



The graphic is a promotional poster for the AutoGen Hackathon. It features a central image of a smiling man in a hoodie with a blue medal around his neck, set against a green background with futuristic digital elements. The text is arranged in a grid-like fashion:

- Top left: **AutoGen Hackathon** in white text on a green background.
- Top right: A photo of a man with a medal.
- Middle left: **lab ai** logo and **24 hours** in white text on a black background.
- Middle right: Two hashtags: `# Build the future` and `# Let the hacking begin!` with corresponding code snippets `def create_solution` and `import ideas`.
- Bottom left: **02.11** in white text on a black background.
- Bottom right: **</Intermediate Level>** in white text on a green background.

**AutoGen Debugger:** AutoGen agent that can debug python programs.

- Raghavan Muthuregunathan

-

# Problem

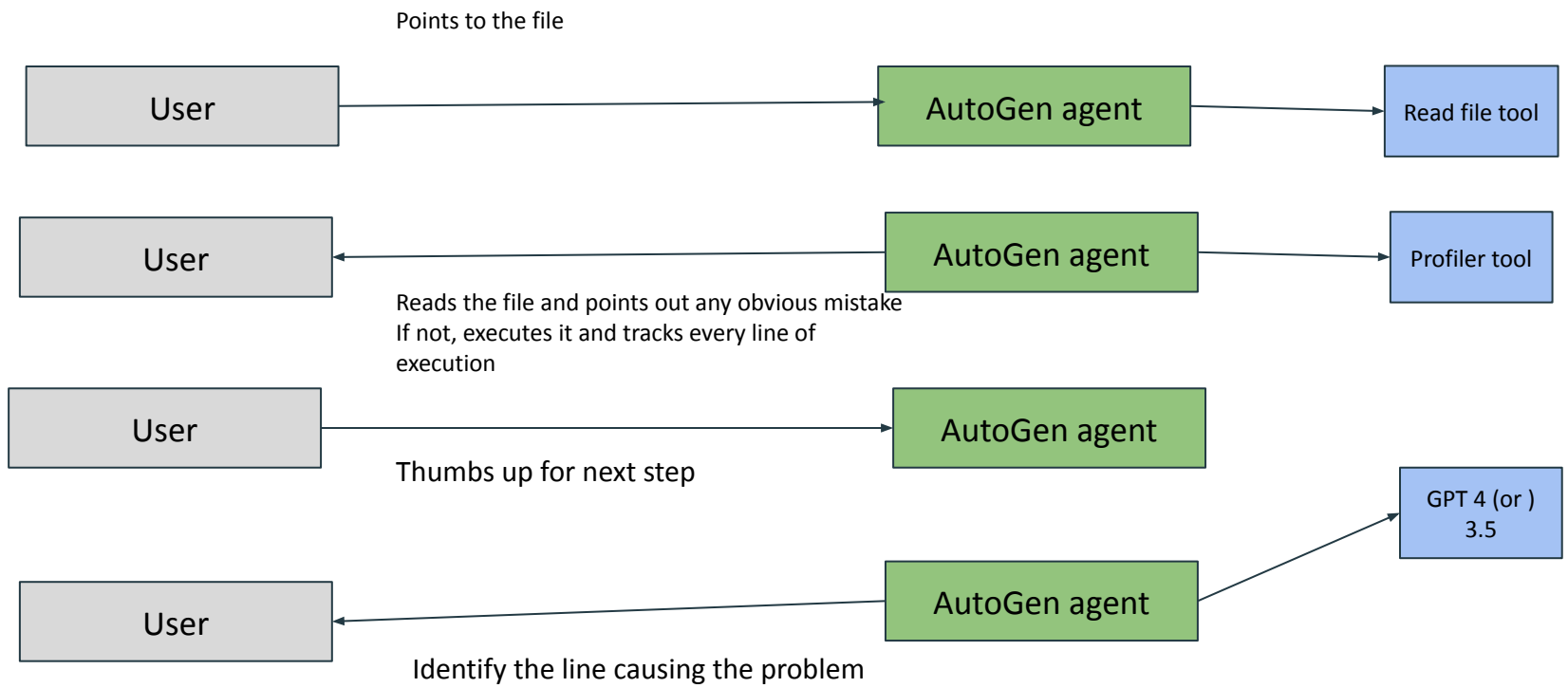
- When you write a very long program and if it does not work, we spend hours debugging the program with
  - breakpoints,
  - log statements,
  - print statements.
- Copy pasting the entire code to GPT may not solve the problem due to token limit.

# Solution

We introduce an **autogen based agent** that uses tools `ReadFileTool` and `ProfilerTool`

1. Executes the program
2. Track line by line execution
3. Identify the part that caused the bug
4. Recommend user a fix

LLMs may hallucinate



# Binary search with a bug (line:5)

```
1 def binary_search(arr, x):
2     left, right = 0, len(arr) - 1
3
4     while left <= right:
5         mid = left + right // 2 ## BUG: should be (left + right) // 2
6
7         # Check if x is present at mid
8         if arr[mid] == x:
9             return mid
10
11        # If x is greater, ignore the left half
12        elif arr[mid] < x:
13            left = mid + 1
14
15        # If x is smaller, ignore the right half
16        else:
17            right = mid - 1
18
19        # If we reach here, the element was not present
20        return -1
21
22 # Example usage:
23 arr = [2, 3, 4, 10, 40]
24 x = 10
25
26 try:
27     result = binary_search(arr, x)
28     if result != -1:
29         print(f"Element is present at index {result}")
30     else:
31         print("Element is not present in array")
32 except Exception as e:
33     print(f"An error occurred: {e}")
34
```

```
(base) raghavan@raghavan-Blade-18-RZ09-0484:~/autogen_hackathon$ python binary_search.py
```

**An error occurred: list index out of range**

```
(base) raghavan@raghavan-Blade-18-RZ09-0484:~/autogen_hackathon$
```

## Agent captures the bug by executing and following the line of execution path

An error occurred: list index out of range  
user\_proxy (to debugger):

```
Read the following file: binary_search.py
Identify the bug in the program. The execution log is as follows:
Executing line 310 in <frozen codecs>:
Executing line 268 in <frozen codecs>:
Executing line 312 in <frozen codecs>:
Executing line 321 in <frozen codecs>:
Executing line 322 in <frozen codecs>:
Executing line 324 in <frozen codecs>:
Executing line 325 in <frozen codecs>:
Executing line 1 in binary_search.py: def binary_search(arr, x):
Executing line 23 in binary_search.py: arr = [2, 3, 4, 10, 40]
Executing line 24 in binary_search.py: x = 10
Executing line 26 in binary_search.py: try:
Executing line 27 in binary_search.py:     result = binary_search(arr, x)
Executing line 2 in binary_search.py:     left, right = 0, len(arr) - 1
Executing line 4 in binary_search.py:     while left <= right:
Executing line 5 in binary_search.py:         mid = left + right // 2
Executing line 8 in binary_search.py:         if arr[mid] == x:
Executing line 12 in binary_search.py:             elif arr[mid] < x:
Executing line 13 in binary_search.py:                 left = mid + 1
Executing line 4 in binary_search.py:         while left <= right:
Executing line 5 in binary_search.py:             mid = left + right // 2
Executing line 8 in binary_search.py:             if arr[mid] == x:
Executing line 32 in binary_search.py: except Exception as e:
Executing line 33 in binary_search.py:     print(f"An error occurred: {e}")
```

debugger (to user\_proxy):

From the provided execution log, it seems like the program enters into an infinite loop after the 'left' pointer is updated to 'mid + 1' in line 13. The issue is most likely with the calculation of the 'mid' index on line 5. It should be the average of 'left' and 'right' (i.e. '(left + right) // 2'), but based on the log the program appears to be calculating it as 'left + (right // 2)', which might sometimes result in a 'mid' value that is less than 'left'.

Let's fix this bug by correctly calculating the middle index. The bug-free binary search function would then be:

```
"""python
def binary_search(arr, x):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            left = mid + 1
        else:
            right = mid - 1
    return -1

arr = [2, 3, 4, 10, 40]
x = 10
try:
    result = binary_search(arr, x)
    if result != -1:
        print("Element is present at index", str(result))
    else:
        print("Element is not present in array")
except Exception as e:
    ... print(f"An error occurred: {e}")
```

With the corrected binary search function, the correct middle index will now be calculated, preventing the infinite loop and allowing the function to correctly determine whether the element 'x' is present in 'arr' or not.

Please replace the content of 'binary\_search.py' with the above code to fix the bug, and run the program again to verify the solution.

## Slightly long program of heap\_sort implementation

```
(base) raghavan@raghavan-Blade-18-RZ09-0484:~/autogen_hackathon$ python heap_sort.py
Sorted array is
5 6 7 11 13 12 (base) raghavan@raghavan-Blade-18-RZ09-0484:~/autogen_hackathon$
```



-----  
debugger (to user\_proxy):

Based on the execution log, the output does not show an apparent error. The program appears to have executed successfully till the end. However, there's a probable logical error in the code which is hard to spot without seeing the output or having access to the full code.

That said, there seems to be an issue with both loops in this chunk of code:

```
'''python
for i in range(n // 2 - 1, 0, -1):
    heapify(arr, n, i)
'''
'''
for i in range(n-1, 0, -1):
    arr[i], arr[0] = arr[0], arr[i]
    heapify(arr, i, 0)
'''
```

The indices 'i' in both loops have been initialized to 'n // 2 - 1' and 'n - 1' respectively, and the terminating condition for the loops is 'i > 0'. This means that 'i = 0' will never be included in the loop.

However, in Python, the index of an array starts at 0, so the first element will never actually be heapified/sorted. If the smallest number is at the first position in the array, it would remain there and the output sorted array would be incorrect.

To fix this issue, the code should be modified as:

```
'''python
for i in range(n // 2 - 1, -1, -1):
    heapify(arr, n, i)
'''
'''
for i in range(n-1, -1, -1):
    arr[i], arr[0] = arr[0], arr[i] # swap
    heapify(arr, i, 0)
'''
```

Please apply these changes to your code and try running it again. Reply with "TERMINATE" if you believe I've answered your question or provide more information if you need help with something else.

-----  
Provide feedback to debugger. Press enter to skip and use auto-reply, or type 'exit' to end the conversation:

When there is a token limit problem for long programs, the agent

- Reads and runs the program,
- tracks the execution path and
- identifies the line causing the problem

Github: <https://github.com/Raghavan1988/DebuggerAgent/blob/main/debugger.py>

Linkedin: <https://www.linkedin.com/in/raghavanmit/>