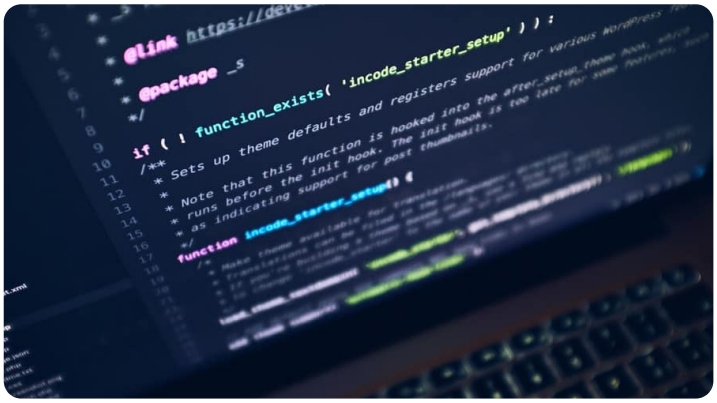
Competitive Programming

CS 3XX1 WS
Blueprint Term – Fall 2023

"First solve the problem, then write the code." - John Johnson

"Confusion is part of programming." - Felienne Hermans

"You might not think that programmers are artists, but programming is an extremely creative profession. It's logic-based creativity." - John Romero



Course Information

Course Prerequisites: None
Hardware/Software Prerequisites (if any): a LaTeX compiler, a recent C++ compiler, or any code editor with at least two languages, a computer with a mic, camera, and internet connection.
Content Area: This course fulfills the requirements of a CS Elective or a Free elective.

Instructor Information

Instructor: Waqar Saleem
Office Location: C-122
Email: waqar.saleem@sse.habib.edu.pk
Office Hours: TBD

Instructor: Ali Muhammad Asad
Office Location: NA
Email: aa07190@st.habib.edu.pk
Office Hours: NA

Instructor: Areesha Amir
Office Location: NA
Email: aa07613@st.habib.edu.pk
Office Hours: NA

Instructor: Iqra Ahmed
Office Location: NA
Email: ia07674@st.habib.edu.pk
Office Hours: NA

Course Description

Programming competitions have become increasingly popular over the years, with many participants, both professional and amateurs, competing for recognition and prizes. Competitive programming has become a global phenomenon, with contests and competitions held regularly in various countries. These competitions involve solving algorithmic problems within a specified amount of time, typically a few hours, designed to test the participants' problem-solving and programming skills under time pressure.

Competitive Programming provides a fun setting for algorithmic problem-solving and computer programming. Often conducted as a team sport, it is a valuable tool for improving programming skills, particularly among undergraduate students in computer science programs. Therefore, the skills required for competitive programming align well with the learning objectives of a computer science program.

Competing requires a high level of knowledge of efficient input/output techniques, libraries of different languages, data structures, efficient and optimized algorithms, and problem-solving and critical

thinking skills, which are fundamental to a computer science education, and will be explored in this course and much more.

Course Aims

This course aims to empower students with existing exposure to competitive programming, data structures, algorithms, and programming languages with a more comprehensive skill set by immersing students in the world of competitive programming. Students will have an opportunity to deepen their understanding of the fundamentals, data structures, and algorithms and enhance their problem-solving skills, collaboration with peers, and ability to come up with efficient and optimized solutions for problems under time constraints and participate in programming contests, thereby preparing them for real-world programming challenges.

In addition, it will also allow students a creative outlet for their studies in abstract computer science topics like data structures and algorithms, computational geometry, and geometric problems, graph theory, and network flow analysis. Through inclusive activities, it will indirectly serve to popularize competitive programming on campus and prepare teams for the International Collegiate Programming Contes (ICPC), which will in turn make our students more attractive to industry and graduate schools.

Course Learning Outcomes (CLOs)

CLO	Outcome / Description	Learning-Domain-Level
CLO 1: Problem Solving	Dissect complex algorithmic problems, employing systematic problem-solving techniques to devise efficient and optimized solutions	Cog-4
CLO 2: Algorithmic Proficiency	Deep understanding of algorithmic concepts, recognizing appropriate algorithms and data structures for various problem scenarios, and employing them effectively	Cog-2, Cog-3, Cog-5
CLO 3: Efficient Code Implementation	Translate algorithmic solutions into clean, concise, and efficient code	Cog-6
CLO 4: Time-Pressured Development	Devise creative solutions under time constraints such as in a competitive environment	Cog-5

CLO	Outcome / Description	Learning-Domain-Level
CLO 5: Collaboration	Fruitfully collaborate with their team on the creation and development of solutions to problems	Aff-3

Format and Procedures

This is a 300-level course of 3 credit hours, requiring a good grasp over at least two programming languages: C++ and Python are preferred, however, Java, Javascript, and any other popular programming languages are also acceptable. The rule of thumb for out-of-class time for a course is at least 2 hours of work outside class for every credit hour. Attention in lectures is imperative, and all assignments must be done in a timely manner.

We will use several online platforms:

- Canvas: Habib University's LMS and our course page, on it is your one-stop-shop for all official course information.
- Live Syllabus(if made): This is an up-to-date version of the course schedule as the semester proceeds.
- GitHub: *must add something over here if needed else remove please**
- Open Kattis: problem sets will be created and attempted over Kattis.
- Codeforces: mini-contests, mids, finals, and all such competitions will be taken over Codeforces.

Expectations:

So that you succeed in this course, we expect that you follow the good academic practices listed below:

- You will become familiar with the *online platforms* used in this course.
- You will *check your email regularly* and stay abreast of course communication.
- You will *read the book* and stay abreast of it as the course proceeds.
- You will be *fully present* in the class. That is, both physically and mentally. Please give your full attention to the class and participate actively.
- You will *take responsibility* for your learning:
    - *Seek help* when you need it.
    - *Be honest* about your work.

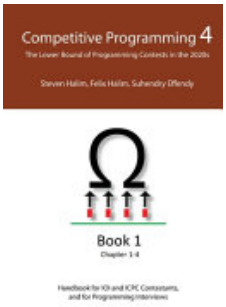
    - *Complete* the assignments and all relevant work in a timely manner.
    - Be a *good team player*. A good part of this course will be in teams. Ensure that teamwork is a good learning experience for everyone.
- Concerns regarding a score can be reported up to a week after its release. Concerns raised later cannot be entertained.
- You will maintain behavior in class that *befits* Yohsin and acknowledges the classroom as a place of learning, exploration, and experimentation.

Mode of Instruction

Instruction in higher education all over the world has vacillated unpredictably between in-person and online for more than two years, leading to at least one important lesson. We do not like remote, online learning or instruction. To every extent possible, this course will take place in person. We will meet twice a week for 75-minute lecture sessions.

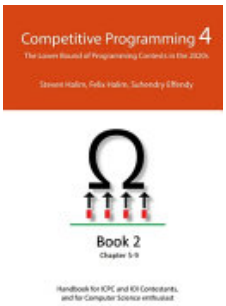
In the unfortunate circumstance where we need to go online, relevant instructions will be shared accordingly. For that contingency, you should have a computer with an internet connection that is capable of running a latest browser version and Zoom.

Required Texts and Materials



Competitive Programming 4 - Book 1
ISBN: 9781716745522
Authors: Steven Halim, Felix Halim, Suhendry Effendy
Publication Date: 2018-12-03

Competitive Programming 4 - Book 2
ISBN: 9781716745515
Authors: Steven Halim, Felix Halim, Suhendry Effendy
Publication Date: 2020-07-18



Assessments

Assessment Type	(%)	Remarks
Problem Sets (best _)	30	These will be attempted individually on Open Kattis. Problem Sets will be released every week.
Problem Setting	10	By the course end, each student will have to individually submit an original problem made by them.
External Contest Participation	05	Each student will be rewarded points for participating in contests externally. This can be individual or in teams.
In-Class Contests	20	These will be done in teams of three at most over Codeforces, simulating a competitive programming environment.
Midterm Contest	15	The midterm will be a competitive programming contest done in teams of at most three.
Final Contest	20	The final will be a competitive programming contest done in teams of at most three.

To keep up with the lectures and to provide adequate practice, problem sets will be released every week containing problems on material covered in the same week. Students will have until the release of the next problem set to submit their solutions.

Grading Scale

Letter Grade	GPA Points	Percentage
A+	4.00	[95-100]
A	4.00	[90-95)
A-	3.67	[85-90)
B+	3.33	[80-85)
B	3.00	[75-80)

Letter Grade	GPA Points	Percentage
B-	2.67	[70-75)
C+	2.33	[67-70)
C	2.00	[63-67)
C-	1.67	[60-63)
F	0.00	[0, 60]

Note: [a, b) is a range of numbers from a to b where a is included in the range and b is not.

Late Submission Policy

Please observe the deadline prescribed for each assessment. There is no late submission policy. It is better to submit partial work on time and receive partial credit than to submit complete work late and receive no credit. In order to avoid last minute emergencies, e.g. power failure close to the deadline, start your work early and aim to finish it in advance of the deadline. With special approval from your instructor, you may submit late within the faculty's assigned deadline with a 20% late submission penalty. Please be mindful of deadlines and discuss with your instructor beforehand if you foresee any issues.

[*And any other information as per instructor's need]

Week-Wise Schedule (Tentative)

The schedule may change in view of class progress as the semester proceeds. All indicated chapters under Reading are from the course textbook. See the **Live Syllabus** for an updated version. [*Please upload live syllabus]

Week	Description / Topic	Readings	Assessments and Due Date
Week - 1	Introduction: Input/Output Techniques, Ad Hoc Simulation	Chap 1, 5.2, 6.2, and Chapter 9	Problem Set 1
Week - 2	Elementary Data Structures, Libraries in Python and C++	Chap 2 till 2.2, 8.7.2	Problem Set 2
Week - 3	Data Structures and Sub-linear complexity data structures	Chap 2 complete, 9.3	Problem Set 3
Week - 4	Searching and Sorting, Problem-Solving Paradigm: Divide and Conquer	Chap 3 till 3.3, 4.2.6, 8.2, 9.2	Problem Set 4

Week	Description / Topic	Readings	Assessments and Due Date
Week - 5	Greedy Algorithms	Chap 3 complete, 8 with focus on 8.3, 9	Problem Set 5
Week - 6	Dynamic Programming	Chap 3 complete, 8 with focus on 8.3, 9	Problem Set 6
Week - 7	Midterm Week - Midterm Contest	All readings yet	-
Week - 8	Graphs, Graph Traversal, and algorithms including BFS and DFS	Chap 4, focus on 4.2, 8.2.1, 8.2.2, 8.5, 8.7.4, 8.7.6	Problem Set 7
Week - 9	Intermediate Graph Algorithms and Trees	Chap 4, focus on 4.2, 4.3, 4.6 and 4.7, 8.2.2, 8.5, 8.6	Problem Set 8
Week - 10	Shortest Path Algorithms	Chap 4, focus on 4.4 and 4.5, 8.6	Problem Set 9
Week - 11	Network Flow Problems	Chap 8, focus on 8.4, 9.25	Problem Set 10
Week - 12	Computational Geometry and Geometry Algorithms	Chap 7, 8.7, 9.18, 9.19	Problem Set 11
Week - 13	Strings, matching, Suffix Tree, Prefix Tree, Tries	Chap 6	Problem Set 12
Week - 14	Mathematics and Number Theory	Chap 5, focus on 5.2 and 5.3, relevant problems from chap 9	Problem Set 13
Week - 15	Combinatorics	Chap 5, focus on remaining topics: 5.4 onwards, chap 9 problems	Problem Set 14
Week – 16			
Some dates	Reading Days	-	
Some dates	Final Examination Days§	All readings yet	

Attendance Policy

You are expected to attend and participate in all lectures and contests. Under extenuating circumstances, you may miss up to 04 synchronous sessions. In case of a missed session, you must inform your instructor of the reason. Failing to do so may raise an early academic alert with the Office of Academic Performance (OAP). Excessive absences will lead to an automatic withdrawal from the course.

Final Exam Policy

The Final Exam is going to be a 3-hour long programming contest in teams of at most 3 students. The contest is going to be held on Codeforces and will simulate a competitive programming environment, following all formats and practices of the International Collegiate Programming Contest (ICPC).

Academic Integrity

Each student in this course is expected to abide by the Habib University Student Honor Code of Academic Integrity. Any work submitted by a student in this course for academic credit will be the student's own work.

Scholastic dishonesty shall be considered a serious violation of these rules and regulations and is subject to strict disciplinary action as prescribed by Habib University regulations and policies. Scholastic dishonesty includes, but is not limited to, cheating on exams, plagiarism on assignments, and collusion.

a. Plagiarism: Plagiarism is the act of taking the work created by another person or entity and presenting it as one’s own for the purpose of personal gain or of obtaining academic credit. As per University policy, plagiarism includes the submission of or incorporation of the work of others without acknowledging its provenance or giving due credit according to established academic practices. This includes the submission of material that has been appropriated,// bought, received as a gift, downloaded, or obtained by any other means. Students must not, unless they have been granted permission from all faculty members concerned, submit the same assignment or project for academic credit for different courses.
b. Cheating: The term cheating shall refer to the use of or obtaining of unauthorized information in order to obtain personal benefit or academic credit.
c. Collusion: Collusion is the act of providing unauthorized assistance to one or more person or of not taking the appropriate precautions against doing so.

All violations of academic integrity will also be immediately reported to the Student Conduct Office.

You are encouraged to study together and to discuss information and concepts covered in lecture and the sections with other students. You can give "consulting" help to or receive "consulting" help from

such students. However, this permissible cooperation should never involve one student having possession of a copy of all or part of work done by someone else, in the form of an e-mail, an e-mail attachment file, a diskette, or a hard copy.

Should copying occur, the student who copied work from another student and the student who gave material to be copied will both be in violation of the Student Code of Conduct.

If you wish to use generative-AI tools to complete any of your assessments, you must first obtain permission from your course instructor. AI generated work will not be accepted in all classes or even all assessments. The instructor's permission is required. If the permission is granted, you should declare its use and properly cite the source of the generated content. Failing to identify AI written or assisted work is academic dishonesty and will be treated as any case of plagiarism by the university.

The principle for academic integrity is that your submissions must be substantially your own work and that any work that is not originally your thought must be identified and credited. If the use of AI tools is prohibited in the course, respect the rules and do not use these tools for assessments. The fundamental purpose of assessment is to learn, synthesize information and explain new connections and interpretations that arise from your secondary research. Be aware that unauthorized use of AI tools for assessments can result in a conduct case being filed. This can have serious consequences for your academic standing and future career opportunities.

During examinations, you must do your own work. Talking or discussion is not permitted during the examinations, nor may you compare papers, copy from others, or collaborate in any way. Any collaborative behavior during the examinations will result in failure of the exam, and may lead to failure of the course and University disciplinary action.

Penalty for violation of this Code can also be extended to include failure of the course and University disciplinary action.

Program Learning Outcomes (For Administrative Review)

Upon graduation, students will have the following abilities:

- PLO 1: Analysis: Analyse a given situation and reduce it to one or more problems that can be solved via computer intervention.
- PLO 2: Design: Design one or more computer-based solutions of a given problem and select the solution that is best under the circumstances.
- PLO 3: Programming: Program a given solution in a variety of programming languages belonging to different paradigm.

- PLO 6: Self-learning: Research, learn, and apply requirements needed to implement a solution for a given high level problem description.

- PLO 8: Communication and Teamwork: Work effectively in inter-disciplinary teams.

Program Learning Outcomes (PLOs) mapped to Course Learning Outcomes (CLOs)					
	CLOs of the course are designed to cater following PLOs: PLO 1: Analysis PLO 2: Design PLO 3: Programming PLO 6: Self-Learning PLO 8: Communication and Teamwork				
	Distribution of CLO weightages for each PLO				
	CLO 1	CLO 2	CLO 3	CLO 4	CLO 5
PLO 1					
PLO 2					
PLO 3					
PLO 6					
PLO 8					

Mapping of Assessments to CLOs

Assignments	CLO #01	CLO #02	CLO #03	CLO #04	CLO #05
Problem Sets	X	X	X		
Problem Setting	X	X			
External Contest Participation	X	X	X	X	X
In-Class Contests	X	X	X	X	X
Midterm Contest	X	X	X	X	X
Final Contest	X	X	X	X	X

Recording Policy

Only asynchronous and synchronous online sessions will be recorded and uploaded on our Video Management System (Panopto). Link to the folder of recordings will be available to all students. Hyflex classes might be recorded if faculty deems it appropriate.

Accommodations for Students with Disabilities

In compliance with the Habib University policy and equal access laws, I am available to discuss appropriate academic accommodations that may be required for student with disabilities. Requests for academic accommodations are to be made during the first two weeks of the semester, except for unusual circumstances, so arrangements can be made. Students are encouraged to register with the Office of Academic Performance to verify their eligibility for appropriate accommodations.

Inclusivity Statement

We understand that our members represent a rich variety of backgrounds and perspectives. Habib University is committed to providing an atmosphere for learning that respects diversity. While working together to build this community we ask all members to:

- share their unique experiences, values and beliefs
- be open to the views of others
- honor the uniqueness of their colleagues
- appreciate the opportunity that we have to learn from each other in this community
- value each other's opinions and communicate in a respectful manner
- keep confidential discussions that the community has of a personal (or professional) nature
- use this opportunity together to discuss ways in which we can create an inclusive environment in this course and across the Habib community

Office Hours Policy

Every student enrolled in this course must meet individually with the course instructor during course office hours at least once during the semester. The first meeting should happen within the first five weeks of the semester but must occur before midterms. Any student who does not meet with the instructor may face a grade reduction or other penalties at the discretion of the instructor and will have an academic hold placed by the Registrar's Office.