Prompt Engineering Frameworks

Roy Van

Part 1:

Task: I want AI to explain the time complexity of C++ codes using Big O notation. Analyze its complexity and efficiency variations with different inputs and provide an optimized sorting algorithm.

Context: We are analyzing a sorting algorithm implemented using Selection Sort. Selection Sort takes the minimum value in an array repeatedly and switches its position with the first unsorted term. This will continue until the entire array is sorted.

References: Here is an example of a C++ code of selection sort:

```
#include <iostream>
using namespace std;

void selectionSort(int arr[], int n){
    for(int i = 0; i < n; i++){
        for(int j = i + 1; j < n; j++){
            if(arr[i] > arr[j]){
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
                // swap(arr[i], arr[j]);
            }
        }
    }
}

int main(){
    int arr[] = {5, 5, 4, 2, 5, 6, 3, 4};
    int n = sizeof(arr) / sizeof(arr[0]); // # of elements in the array
    selectionSort(arr, n);

    // Print the resulting array
    for(int i = 0; i < n; i++){
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Evaluate: I will evaluate AI's response based on the aspects below:

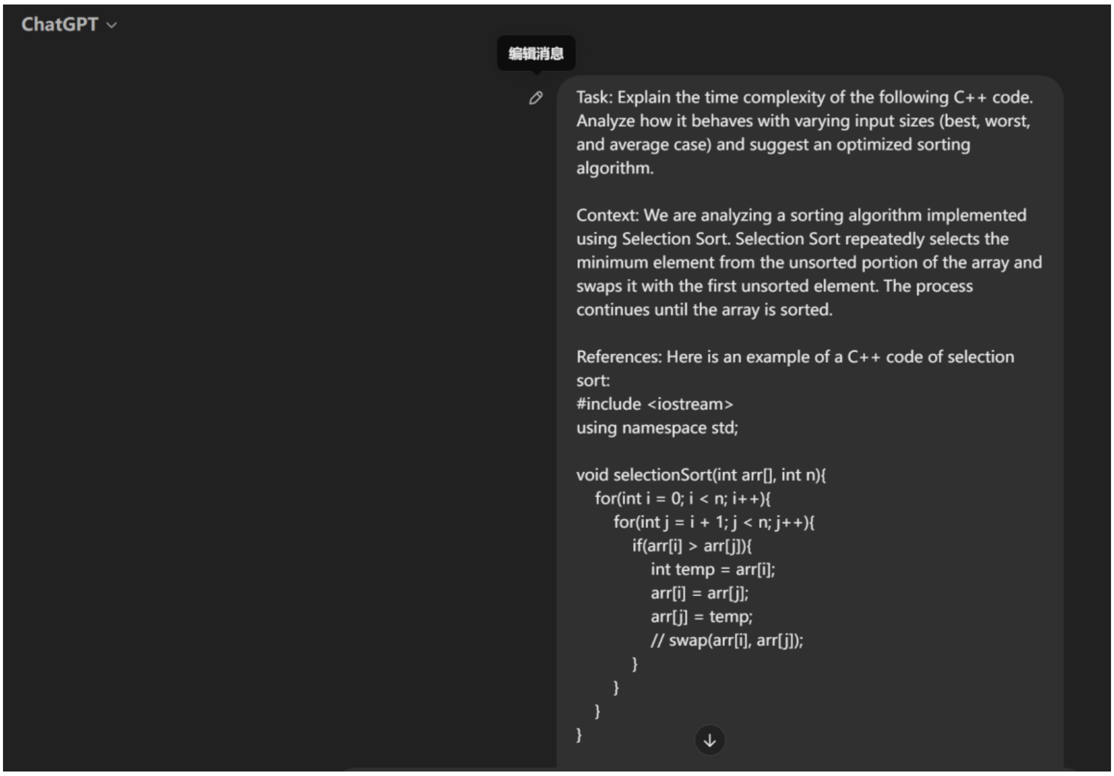
- Does AI identify the time complexity of selection sort correctly? The answer should be $O(n^2)$ due to the two loops within the codes. Does AI also explain the reason of Big O of selection sort being $O(n^2)$ and make connections to the example I have provided?

- Does AI analyze the best, average and worst case of selection sort? Their Big O should all be the same which is $O(n^2)$.
- Suggestions for further improvements. To optimize the efficiency of sorting algorithms, what are some optimizations I can do? Does AI suggest better sorting algorithms such as quick sort and merge sort which has an average complexity of $O(n \log n)$ and explain why they are better algorithms for larger scale inputs?

Iterate: If AI's outputs are not ideal, I will make the following refinements.

- I could ask for a more detailed report about in the best-case scenario, although fewer swaps are made, how selection sorts behave and affects their overall time complexity.
- I could specify that the AI should make comparisons between more advanced sorting algorithms such as merge sort and quick sort, and ask for their performances under extreme conditions.
- As an additional layer, I could ask for information on analyzing selection sort's space complexity(which is known to be 1).

Paet 2:



ChatGPT

```
    }
}

int main(){
    int arr[] = {5, 5, 4, 2, 5, 6, 3, 4};
    int n = sizeof(arr) / sizeof(arr[0]); // # of elements in the array
    selectionSort(arr, n);

    // Print the resulting array
    for(int i = 0; i < n; i++){
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Explain how Selection Sort behaves in the best-case scenario, where fewer swaps are made, and analyze how this affects its overall time complexity. Compare this with its worst-case time complexity. Additionally, compare Selection Sort with more advanced sorting algorithms like Merge Sort and Quick Sort, and discuss their performances under extreme conditions (e.g., large input size, nearly sorted arrays). Finally, analyze the space complexity of Selection Sort and compare it with the space complexity of Merge Sort and Quick Sort

< 2/2 >

AI Response:

ChatGPT

< 2/2 >

Time Complexity of Selection Sort:

Selection Sort has the same time complexity in both the best and worst cases, which is $O(n^2)$. This is because:

1. **Outer Loop:** The outer loop runs n times, where n is the size of the array.
2. **Inner Loop:** For each element in the outer loop, the inner loop runs for $n - i - 1$ iterations, where i is the current index of the outer loop.

Thus, the number of comparisons is roughly the sum of the first $n - 1$ integers, which gives a total of $\frac{n(n-1)}{2} = O(n^2)$ comparisons.

In your code, swaps are performed only when a smaller element is found (i.e., when the condition `if (arr[i] > arr[j])` is true), but the total number of comparisons is still $O(n^2)$. This results in a **time complexity** of:

- **Best Case:** $O(n^2)$ — even if the array is already sorted, the algorithm still performs the same number of comparisons.
- **Worst Case:** $O(n^2)$ — this occurs when the array is in reverse order, requiring the maximum number of comparisons and swaps.
- **Average Case:** $O(n^2)$ — on average, it also takes about n^2 comparisons to sort the array.

Space Complexity of Selection Sort:

Selection Sort is an **in-place sorting algorithm**, meaning it sorts the array without using extra space proportional to the input size. The space ↓ ıplexity is:

ChatGPT

Space Complexity of Selection Sort:

Selection Sort is an **in-place sorting algorithm**, meaning it sorts the array without using extra space proportional to the input size. The space complexity is:

- **Space Complexity:** $O(1)$, as the algorithm only uses a small constant amount of extra space (for temporary storage when swapping elements).

This is one of the main advantages of Selection Sort over algorithms like Merge Sort and Quick Sort, which require additional space for recursion or temporary arrays.

Comparison with Other Sorting Algorithms:

1. Merge Sort:

- **Time Complexity:**
  - **Best Case:** $O(n \log n)$ — This occurs when the array is already sorted, but Merge Sort still performs the same number of operations.
  - **Worst Case:** $O(n \log n)$ — Merge Sort's performance is consistent across all cases.
  - **Average Case:** $O(n \log n)$ — This is typical for Merge Sort.
- **Space Complexity:** $O(n)$, because Merge Sort requires an auxiliary array for merging subarrays.
- **Performance:** Merge Sort performs better than Selection Sort in terms of time complexity, especially for large datasets. It is highly stable and works well for arrays that are already nearly sorted.

2. Quick Sort:

- **Time Complexity:**

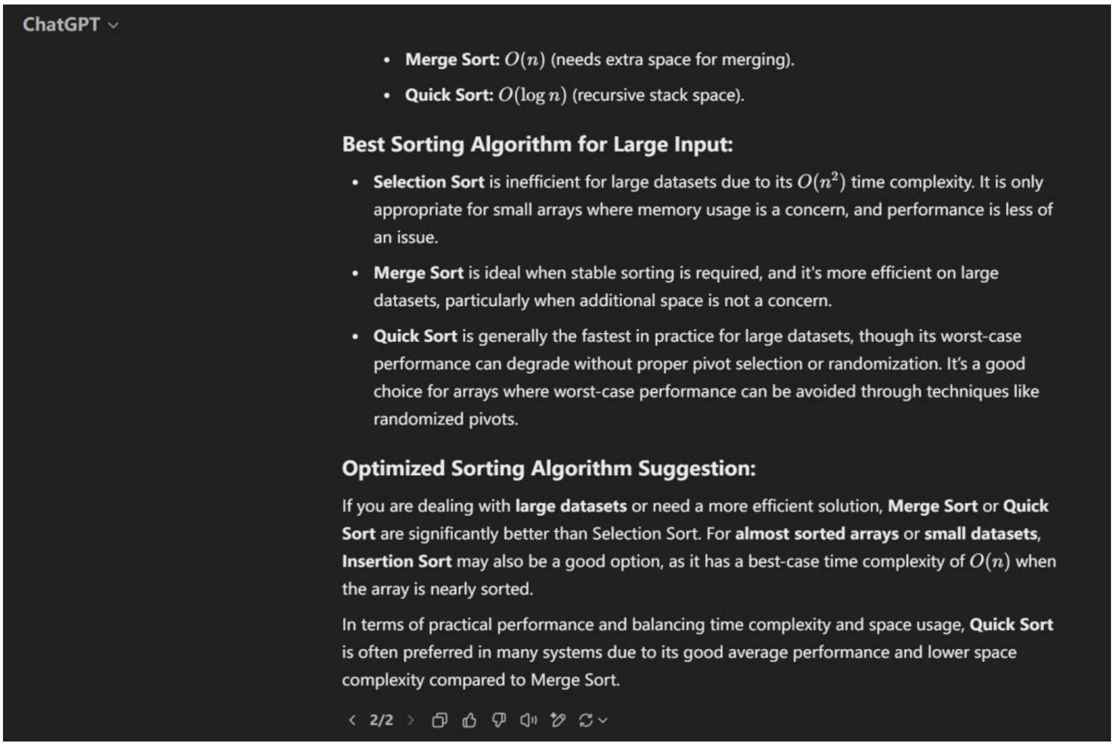
ChatGPT

2. Quick Sort:

- **Time Complexity:**
  - **Best Case:** $O(n \log n)$ — This occurs when the pivot divides the array evenly, ensuring balanced partitions.
  - **Worst Case:** $O(n^2)$ — This happens when the pivot is poorly chosen (e.g., when the array is already sorted or reverse sorted).
  - **Average Case:** $O(n \log n)$ — Quick Sort generally performs well in most situations with a randomized pivot.
- **Space Complexity:** $O(\log n)$ — For the recursive stack used during partitioning, which is smaller than Merge Sort's auxiliary space.
- **Performance:** Quick Sort is typically faster than both Selection Sort and Merge Sort in practice, especially on large arrays. However, it can suffer from poor performance if the pivot is not chosen carefully (e.g., when dealing with nearly sorted data).

Comparison Summary:

- **Time Complexity:**
  - **Selection Sort:** $O(n^2)$ in all cases.
  - **Merge Sort:** $O(n \log n)$ in all cases.
  - **Quick Sort:** $O(n \log n)$ on average, but $O(n^2)$ in the worst case.
- **Space Complexity:**
  - **Selection Sort:** $O(1)$ (in-place).
  - **Merge Sort:** $O(n)$ (needs extra space f[illegible]nerging).



ChatGPT

- **Merge Sort:** $O(n)$ (needs extra space for merging).
- **Quick Sort:** $O(\log n)$ (recursive stack space).

Best Sorting Algorithm for Large Input:

- **Selection Sort** is inefficient for large datasets due to its $O(n^2)$ time complexity. It is only appropriate for small arrays where memory usage is a concern, and performance is less of an issue.
- **Merge Sort** is ideal when stable sorting is required, and it's more efficient on large datasets, particularly when additional space is not a concern.
- **Quick Sort** is generally the fastest in practice for large datasets, though its worst-case performance can degrade without proper pivot selection or randomization. It's a good choice for arrays where worst-case performance can be avoided through techniques like randomized pivots.

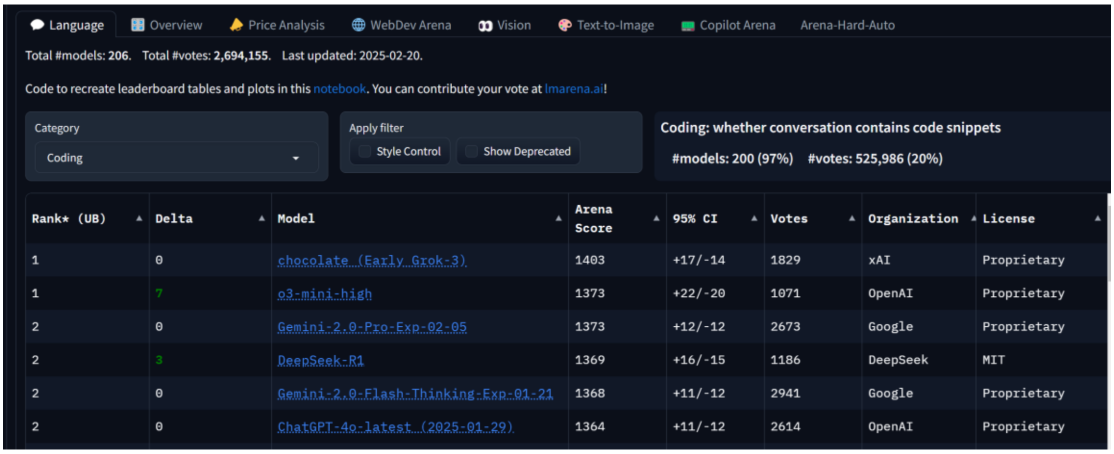
Optimized Sorting Algorithm Suggestion:

If you are dealing with **large datasets** or need a more efficient solution, **Merge Sort** or **Quick Sort** are significantly better than Selection Sort. For **almost sorted arrays** or **small datasets**, **Insertion Sort** may also be a good option, as it has a best-case time complexity of $O(n)$ when the array is nearly sorted.

In terms of practical performance and balancing time complexity and space usage, **Quick Sort** is often preferred in many systems due to its good average performance and lower space complexity compared to Merge Sort.

2/2

Chosen LLM: ChatGPT 4o



Language | Overview | Price Analysis | WebDev Arena | Vision | Text-to-Image | Copilot Arena | Arena-Hard-Auto

Total #models: 206. Total #votes: 2,694,155. Last updated: 2025-02-20.

Code to recreate leaderboard tables and plots in this notebook. You can contribute your vote at lmarena.ai!

Category: Coding | Apply filter: Style Control, Show Deprecated

Coding: whether conversation contains code snippets

#models: 200 (97%) #votes: 525,986 (20%)

Rank* (UB)	Delta	Model	Arena Score	95% CI	Votes	Organization	License
1	0	chocolate (Early Grok-3)	1403	+17/-14	1829	xAI	Proprietary
1	7	o3-mini-high	1373	+22/-20	1071	OpenAI	Proprietary
2	0	Gemini-2.0-Pro-Exp-02-05	1373	+12/-12	2673	Google	Proprietary
2	3	DeepSeek-R1	1369	+16/-15	1186	DeepSeek	MIT
2	0	Gemini-2.0-Flash-Thinking-Exp-01-21	1368	+11/-12	2941	Google	Proprietary
2	0	ChatGPT-4o-latest (2025-01-29)	1364	+11/-12	2614	OpenAI	Proprietary

From this leaderboard (https://lmarena.ai/?leaderboard), I can see that ChatGPT 4o by OpenAI is very good at answering coding related questions, this is the reason I pick this model to assist my computer science learning.

References:

https://openai.com/index/gpt-4-research/

https://arxiv.org/abs/2303.12712

https://lmarena.ai/?leaderboard

With respect to its accuracy and completeness, with some refinements based on original input, ChatGPT 4o is able to provide detailed and well-organized answers that matches my prompt.

Part 3 Reflection:

I chose my special structure for prompts to ensure that artificial intelligence can provide a versatile, well-considered response to my questions. I ask AI to conduct a thorough analysis on ‘Selection Sort’, this includes calculating and comparing the time complexity and space complexity under best case, average case and worst-case scenarios. I also asked AI to make comparisons between other efficient and complicated sorting algorithms such as merge sort and quick sort, as well as comparing their complexities. I ensured that the response covers as much useful knowledge as possible by refining my prompts through adding in more specific requirements and possibilities, so that ChatGPT 4o can better understand my prompts and provide more practical and innovative information.

Five steps framework is highly helpful for me to explore the creation of a focused prompt. It helped clarify my communication when working with AI, making sure that I am getting a thoughtful response instead of any irreverent and superficial answers. Through this exercise, I realized that with cooperating with AI, clear and logical prompts are crucial. Since when we are dealing with coding issues, or analyzing algorithms, understanding how to craft effective prompts will also help in using AI as a tool for problem-solving and learning.