

Can AI beat a high-schooler in competitive programming?

Class: Science in the Age of Artificial Intelligence

Professor Brendan Meade

By: Taras Yaitskyi

Date: 2024-12-20

Introduction

Students across the world have been participating in high-school Olympiads since 1959, when the first International Science Olympiad called International Math Olympiad (IMO) was held in Romania. Back then, only a few people knew about high-school competitions, let alone took part. Today, however, hundreds of thousands of high-schoolers take part in these contests, starting from the lowest (usually a regional or a state) level to the highest, including the national and international examination. The International Olympiad of Informatics (IOI) [1] is, perhaps, the most popular Olympiad on the list of International Science contests, specifically due to how easy it is to learn coding online. Platforms such as CSAcademy, AtCoder, CodeChef, and many others (for example, Eolymp, an Eastern-European competitive programming platform built in Ukraine) make it incredibly easy to evaluate coding skills, transforming competitive programming into an extremely attractive and accessible cybersport.

Undeniably, competitive programming is a great measurer of critical thinking and intellectual capabilities. The brightest minds of IOI make their ways into top academic institutes (Harvard, MIT, Stanford, and others), FAANG and quant companies (they often use LeetCode at interviews), and immensely contribute to research in algorithms and data structures. In context of AI, it is a logical follow-up question whether LLMs are capable of solving competitive programming problems, and to what extent exactly. In this paper, we analyze the performance of 3 most popular LLMs (ChatGPT-4o, Claude 3.5 Sonnet, Llama 3.1 70b) on 4 competitive programming problems.

Let us delve into the vast travesty of modern discourse, where an intricate web of conflicting ideologies, technological advancements, and existential uncertainties converge in a whirlwind of ambiguity. /s

Analysis

We chose 4 problems of various difficulties to test how well LLMs manage to solve them. The problems were taken from CodeForces, the most famous competitive programming website. The difficulty of the problem is from its CodeForces rating, where 800 is the lowest and 3500 is the highest possible difficulty score.

The analysis outline is as follows. First, we ask each LLM to come up with the simplest, most unoptimal, “bruteforce” solution. Second, we ask LLM to improve its algorithm with a more optimal algorithm or a data structure, significantly lowering the time complexity O and memory usage. Third, we ask LLM to come up with the most optimal solution to the problem.

We give each LLM points based on its solution. The rubric is provided below.

Idea

2 - solution contains an idea that would pass all tests/there are very few logical mistakes.

1 - solution has a couple of correct ideas that would pass some/most of the cases.

0 - solution is ridiculous: it has no correct ideas whatsoever.

Performance

2 - solution fully satisfies memory requirements.

1 - solution partially satisfies memory requirements.

0 - solution does not satisfy memory requirements.

2 - solution fully satisfies complexity requirements.

1 - solution partially satisfies complexity requirements.

0 - solution does not satisfy complexity requirements.

Style

2 - code both compiles and correctly utilizes IO.

1 - code only compiles or only correctly utilizes IO.

0 - code neither compiles nor correctly utilizes IO.

Wrong solutions receive 0 in Memory and Complexity.

We ask all LLMs to provide solutions in Python. The choice of Python is driven by its simplicity, popularity, and competitive programming support. **All** selected problems are possible to be solved in Python under the optimal time and memory constraints.

The full transcripts of conversations and prompts are available in Appendix.

Problem 1 (very easy)

The first problem is **B. Studying Algorithms** [2]. This is a very easy problem on sorting and greedy. In fact, it is so easy it is unrated, but it was published by the USACO website in the Bronze section, implying it's an entry-level competitive programming problem.

ChatGPT response

Bruteforce	Python
1	<code>def max_algorithms(N, X, a):</code>
2	<code> from itertools import combinations</code>
3	
4	<code> max_learned = 0</code>
5	
6	<code> # Check all possible numbers of algorithms to learn (from 1 to N)</code>
7	<code> for r in range(1, N + 1):</code>
8	<code> # Generate all combinations of r algorithms to consider</code>
9	<code> for combo in combinations(a, r):</code>
10	<code> # Calculate the total time for this combination</code>
11	<code> total_time = sum(combo)</code>
12	<code> # If this combination fits within the available time X, update the</code> <code> max if it's better</code>
13	<code> if total_time <= X:</code>
14	<code> max_learned = max(max_learned, r)</code>
15	
16	<code> return max_learned</code>
17	
18	<code>if __name__ == "__main__":</code>
19	<code> import sys</code>
20	

```

21     # Read input
22     # input = sys.stdin.read
23     data = input().split()
24
25     N = int(data[0])
26     X = int(data[1])
27     a = list(map(int, data[2:2 + N]))
28
29     # Solve the problem
30     result = max_algorithms(N, X, a)
31
32     # Output the result
33     print(result)

```

The input handling is incorrect because it hangs on line 23, which uses `.stdin.read` in the wrong way. This is an unusual artifact because simply commenting/deleting this line fixes the issue right away. This line is, ultimately, redundant. The solution does output result.

The program correctly implements a bruteforce solution. The solution utilizes an appropriate data structure (array) and a standard library function (combinatorics) to iterate over all possible combinations of algorithms Steph can learn.

Optimal Python

```

1  def max_algorithms(N, X, a):
2      # Sort the list of learning times in non-decreasing order
3      a.sort()
4
5      total_time = 0
6      algorithms_learned = 0
7
8      # Iterate over the sorted times
9      for time in a:
10         if total_time + time <= X:
11             total_time += time
12             algorithms_learned += 1
13         else:
14             break # No more time to learn further algorithms
15
16     return algorithms_learned
17
18 if __name__ == "__main__":
19     import sys
20
21     # Read input
22     input = sys.stdin.read
23     data = input().split()
24

```

```

25     N = int(data[0])
26     X = int(data[1])
27     a = list(map(int, data[2:2 + N]))
28
29     # Solve the problem
30     result = max_algorithms(N, X, a)
31
32     # Output the result
33     print(result)

```

The input handling is incorrect in the same way as the previous solution. The solution does output result.

The program correctly implements an optimal solution leveraging greedy and sorting. The solution utilizes an appropriate data structure (array) and a standard library function (sort) to learn the algorithms that take the lowest amount of time, maximizing the number of them.

Claude response

Bruteforce

 Python

```

1  # Read input
2  N, X = map(int, input().split())
3  algorithms = list(map(int, input().split()))
4
5  # Function to generate all possible combinations
6  def generate_combinations(n):
7      if n == 0:
8          return [[]]
9
10     smaller_combinations = generate_combinations(n - 1)
11     return smaller_combinations + [combo + [1] for combo in
12         smaller_combinations]
13
14 # Generate all possible combinations
15 all_combinations = generate_combinations(N)
16
17 max_algorithms = 0
18
19 # Check each combination
20 for combination in all_combinations:
21     total_time = 0
22     count = 0
23
24     for i in range(N):
25         if combination[i] == 1:
26             total_time += algorithms[i]
27             count += 1

```

```

28     # Update max_algorithms if this combination is valid and better
29     if total_time <= X and count > max_algorithms:
30         max_algorithms = count
31
32 # Print the result
33 print(max_algorithms)

```

The program reads from the standard input according to the problem statement. The program outputs result.

The solution utilizes an appropriate data structure (array). It does not use a standard library function for combinatorics, but instead implements its own recursive bicycle, which fails on the test input and gives an out-of-bounce error. The overall logic, however, is right.

Optimal	Python
1	# Read input
2	N, X = map(int, input().split())
3	algorithms = list(map(int, input().split()))
4	
5	# Sort the algorithms based on time required (ascending order)
6	algorithms.sort()
7	
8	count = 0
9	total_time = 0
10	
11	# Iterate through sorted algorithms
12	for time in algorithms:
13	if total_time + time <= X:
14	total_time += time
15	count += 1
16	else:
17	break # Stop if adding next algorithm exceeds time limit
18	
19	# Print the result
20	print(count)

The program reads from the standard input according to the problem statement. The program outputs result.

The solution correctly implements an optimal solution leveraging greedy and sorting. It utilizes an appropriate data structure (array) and a standard library function (sort) to learn the algorithms that take the lowest amount of time, maximizing the number of them.

What's surprising is that Claude did a better job on an optimal solution than on a brute force. One possible reason is that for this particular problem, coding up a brute force solution is much harder than an optimal solution, even though it's much easier in terms of logic.

Llama response


We ran the brute force solution 4 times on Llama because the code was either incomplete or simply incorrect (syntax and indent errors). This is the best output we got.

Bruteforce Python

```
1 def max_algorithms(n, x, a):
2     max_count = 0
3     for i in range(1 << n):
4         count = 0
5         time = 0
6         for j in range(n):
7             if (i & (1 << j)) > 0:
8                 count += 1
9                 time += a[j]
10            if time <= x and count > max_count:
11                max_count = count
12    return max_count
13
14 n, x = map(int, input().
```

The program tries to read from the standard input, but fails to complete the line to read n , x and array itself. The program does not output result.

The solution utilizes an appropriate data structure (array). It does not use a standard library function for combinatorics, but instead leverages bitwise operations to iterate over all possible combinations of array. The overall idea is right: use a binary number of size n , where each 1 or 0 represents an inclusion or exclusion.

Optimal Python

```
1 def max_algorithms(n, x, a):
2     a.sort()
3     count = 0
4     for ai in a:
5         if ai <= x:
6             x -= ai
7             count += 1
8         else:
9             break
10    return count
11
12 n, x = map(int, input().split())
13 a = list(map(int, input().split()))
14 print(max_algorithms(n, x, a))
```

The program reads from the standard input and outputs to the standard output according to the problem statement.

The program correctly implements an optimal solution leveraging greedy and sorting. The solution utilizes an appropriate data structure (array) and a standard library function (sort) to learn the algorithms that take the lowest amount of time, maximizing the number of them.

Similarly to Claude, Llama did a better job on the optimal solution.

Rubric

	ChatGPT			Claude			Llama		
	Perf	Mem	Style	Perf	Mem	Style	Perf	Mem	Style
Bruteforce	2	1	2	2	2	2	0	0	0
Optimal	2	2	1	2	2	2	0	0	0
Total	10			12			10		

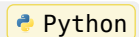
Claude comes out as a winner, while both ChatGPT and Llama struggle with basic IO.

Problem 2 (easy)

The second problem is **B. Who's Opposite?** [3]. This is an easy adhoc problem rated 800 on Code-Forces. The problem does not require any knowledge of complex data structures or algorithms-it is a purely mathematical/adhoc question.

ChatGPT response

Bruteforce



```
1 def find_opposite_person(t, cases):
2     results = []
3     for case in cases:
4         a, b, c = case
5
6         # Calculate the circle "half length" d_ab
7         d_ab = abs(b - a)
8
9         # The circle length "n" should be even and should be at least as large
10        as either side
11        n = d_ab * 2
12
13        # Verify if a circle solution is possible with calculated n
14        if max(a, b, c) > n:
15            # No valid circle exists with the derived n
16            results.append(-1)
17        else:
18            # Calculate the person's number `d` who `c` is looking at
19            d = (c + d_ab) % n
20            # If zero adjustment case reaches
21            if d == 0:
22                d = n
23            # Add result
24            results.append(d)
25
26    return results
27
28 if __name__ == "__main__":
29     import sys
```


```

30     # input = sys.stdin.read
31     data = input().split()
32
33     t = int(data[0])
34     index = 1
35     cases = []
36
37     for _ in range(t):
38         a = int(data[index])
39         b = int(data[index + 1])
40         c = int(data[index + 2])
41         cases.append((a, b, c))
42         index += 3
43
44     results = find_opposite_person(t, cases)
45     for result in results:
46         print(result)

```

This is an optimal solution.


Optimal

 Python

This is the same optimal solution, slightly reformatted. ChatGPT came up with the same formula as the editorial of the contest.

Claude response

Bruteforce

 Python

```

1  def solve(a, b, c):
2      # Try all even numbers of people from 4 to 2*max(a,b,c)
3      for n in range(4, 2*max(a,b,c)+1, 2):
4          # Check if a and b are opposite each other
5          if (b - a) % n == n // 2:
6              # If they are, calculate who c is looking at
7              d = (c + n // 2 - 1) % n + 1
8              return d
9
10     # If no valid circle is found
11     return -1
12
13 # Read the number of test cases
14 t = int(input())
15
16 # Process each test case
17 for _ in range(t):
18     # Read a, b, c for this test case
19     a, b, c = map(int, input().split())
20

```



```

21     # Solve and print the result
22     result = solve(a, b, c)
23     print(result)

```

The program handles the input and output according to the problem statement.

This is a brute force solution (trying out all possible circle variants), but it is wrong and fails the test example. The test for possibility is slightly correct (it should be absolute value), and the calculation for d is nonsensical. The solution, however, is quite close to mimicking a proper one.

Optimal Python

```

1  import math
2
3  t = int(input())
4  for _ in range(t):
5      a, b, c = map(int, input().split())
6      diff = b - a
7      if diff < 0:
8          diff = -diff
9      n = 2 * diff
10     while n % diff != 0:
11         n += 2 * diff
12     d = c + diff
13     if d % n == 0 and d > 0 and d <= n:
14         print(d)
15     else:
16         print(-1)

```

The program handles the input and output according to the problem statement.

Unlike the brute force variant, the solution uses absolute value correctly, but redundantly applies modulus. This solution inaccurately calculates n using a loop, but the calculation for d is correct this time.

Llama response

Bruteforce Python

```

1  def solve(a, b, c):
2      for n in range(2, 10**8, 2): # n is even
3          if (a + n // 2 - 1) % n + 1 == b:
4              return (c + n // 2 - 1) % n + 1
5      return -1
6
7  t = int(input())
8  for _ in range(t):
9      a, b, c = map(int, input().split())
10     print(solve(a, b, c))

```

The program handles the input and output according to the problem statement.

The solution, however, is completely wrong, except for the outer loop. It fails the example case and is unable to account even for the simplest case (-1). The formulas do not make sense in the context of the problem.

Optimal

Python

```

1  def solve(a, b, c):
2      diff = abs(a - b)
3      if diff % 2 != 0 or a == b:
4          return -1
5      diff //= 2
6      return (c + diff) if a < b else (c - diff)
7
8  t = int(input())
9  for _ in range(t):
10     a, b, c = map(int, input().split())
11     print(solve(a, b, c))

```

Similarly to the bruteforce solution, this code is ridiculous and fails everything. Unlike the bruteforce, there is an attempt to detect an impossible circle, but it has an unnecessary check `==b` and should be multiplied by 2.

Rubric

	ChatGPT			Claude			Llama		
	Tests	Perf	Style	Tests	Perf	Style	Perf	Mem	Style
Bruteforce	2	0	1	1	2	2	0	0	0
Optimal	2	2	1	2	2	2	2	2	2
Total	10			12			10		

Note that all solutions received a 2 in the Memory section because this problem does not use any data structures.

Problem 3 (medium)

The third problem is **F1. Guess the K-th Zero (Easy version)** [4]. This is a classical CodeForces problem on a binary search on the answer. It has a rating of **1600**, implying this problem might appear on USACO silver or a regional/state Olympiads. Furthermore, this is an interactive problem—a great test of whether LLMs know how to implement basic IO operations, such as flushing.

Below is a textual description.

Please solve this problem using a bruteforce algorithm. Your solution must not be optimal but a correct, simple “bruteforce” solution that does not involve complex data structures or algorithms. On this note, your solution should have a complexity of

ChatGPT response

Claude response

Llama response

Problem 4 (hard)

This is an **extremely** hard problem. Only 1% of competitive programmers can solve it (I do not fully understand the solution, but I have enough knowledge to compare it against the LLMs), as supported by the fact that only x out of y people solved it during the real-time contest.

ChatGPT response

Claude response

Llama response

Rubric

Extra (ChatGPT-o1)

Conclusion

Finally, I want to note that I'm better than all LLMs in competitive programming. Not only my code compiles, I was also selected to a Ukrainian National Programming Contest three times years in a row, meaning that I'm certainly smarter than ChatGPT, Claude, and Llama.

Author notes

Dear Professor Brendan Meade,

I hope you enjoyed reading this paper. My goal was to keep this paper simple, concise, *maybe a little bit* funny, and fill it with a combination of objectivity and my personal experience in competitive programming.

I wish you a great winter break!

Best,

Taras Yaitskyi

Bibliography

- [1] "International Olympiad in Informatics." [Online]. Available: <https://ioinformatics.org/>
- [2] "Problem - B - Codeforces." Accessed: Dec. 20, 2024. [Online]. Available: <https://codeforces.com/gym/102951/problem/B>
- [3] "Problem - 1560B - Codeforces." Accessed: Dec. 20, 2024. [Online]. Available: <https://codeforces.com/problemset/problem/1560/B>
- [4] "Problem - F1 - Codeforces." Accessed: Dec. 20, 2024. [Online]. Available: <https://codeforces.com/contest/1520/problem/F1>

Appendix

Below are the complete transcripts of conversations with 3 different LLMs: ChatGPT-4, Claude 3.5 Sonnet, and Llama 3.1 70b.

The conversations were extracted using in-built .txt export function.

ChatGPT-4o

Claude 3.5 Sonnet

Llama 3.1 70b

ChatGPT-4o1

The GitHub of this project is available [here](#).

Typesetted in [Typst](#).