From Problems to Performance: Two Decades of Programming Contests in the CCSC Southeast*

Andy D. Digh
Computer Science Department
Mercer University
Macon, GA 31207
digh_ad@mercer.edu

Abstract

This paper examines two decades of programming contests organized by the CCSC Southeast, analyzing recurring problem types and identifying key factors contributing to team success. Five primary problem categories are identified: string manipulation, mathematical computation, search and data structure application, graph algorithms, and problems with high implementation complexity despite simple algorithmic foundations. Analysis reveals Python is associated with higher success rates largely due to its efficiency in development and debugging. The paper also outlines key contest strategies, including teamwork, strategic problem selection, and time management, alongside effective preparation methodologies employed by top-performing teams.

1 INTRODUCTION

Competitive programming provides substantial pedagogical benefits for students [1]. This practice strengthens algorithmic understanding by requiring participants to apply and adapt established algorithms to new and challenging problems. Additionally, engagement in such contests develops crucial computer science skills, such as collaborative problem-solving, preparation for technical interviews, and the capacity for self-directed learning.

Analysis of the Consortium for Computing Sciences in Colleges Southeastern (CCSC SE) programming contest over the past two decades reveals significant, undocumented trends, best practices, and regional characteristics. No prior research exists specifically on CCSC programming competitions, beyond a 2001 panel on student preparation [2].

Since its inception at Furman University in 1994, the CCSC SE contest has seen substantial growth in team participation. This three-hour competition typically occurs on a Saturday morning in November, concluding with an awards ceremony alongside the conference closing. A voluntary 90-minute practice session, featuring two problems, is offered the preceding Friday evening to familiarize participants with the local computing environment, IDEs, and compilers.

Teams are limited to a maximum of four undergraduate members; however, to minimize congestion and noise within the computer laboratories, only two team members are permitted at a computer workstation concurrently, with remaining members utilizing a designated work area. This structure allows for collaborative problem-solving, as team members not at a workstation can develop solutions on paper and subsequently exchange places with a teammate at a computer.

The contest employs an automated, web-based contest management system (CMS) that evaluates submissions for correctness without human intervention. This system allows the judges to see the student source code at any time, as well as verify the student output on the secret judging input. While problem authors strive for clarity, natural language is inherently prone to multiple interpretations. To address any perceived ambiguities in the problem statements, students may submit clarification requests through the contest software. If deemed necessary, global clarifications are disseminated to all participants via the same system. However, the judging panel refrains from providing interpretive assistance for the problems themselves. Feedback provided to students upon submission includes standard competitive programming responses such as "correct," "compiler error," "run time error," "runtime limit exceeded," and "wrong answer." The runtime limit for judging a problem on the CMS is configurable, with a default setting of five seconds. Submissions exceeding this limit typically indicate either an infinite loop in the code or an algorithm that is not sufficiently optimized for the given problem constraints.

To heighten competitive tension, the scoreboard is disabled for the final 45 minutes of the competition. Teams are ranked primarily by the number of problems solved, with penalty points used to break any ties. These penalty points are incurred from two sources: incorrect submissions and the time taken to correctly solve problems. The awards ceremony then recognizes the top ten teams, presenting plaques to the top three, and also offers special acknowledg-

ments for the team that traveled the furthest, the team that solved the first problem, and the team achieving a "Hail Mary" (the last correct submission before time expired).

Since 2004, the CCSC SE programming contest has consistently offered six to ten problems, averaging nine per contest. Winning teams have typically solved an average of 7.5 problems. Faculty involvement is central to problem development; sponsors are encouraged to submit original problems, which are then reviewed by the contest director and a panel of judges. The contest director assumes responsibility for refining problem statements to eliminate ambiguities and for managing judging inputs. Other CCSC regions (Midwest, Eastern, Central Plains, Northeastern) host their own student programming contests with distinct protocols. For example, the 2023 Midwest contest featured five faculty-contributed problems over four hours. The idea of a national programming contest across all ten CCSC regions has even been suggested, and its realization would be an exciting step.

2 TYPES OF CONTEST PROBLEMS

CCSC SE contest problems offer participants a description of the task, a precise specification, and illustrative sample input/output examples. Figure 1, showcasing "Narcissistic Numbers" from the 2024 contest [3], demonstrates this standard problem structure. Solutions are automatically evaluated by the CMS system against a comprehensive set of undisclosed test cases, which are distinct from the samples to thoroughly assess code correctness and robustness.

Based on a retrospective analysis of the past two decades of Southeast programming contest problems, all available online, a discernible pattern of five recurring categories emerges. Nearly all annual problems can be systematically classified into these primary types. Problems from the first three categories were more frequently solved and appeared more often than those in the latter two. These five categories collectively cover foundational content in the curriculum for the first three courses of a typical computer science major.

String Manipulation: Problems within this category necessitate the algorithmic processing of string data. Common sub-tasks include lexical analysis (e.g., word counting), string substitution, and information extraction via parsing techniques. For example, the most solved problem in the 2014 set was entitled “Royal Tweet,” which required students to write a program to scan a given number of social media posts which mentioned the substring “@BritishMonarchy” [4].

Mathematical Computation: This class encompasses problems requiring the application of fundamental geometric principles (e.g., area and angle calculations for geometric figures), and concepts from calculus, discrete math-

Narcissistic Numbers

Write a program to determine if a given number is narcissistic. A narcissistic number is a number that is the sum of its own digits each raised to the power of the number of digits.

Input
The input will be a single integer c ($0 \leq c \leq 100$), the number of test cases. Then follow c integers c_i, $i = 1, 2, \cdots, c-1, c$, with each c_i, on a line by itself and $0 \leq c_i \leq 10^{15}$.

Output
For each value c_i, write on a single line the case number, and YES if c_i is a narcissistic number, NO otherwise. See sample below.

Sample Input
```
4
1
371
22
28116440335967
```

Output Corresponding to Sample Input
```
Case #1: YES
Case #2: YES
Case #3: NO
Case #4: YES
```

Figure 1: Number theory problem from 2024 contest, solved by 90% of teams.

ematics, and number theory. In 2019, students solved "Slicing Potatoes" which describes a potato geometrically represented as an ellipsoid. The problem required the calculation of the volume of each slice of this ellipsoid. This involved using integral calculus, specifically the method of disks or washers, to find the volume of the solid generated by revolving an ellipse about the x-axis, and then determining the volume of specific segments (slices) [5].

Search and Data Structure Application: These problems require the use of fundamental data structures, such as multi-dimensional arrays, sets, and dictionaries, for efficient information search and retrieval. For instance, "Pumpkinfest," the fourth most solved problem in 2018, involved searching a square matrix of any dimension representing a garden to find the largest pumpkin patch size (represented by `p` for pumpkin and `s` for squash) [6]. Figure 2 illustrates three pumpkin patches (sizes 1, 3, and 9) that were readily found using recursion.

s	s	s	s	s	p
s	p	s	s	s	s
s	p	s	s	p	p
s	p	s	s	s	p
s	p	s	s	s	s
s	p	p	p	p	p

Figure 2: 2018 "Pumpkinfest" search problem used a matrix for a garden.

Graph Algorithmic Problems: This category specifically involves problems solvable through the application of standard graph traversal algorithms, notably Dijkstra's algorithm, Depth-First Search (DFS), and Breadth-First Search (BFS). The "Time Loop Trap" problem, featured in the 2024 competition, served as a prime example of a problem leveraging graph theory [7]. This challenge required students to identify the presence of a cycle within a directed graph. DFS proved to be an efficient method for its rapid resolution. This problem was solved exclusively by the winning team.

Implementation-Intensive Problems: This class comprises problems where the selection of an appropriate algorithmic approach is straightforward; however, their implementation demands meticulous and time-consuming coding to ensure correctness on all the edge cases [8]. The "Olympic Dilemma" problem from 2023 asked contestants to write a program to help a weightlifter load a barbell [9]. Given a target weight and a set of available weight plates (each with a specific weight and quantity), their program had to determine the optimal combination of plates to use. While the underlying algorithmic technique (dynamic programming) is standard, the multiple, prioritized tie-breaking rules make the implementation tricky and prone to subtle bugs if not handled meticulously. Only the top two teams were able to solve this challenging problem.

A successful programming contest requires a well-rounded problem set that effectively engages participants. The creation of original problems is a demanding process, requiring significant time and careful execution. The objective is to ensure accessibility for all teams, allowing them to solve at least one problem, while simultaneously challenging the most proficient competitors. A suggested distribution for a nine-problem set is two easy, five medium, and two hard problems, which can be arranged in any sequence. The recent introduction of author attribution on problem statements has increased faculty contributions. Since all problems are reviewed by multiple individuals and widely disseminated, they could contribute to a faculty member's scholarship profile and creative work.

3 LANGUAGE CHOICE & TEAM SUCCESS

Optimal contest strategy allows teams to choose the most suitable programming language for each problem, rather than being restricted to one. Data from online contest results since 2004 suggests a correlation between language choice and team success: Python is associated with the most correctly solved problems, closely followed by Java. This higher success rate for Python teams likely stems from its strengths in rapid development and concise code which often outweigh its slower execution for a significant portion of competitive pro-

gramming problems.

Figure 3 illustrates a significant shift in programming language preference at the CCSC SE competitive programming contest over the past two decades. Java dominated from 2005 to 2017, but Python's adoption surged dramatically, surpassing all other languages by 2018. Its share of correct solutions soared from 18% in 2008 to 67% by 2018, continuing an upward trend to approximately 90% of all accepted correct solutions by 2021.

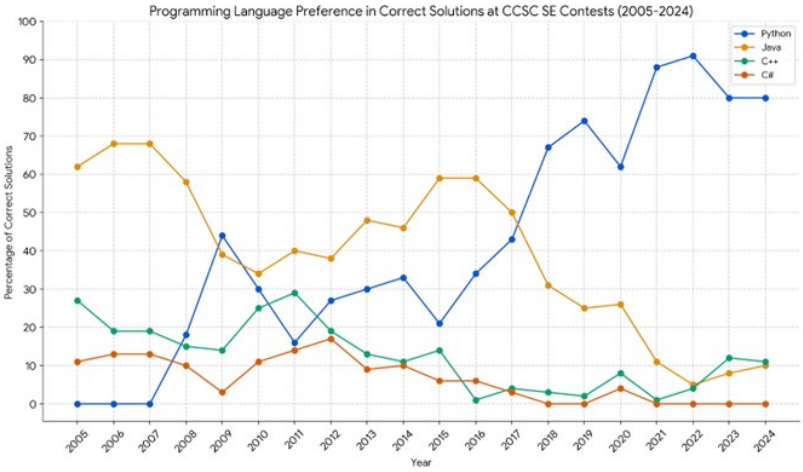

Figure 3: Analysis of success by programming language from 2005 to 2024.

Python's increased adoption in competitive programming, particularly evident since the 2008 release of Python 3.0, is largely attributable to its comprehensive standard library and extensive third-party packages. These resources, especially those optimized for string manipulation, significantly accelerate development time, making Python an attractive and efficient choice for contestants.

For problems where the optimal algorithm is found quickly and implemented cleanly, Python's efficiency in debugging can also lead to faster correct submissions and ultimately, greater success. It allows teams to spend less time on "syntactic overhead" and more time on "algorithmic thinking," which is often the core challenge of these contests. Additionally, Python can help "improve the efficiency of team performance as students would spend less time submitting incorrect programs and waiting for results" [10]. The consistent use of Python by the winning teams over the last six years at CCSC SE highlights a significant correlation between language selection and high-ranking performance in programming contests.

4 CONTEST STRATEGIES

The principle of *synergy* is critical in programming contests. This concept, expressed as "3 * 1 = 4" by Ernst et al., underscores how effective teamwork—encompassing communication, collaborative problem-solving, and time management—leads to superior outcomes compared to individual work [8]. Interviews with eight students from two teams competing in the 2024 CCSC SE contest revealed recurring strategic insights consistent with these principles when they were asked to reflect on lessons learned and contest performance.

- **Teamwork is crucial:** Collaboration among teammates was frequently cited as essential for overcoming obstacles and identifying solutions.
- **Problem selection matters:** Students mentioned that their team focused on "picking problems we felt best equipped to solve".
- **Fresh perspective is valuable:** Participants emphasized the benefit of a "fresh pair of eyes" in reviewing code, which aided in identifying previously overlooked errors [11].

Student interviews confirmed a common perception of competitive programming contests as an intensely fast-paced experience, with many describing the 180-minute duration as remarkably short. Analysis of qualitative data from interviews indicates that successful CCSC SE teams employ a distinct, four-step time management approach to optimize their performance.

1. **First 15 minutes:** When tackling problems, teams should first focus on identifying the easiest ones. Avoid the temptation to start with problems that seem most interesting.
2. **Divide and Conquer by Specialization:** Teams should begin by establishing a rough order of attack, with initial efforts concentrated on solving the easiest problems first. Many teams find success using a specialist approach, where problems are assigned to members based on expertise. While code style does not earn points, adopting good programming habits such as proper indentation and clear variable names is highly beneficial for preventing bugs and facilitating team debugging. In addition, it is very important to always have a teammate cross-reference your output with the problem statement before submission. This simple check can save you from frustrating presentation errors caused by small mistakes like a misplaced comma or period.
3. **Halfway Point Problem-Solving:** The focus shifts to all problems of medium difficulty. During this phase, it is beneficial to perform as

much work away from the computer as possible, utilizing the printer to review and debug solutions that received incorrect judgments. Looking at the scoreboard to see which problems have been solved can also help. Essential to this stage is the capacity to identify and resolve any ambiguities within problem statements, or to formulate sound assumptions. Any necessary clarifications should be submitted to judges.

4. **Final Hour:** The last hour is dedicated to problems that were attempted but were not successfully judged correct. For wrong answers, be sure you have tested all boundary conditions described in the problem input. Be wary of edge cases that involve zero or large limits. Teams should only move on to harder problems if all easy and medium-difficulty problems have been completed. Participants often face "tunnel vision"—the tendency to keep working on a single problem despite repeated incorrect submissions. To overcome this, it is crucial to learn when to hand the keyboard to a teammate or move on to a different problem and revisit the challenging one later with a different perspective [1].

5 PREPARING FOR COMPETITIVE CODING

Successful competitive programming teams often use a structured, multifaceted training approach. A common and effective strategy involves integrating a one-credit academic course specifically for programming team strategies. This elective course typically features weekly, scheduled sessions designed to foster collaborative problem-solving and refine competitive programming techniques. Graded on a satisfactory/unsatisfactory basis, such a course usually requires at least one semester of prior programming experience. Many universities recognize this structured team participation as a form of *experiential learning*, aligning with educational philosophies that prioritize "learning by doing" and reflective practice [2].

A holistic examination of the CCSC SE programming competition results spanning two decades indicates that multiple institutions have consistently excelled, largely due to their similar preparation strategies. Bob Jones University has demonstrated exceptional performance, earning the highest number of top-three finishes. Furman University and Mercer University have also shown significant and sustained success in the competition. The consistent high performance of these three institutions is largely due to a combination of factors. First, teams from these universities include a dedicated course as mentioned above. Second, these institutions strategically leverage the CCSC SE competition rules, which permit participants to utilize standard library APIs, archived problem solutions, and pre-selected reference materials. Com-

monly used resources include *The Algorithm Design Manual* [12], *Competitive Programming 4* [13], and *Programming Challenges: The Programming Contest Training Manual* [14]. Third, these programs cultivate a strong recruitment culture, often driven by enthusiastic current team members who are instrumental in generating interest and attracting new talent. This supportive environment is further reinforced by the engagement of alumni, who frequently return to practices to offer invaluable advice. This iterative process of shared knowledge and continuous engagement embodies the well-known proverb, "iron sharpens iron," fostering ongoing improvement and sustained participation.

Furthermore, these institutions implement comprehensive preparatory strategies, including frequent internal practice competitions held throughout the academic year. For example, as illustrated in Figure 4, Bob Jones University enhances its preparation through an official three-hour contest hosted on its campus each spring semester.

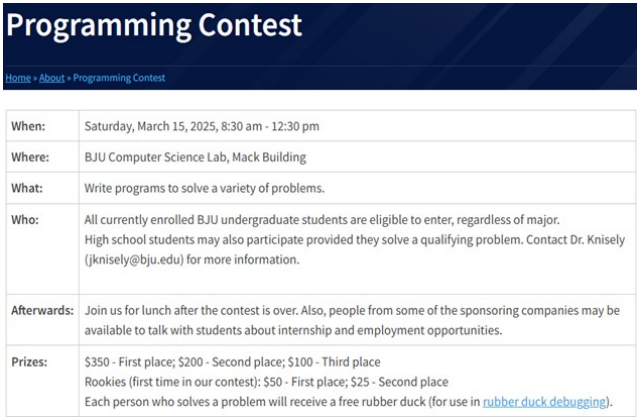

Figure 4: The Bob Jones University Spring Programming Contest.

More than just a competition, this campus event provides crucial competitive experience, along with monetary awards and rubber ducks. Strategically, it connects students with corporate sponsors for internships and employment, and attracts high school students to STEM programs. It is also an effective training ground for top-tier programming competitions, including The International Collegiate Programming Competition (ICPC) and Codeforces.

6 CONCLUSIONS AND FUTURE WORK

The CCSC SE contest has historically restricted participants to a single laboratory computer per team, strictly prohibiting external electronic devices such

as cell phones and personal laptops, which must remain powered off during the competition. This policy gained increased urgency in the fall of 2023 with the advent of advanced large language models capable of generating source code. During the conference held that year, contest judges conducted an experiment, submitting each of the nine problems to ChatGPT for Python solution generation. The AI-generated code successfully passed the secret input tests for three of the nine problems. While this outcome was informally noted as a "win for humanity" at the time, it is anticipated that AI performance in such tasks will significantly improve in the future.

Given the advancements in AI code-generation, modifying programming contest problem structures to resist automated solutions is crucial. Pawagi and Kumar (2024) propose "probeable problems" as an effective approach. This method involves intentionally underspecified problem statements, requiring contestants to "ask clarifying questions about specific inputs and receive immediate feedback on desired behavior" [15]. This methodology deliberately omits edge case specifications, compelling students to identify and resolve ambiguities. This approach cultivates a critical skill that integrates reading comprehension with coding. Furthermore, it strengthens problem designs against AI code-generation tools, which often falter with underspecified requirements. Consequently, this shifts the problem-solving paradigm to be "requirements-first" rather than "implementation-first" [16].

Future research endeavors should explore the following promising related areas for further inquiry. First, a longitudinal study could investigate the career paths of students who participate in programming contests (e.g., graduate school enrollment, job titles, etc.). This research would offer valuable, concrete insights into the long-term professional impact of engaging in these competitive environments. Second, to effectively plan a national CCSC programming contest, a thorough assessment of its feasibility and logistical challenges is needed. Finally, following a competition, a meticulous review of archived source code from incorrect submissions within the CCSC SE's contest management software is recommended. This analysis could effectively reveal prevalent student errors and misconceptions in programming logic (e.g., off-by-one errors in loops, incorrect use of data structures for specific problem types, logical flaws in algorithm implementation, common mistakes in handling edge cases, misunderstanding of specific language features, etc.). These findings could help inform curriculum enhancements for contest preparation and highlight pedagogical interventions for coaches.

The CCSC SE programming competition is a valuable platform where students can enhance technical skills, strengthen resumes, facilitate networking, and engage in direct peer comparison. It also consistently energizes the annual fall conference and enriches the undergraduate experience for participants.

References

[1] Aaron Bloomfield and Borja Sotomayor. A programming contest strategy guide. In *SIGCSE '16: Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 609–614, 2016.

[2] Andy Digh et al. Selecting and preparing student programming teams. *The Journal of Computing Sciences in Colleges*, 17(1):359–360, 2001.

[3] Barbara Johnson. Narcissistic numbers, 2024. Programming Contest Problem.

[4] Andy Digh. Royal tweet, 2014. Programming Contest Problem.

[5] Chris Healy. Slicing potatoes, 2019. Programming Contest Problem.

[6] Andy Digh. Pumpkinfest, 2018. Programming Contest Problem.

[7] Barbara Johnson. Time loop trap, 2024. Programming Contest Problem.

[8] Fabian Ernst et al. Teamwork in programming contests: 3 * 1 = 4. *Crossroads: The ACM Student Magazine*, Winter 1996.

[9] Chip Bell. Olympic dilemma, 2023. Programming Contest Problem.

[10] Stoney Jackson et al. An analysis of team performance in high school programming contests. In *SIGITE '14: Proceedings of the 15th Annual Conference on Information Technology Education*, pages 27–32, 2014.

[11] Students 1-8. Personal interviews, 2024. Conducted by Andy D. Digh, 2–13 December 2024.

[12] Steven S. Skiena. *The Algorithm Design Manual*. Springer, 2021.

[13] Steven Halim et al. *Competitive Programming 4*. LuLu, 2018.

[14] Steven S. Skiena and Miguel A. Revilla. *Programming Challenges: The Programming Contest Training Manual*. Springer, 2003.

[15] Mrigank Pawagi and Viraj Kumar. Probeable problems for beginner-level programming-with-ai contests. In *ICER '24: Proceedings of the 2024 ACM Conference on International Computing Education Research*, volume 1, pages 166–176, 2024.

[16] Austin M. Shin and Ayaan M. Kazerouni. A model of how students engineer test cases with feedback. *ACM Transactions on Computing Education*, 24(1):1–31, Jan 2024.