A black text with white background

Description automatically generated

# DESIGN AND ANALYSIS OF ALGORITHMS

# LAB FILE

**Submitted by –**

**Name-Prabhat Mishra**

**Sap\_id-500120575**

**Enrolment No-R2142231785**

**Batch-52**

**3rd Semester,**

**BTech CSE**

**UPES Dehradun, Uttarakhand**

**CONTENTS**

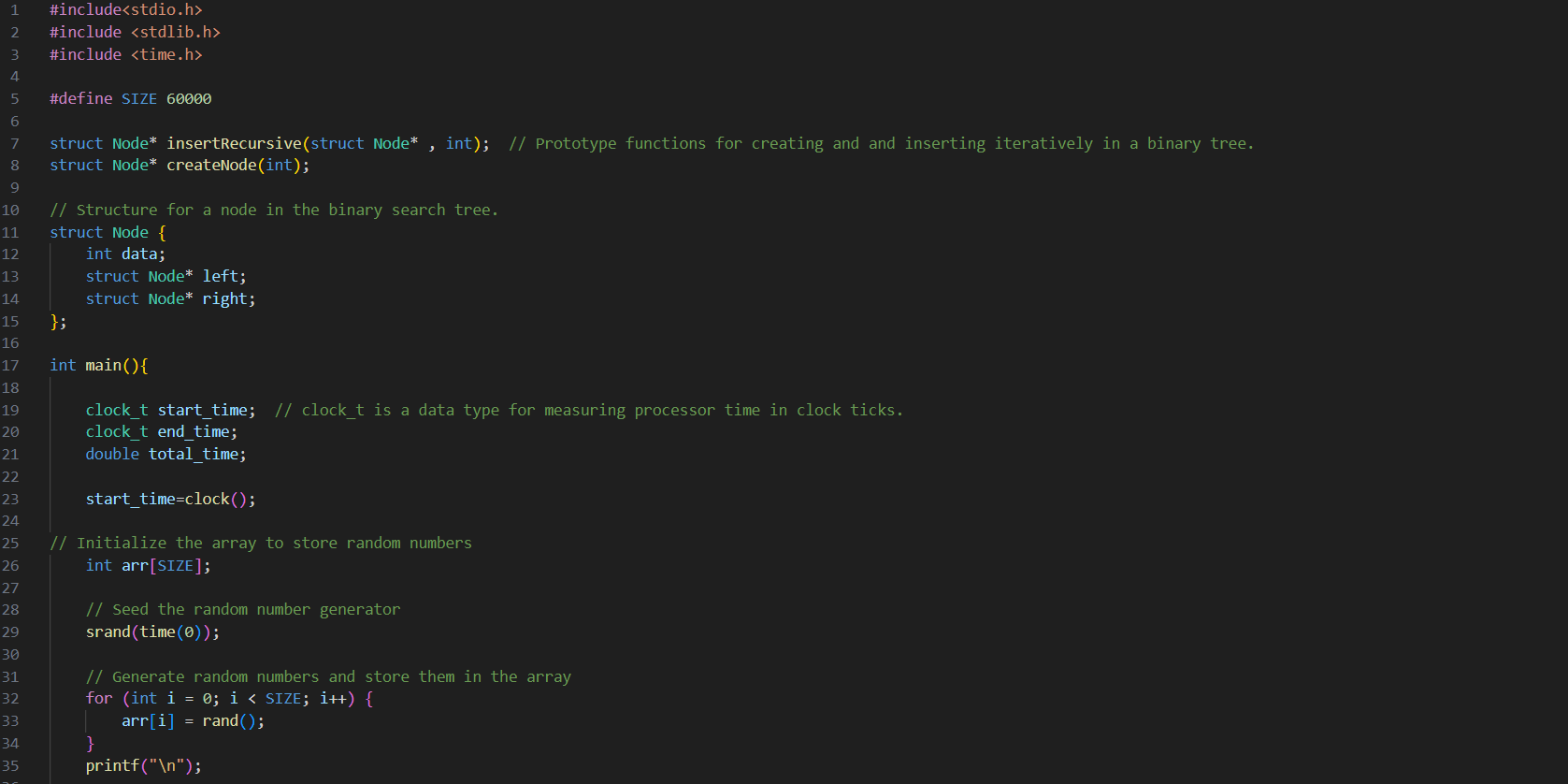
|  |  |  |  |
| --- | --- | --- | --- |
| **Sl no.** | **Experiments** | **Page No.** | **Remarks** |
| **1.** | Implement the insertion inside iterative and recursive Binary search tree and compare their  performance. | 4 |  |
| **2.** | Implement divide and conquer based merge sort and quick sort algorithms and  compare their performance for the same set of elements. | 10 |  |
| **3.** | Compare the performance of Strassen method of matrix multiplication with  traditional way of matrix multiplication. | 16 |  |
| **4.** | Implement the activity selection problem to get a clear understanding of greedy  approach. | 23 |  |
| **5.** | Get a detailed insight of dynamic programming approach by the implementation of  Matrix Chain Multiplication problem and see the impact of parenthesis positioning  on time requirements for matrix multiplication. | 27 |  |
| **6.** | Compare the performance of Dijkstra and Bellman ford algorithm for the single  source shortest path problem. | 29 |  |
| **7.** | Through 0/1 Knapsack problem, analyse the greedy and dynamic programming  approach for the same dataset. | 33 |  |
| **8.** | Implement the sum of subset. | 39 |  |
| **9.** | Compare the Backtracking and Branch & Bound Approach by the implementation  of 0/1 Knapsack problem. Also compare the performance with dynamic  programming approach. | 41 |  |
| **10.** | Compare the performance of Rabin-Karp, Knuth-Morris-Pratt and naive string-matching algorithms. | 46 |  |
| **11.** | My GitHub Link. | 52 |  |

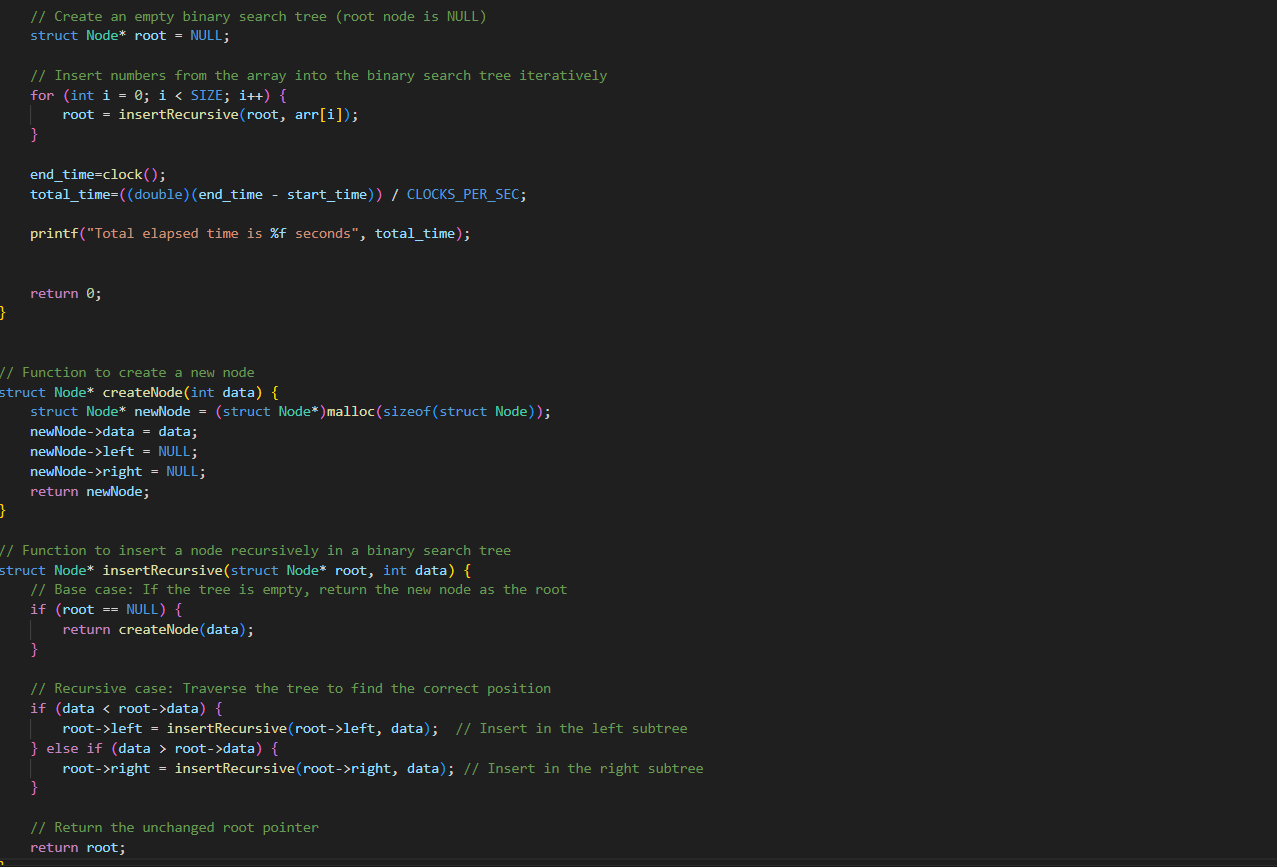
**EXPERIMENT 1**

**Implement the insertion inside iterative and recursive Binary search tree and compare their performance.**

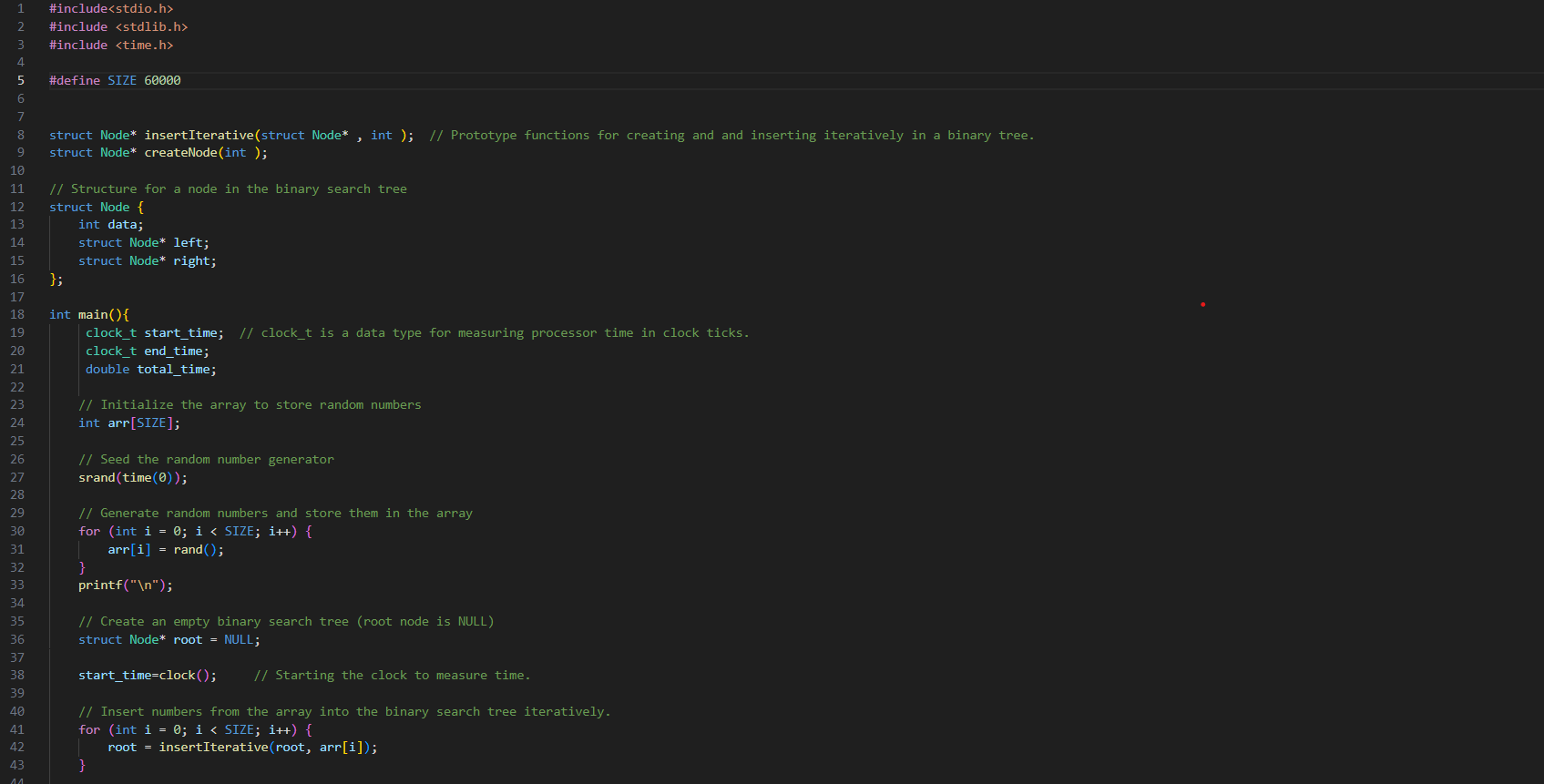
**CODE:**

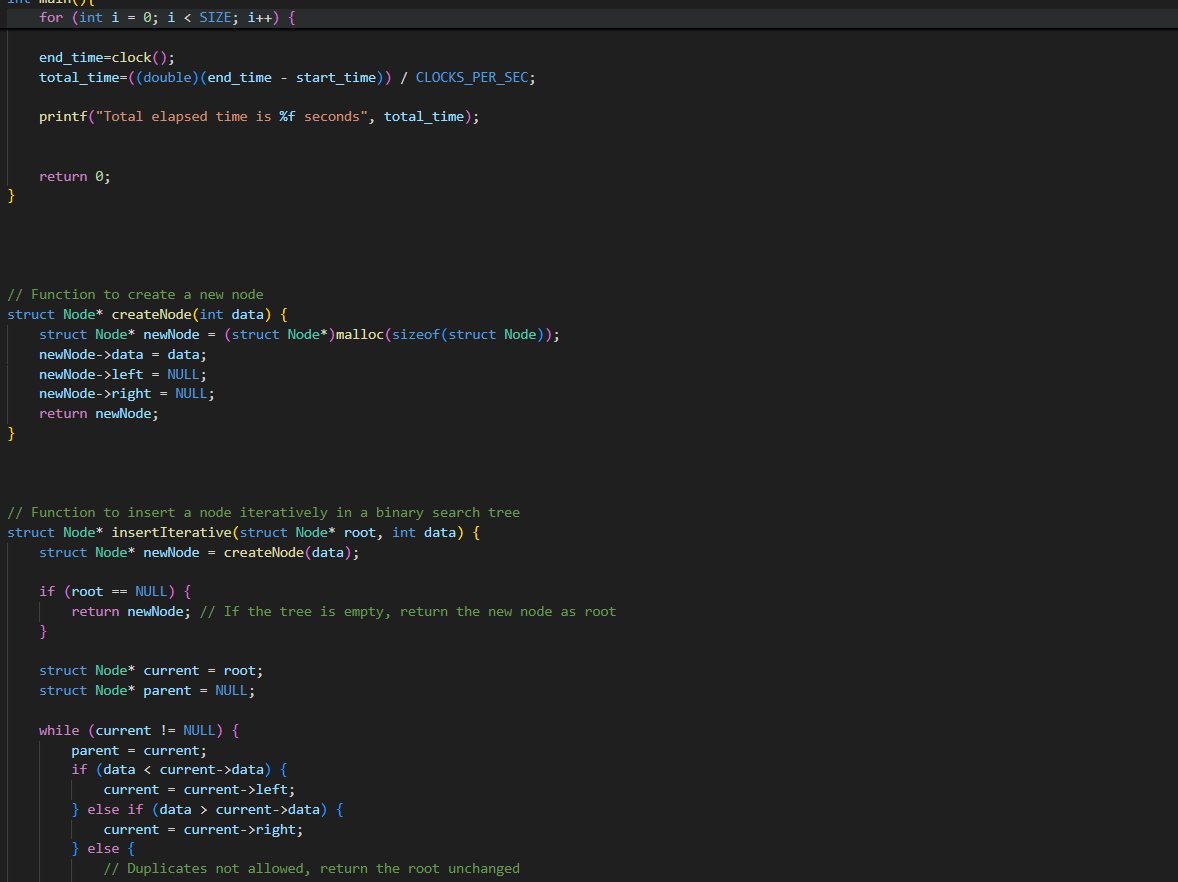
**Recursive BST:**

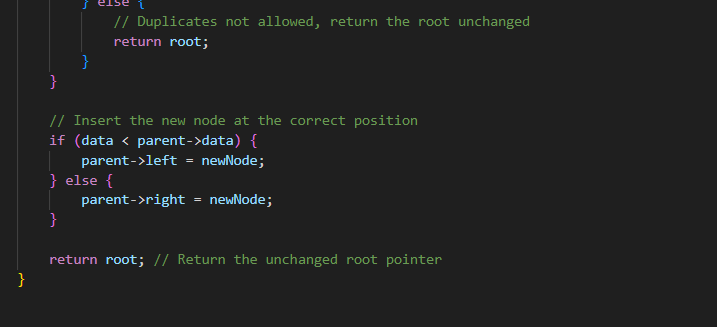




**ITERATIVE BST:**

****

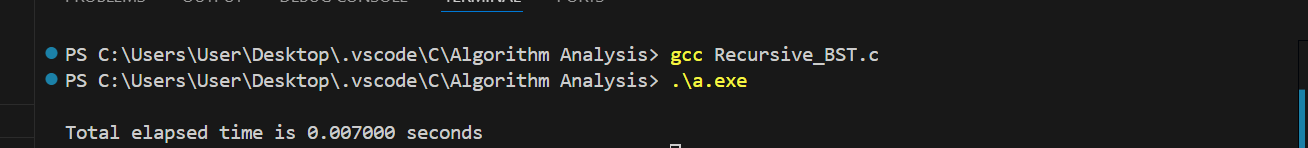
****

****

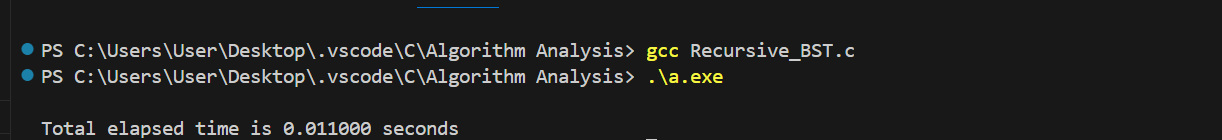
**OUTPUTS:**

**RECURSIVE BST:**

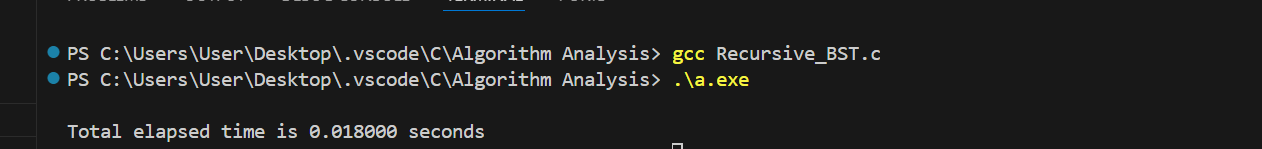
**1. N=10000**



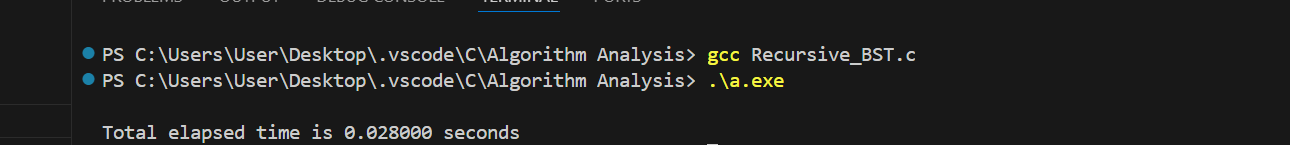
**2. N=20000**



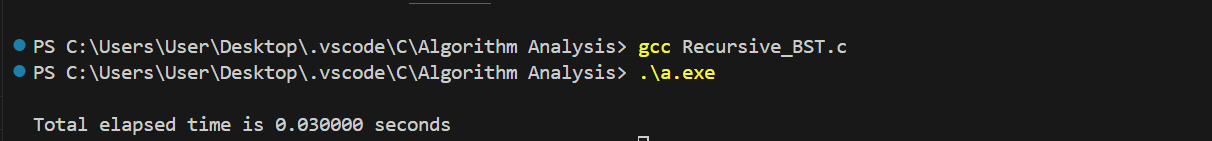
**3. N=30000**



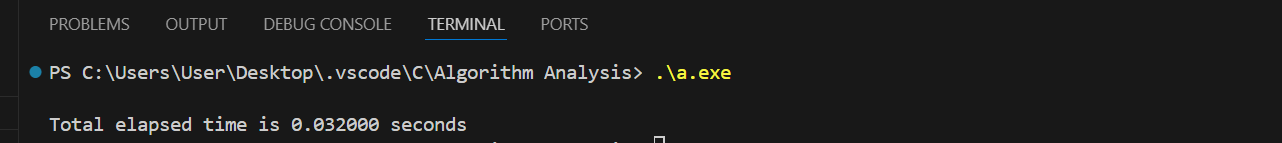
**4. N=40000**



**5. N=50000**

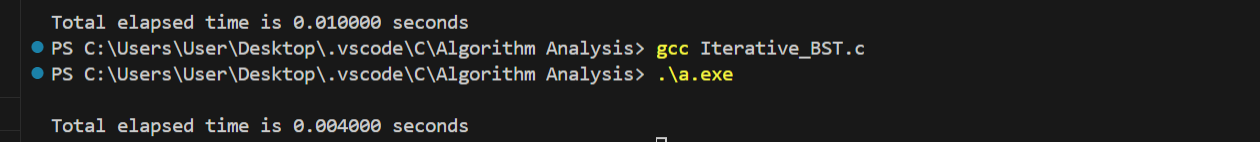


**6. N=60000**

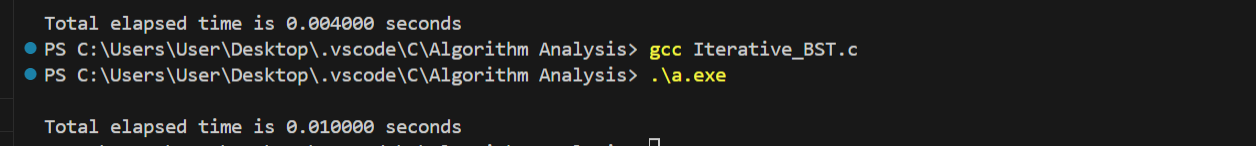


**ITERATIVE BST:**

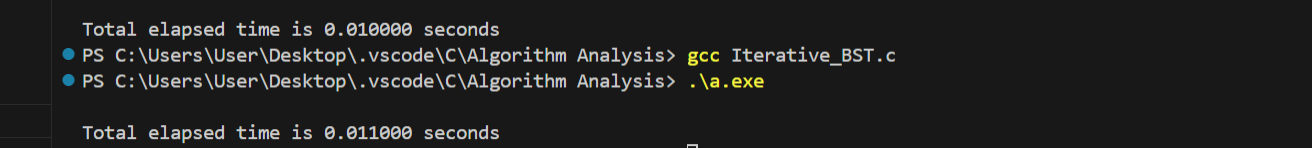
**1. N=10000**



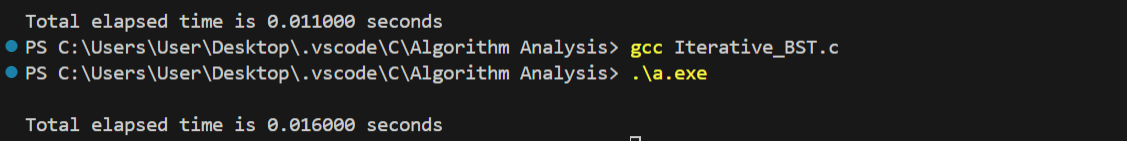
**2. N=20000**



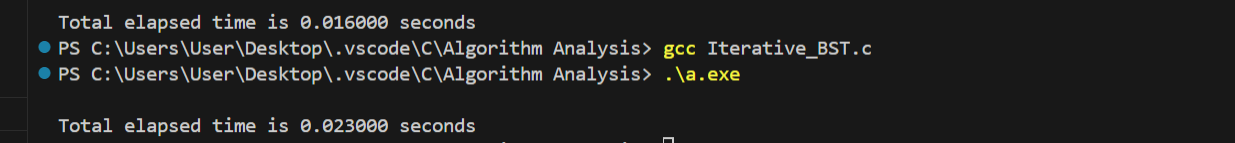
**3. N=30000**



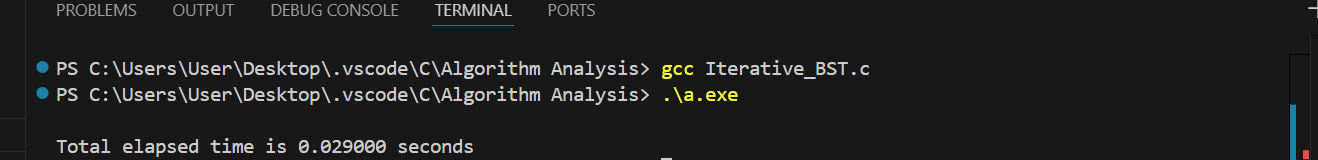
**4. N=40000**



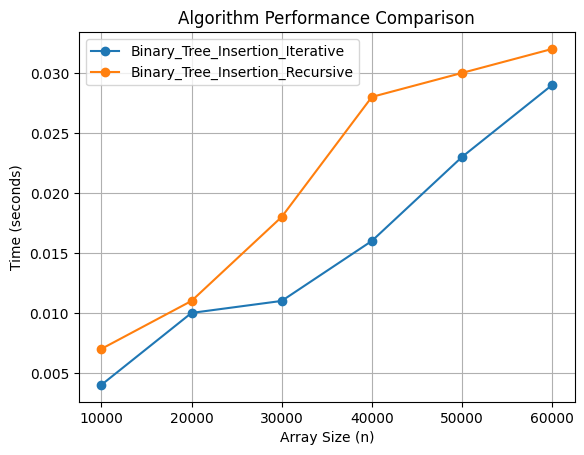
**5. N=50000**



**6. N=60000**



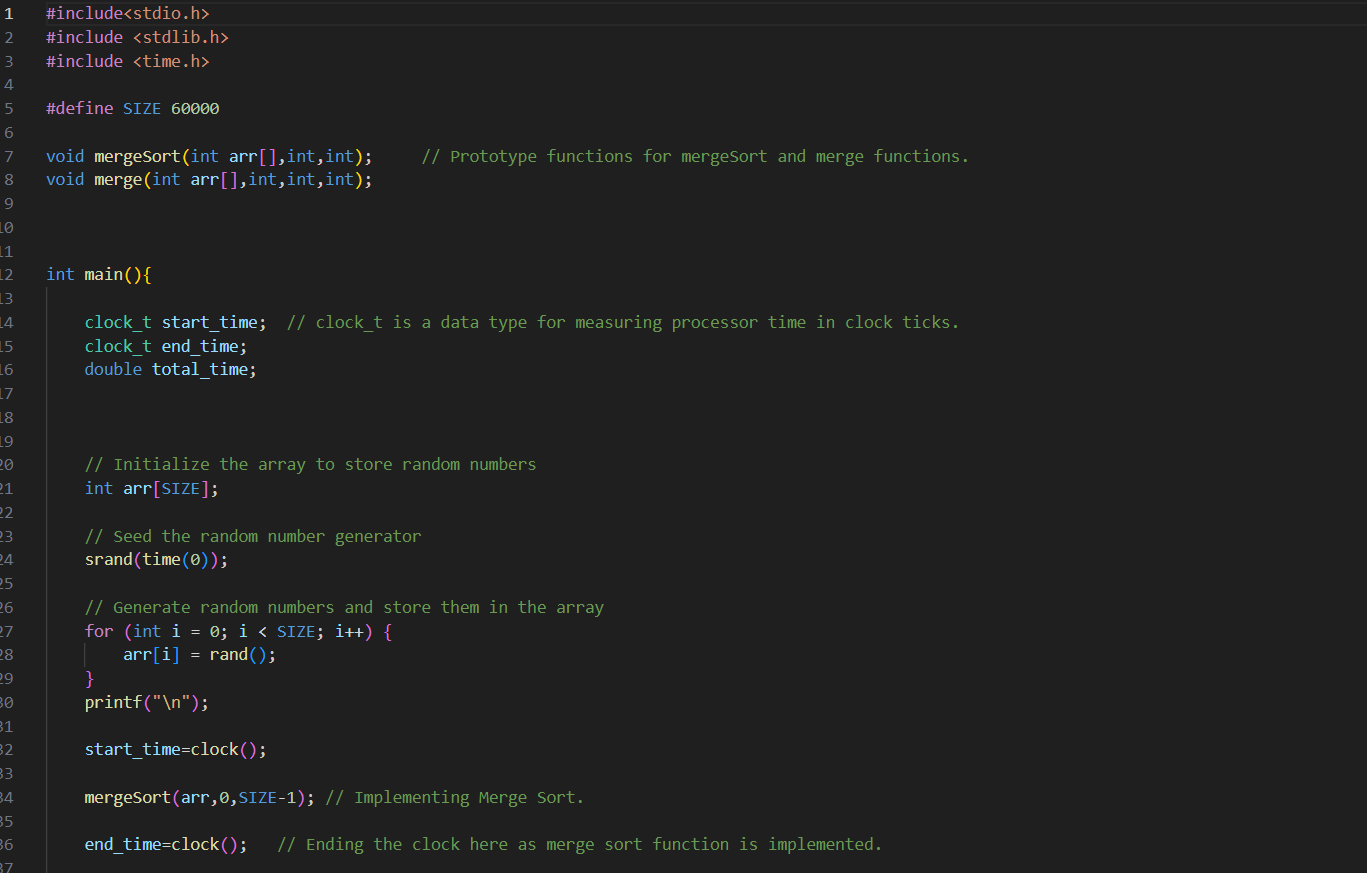
**GRAPH ANALYSIS:**

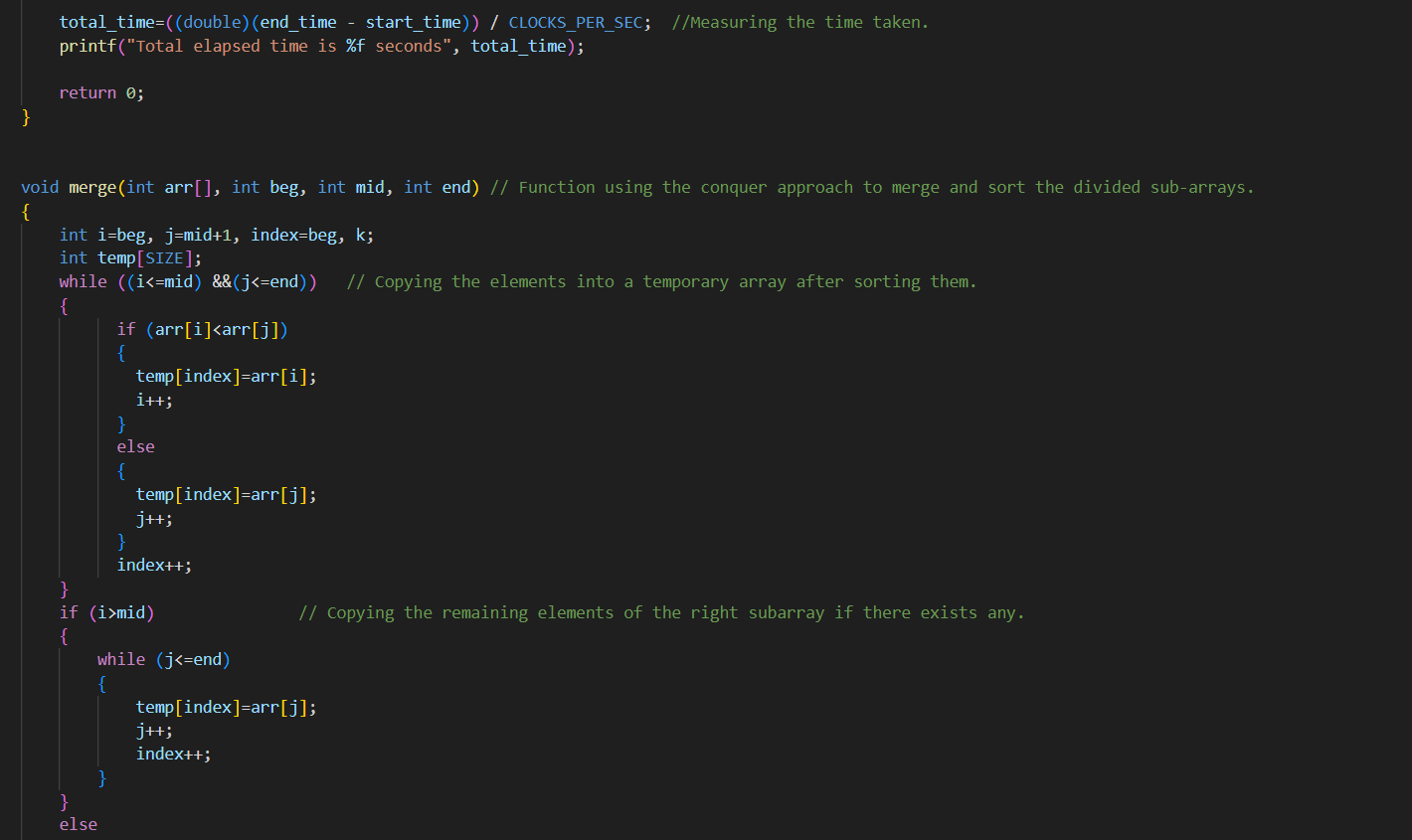
****

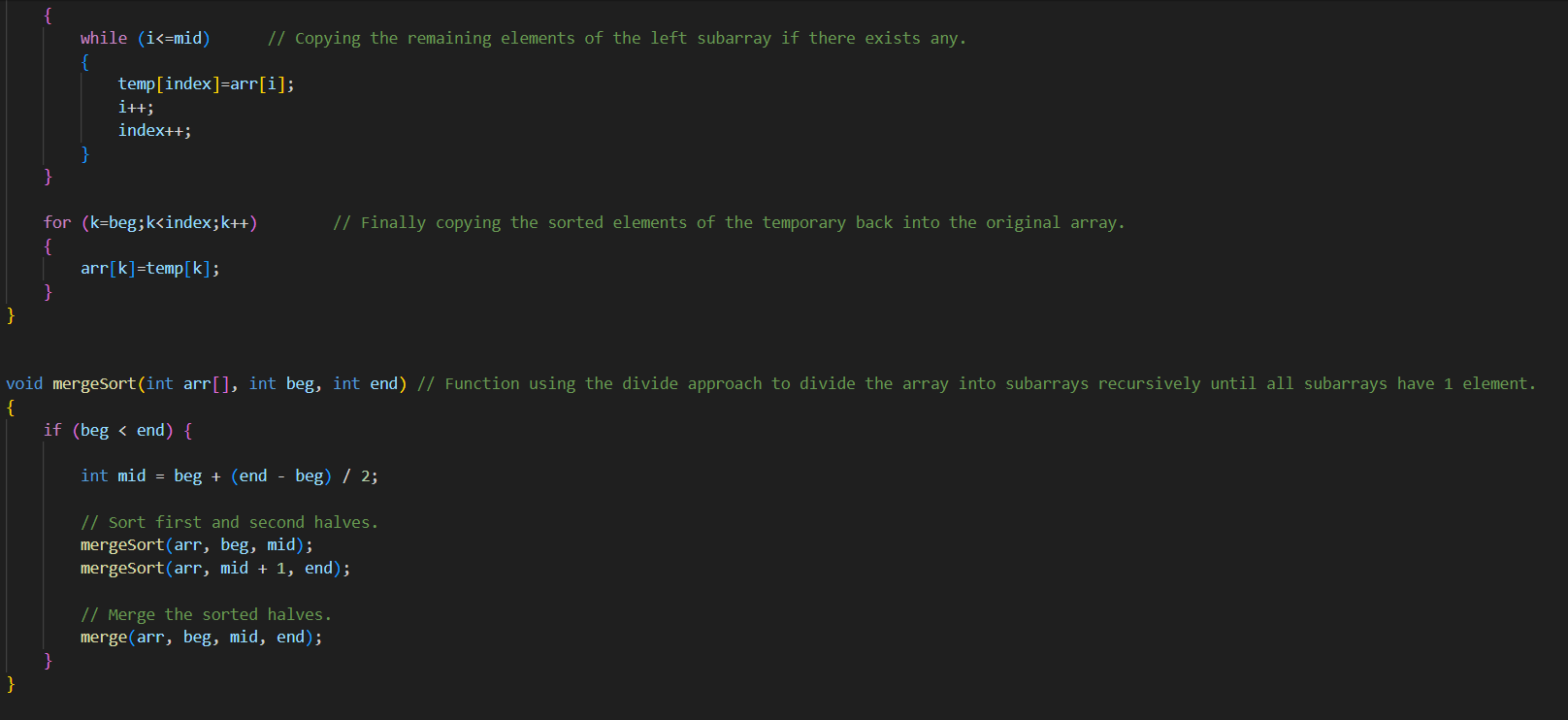
**EXPERIMENT 2**

**Implement divide and conquer based merge sort and quick sort algorithms and compare their performance for the same set of elements.**

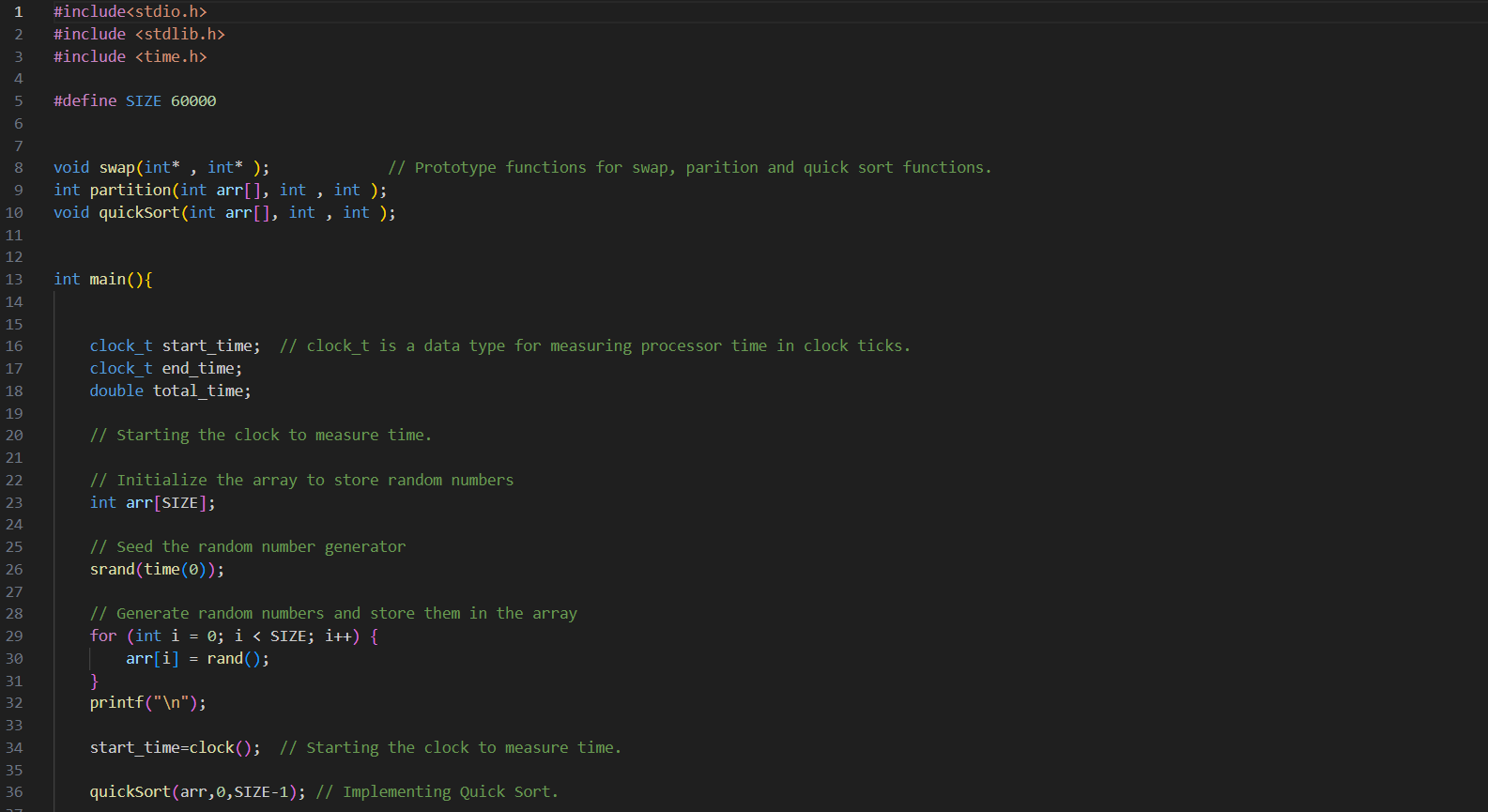
**CODE:  
MERGE SORT:**

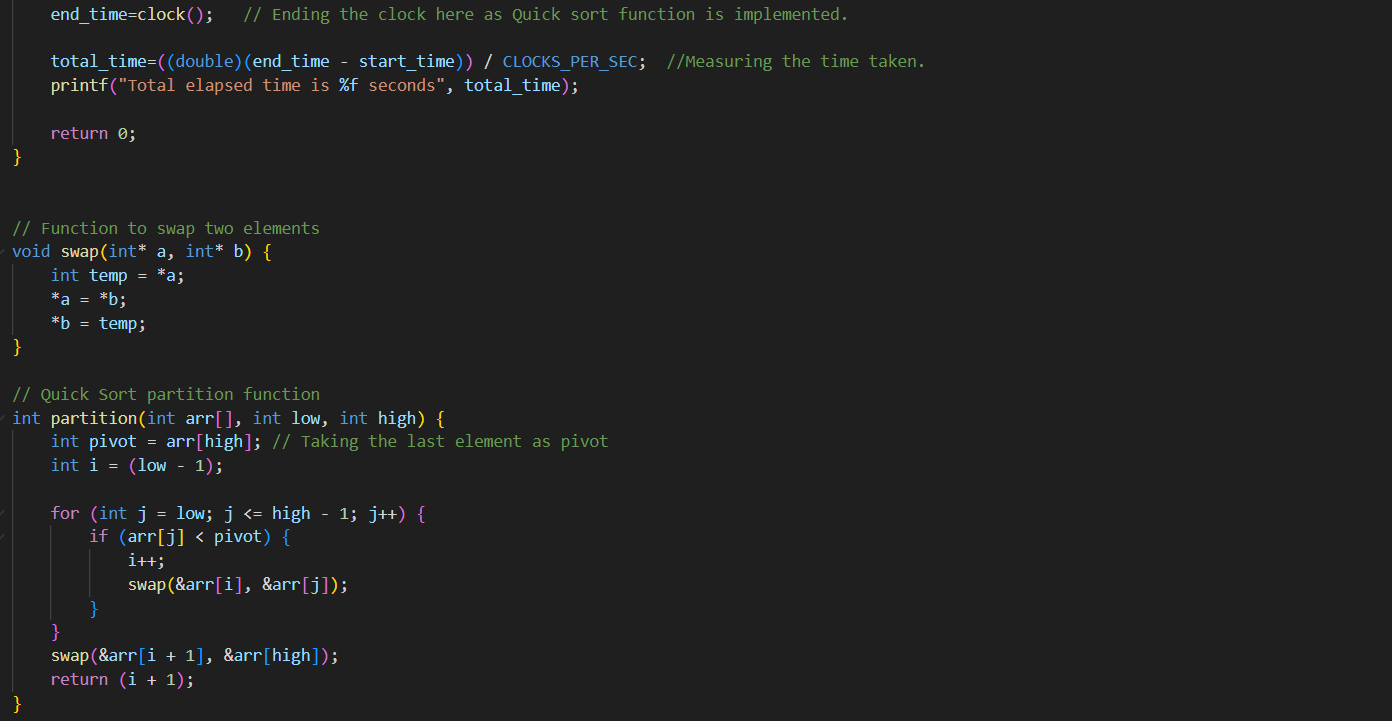
****

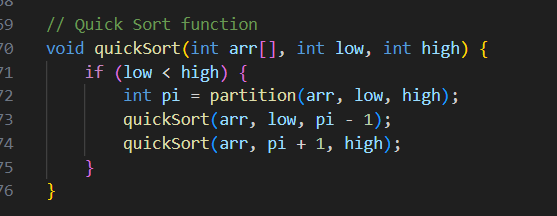
****

****

**QUICK SORT:**

****

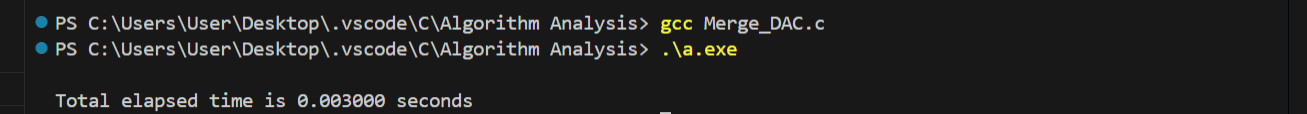
****

****

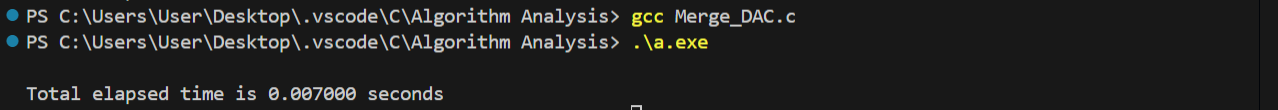
**OUTPUTS:**

**MERGE SORT:**

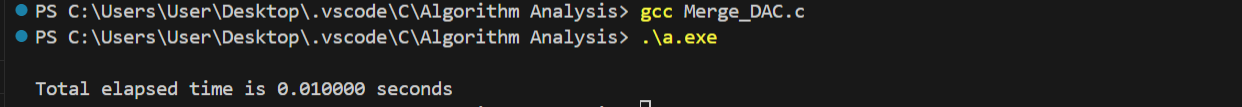
**1. N=10000**



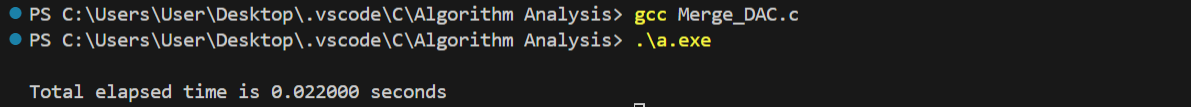
**2. N=20000**



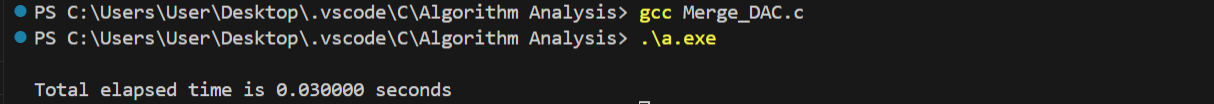
**3. N=30000**



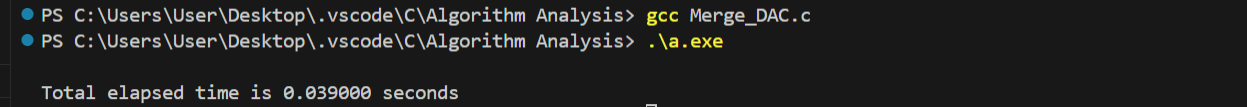
**4. N=40000**



**5. N=50000**

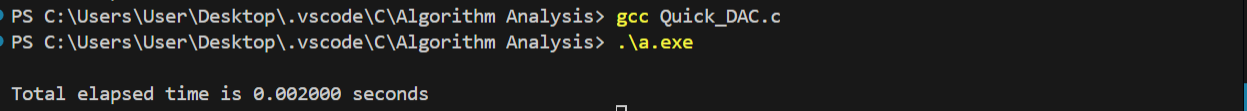


**6. N=60000**

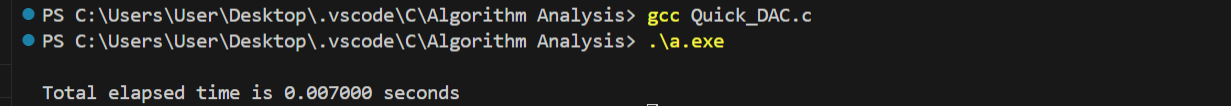


**QUICK SORT**

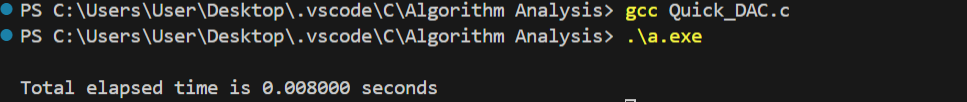
**1. N=10000**



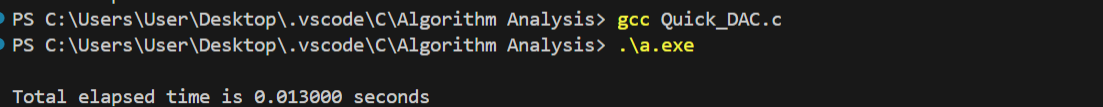
**2. N=20000**



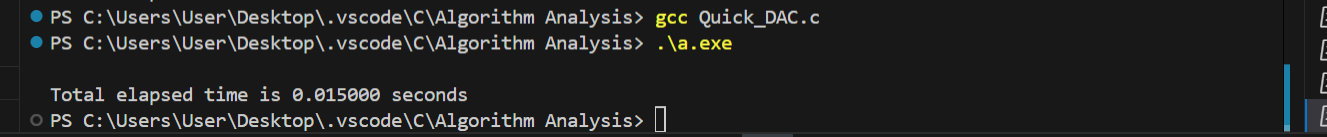
**3. N=30000**



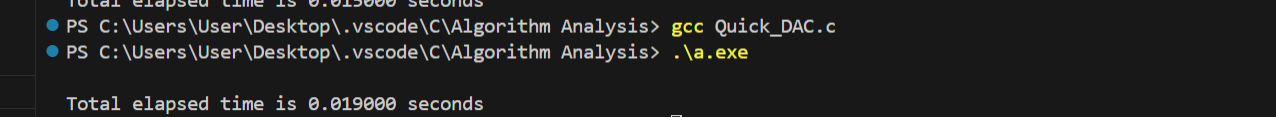
**4. N=40000**



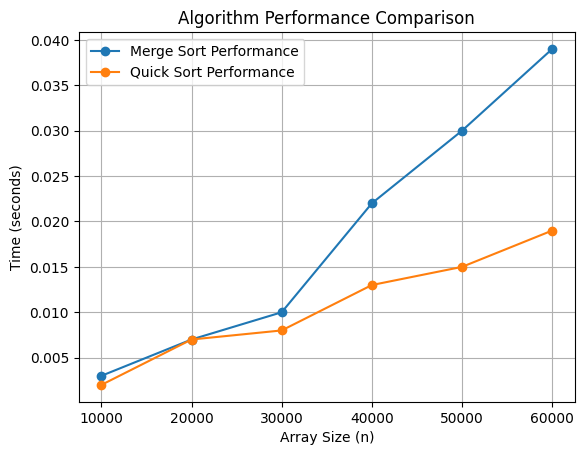
**5. N=50000**



**6. N=60000**



**GRAPH ANALYSIS:**

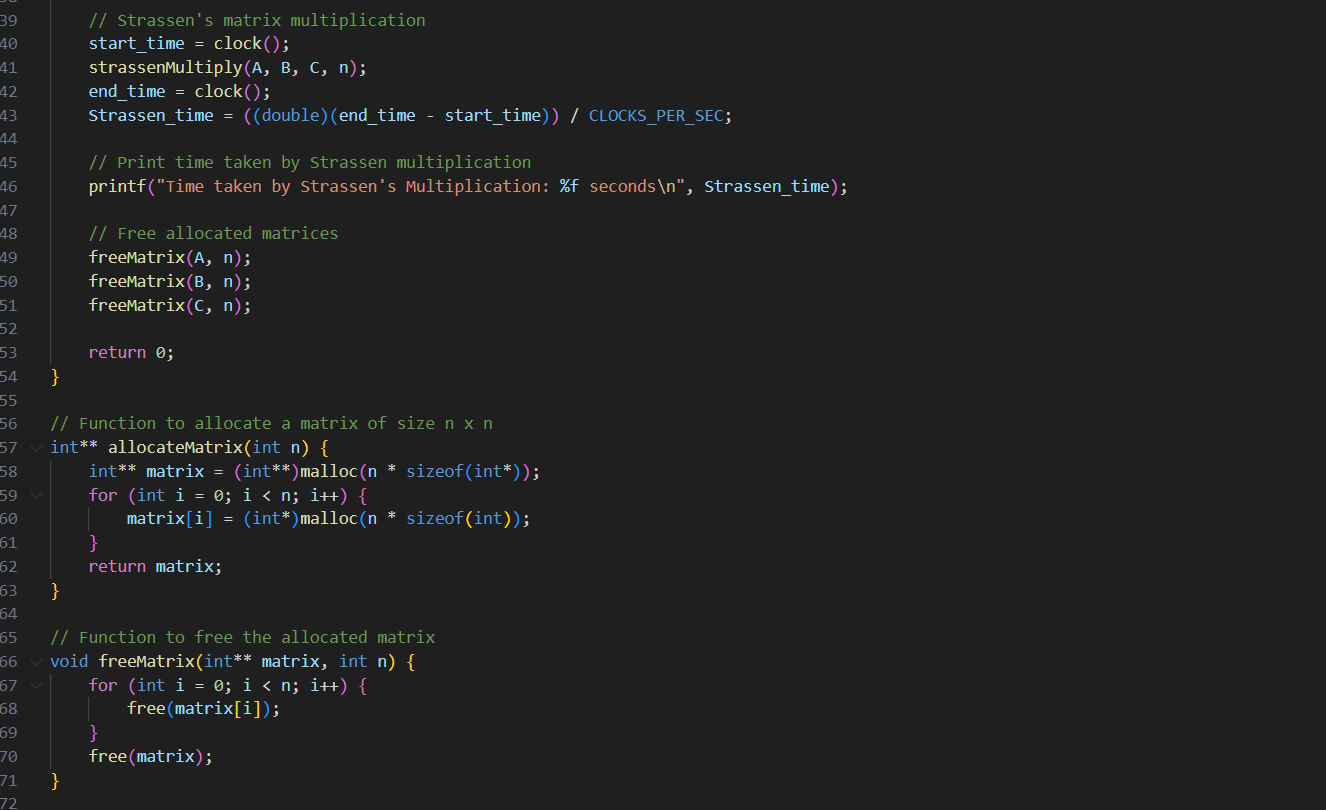
****

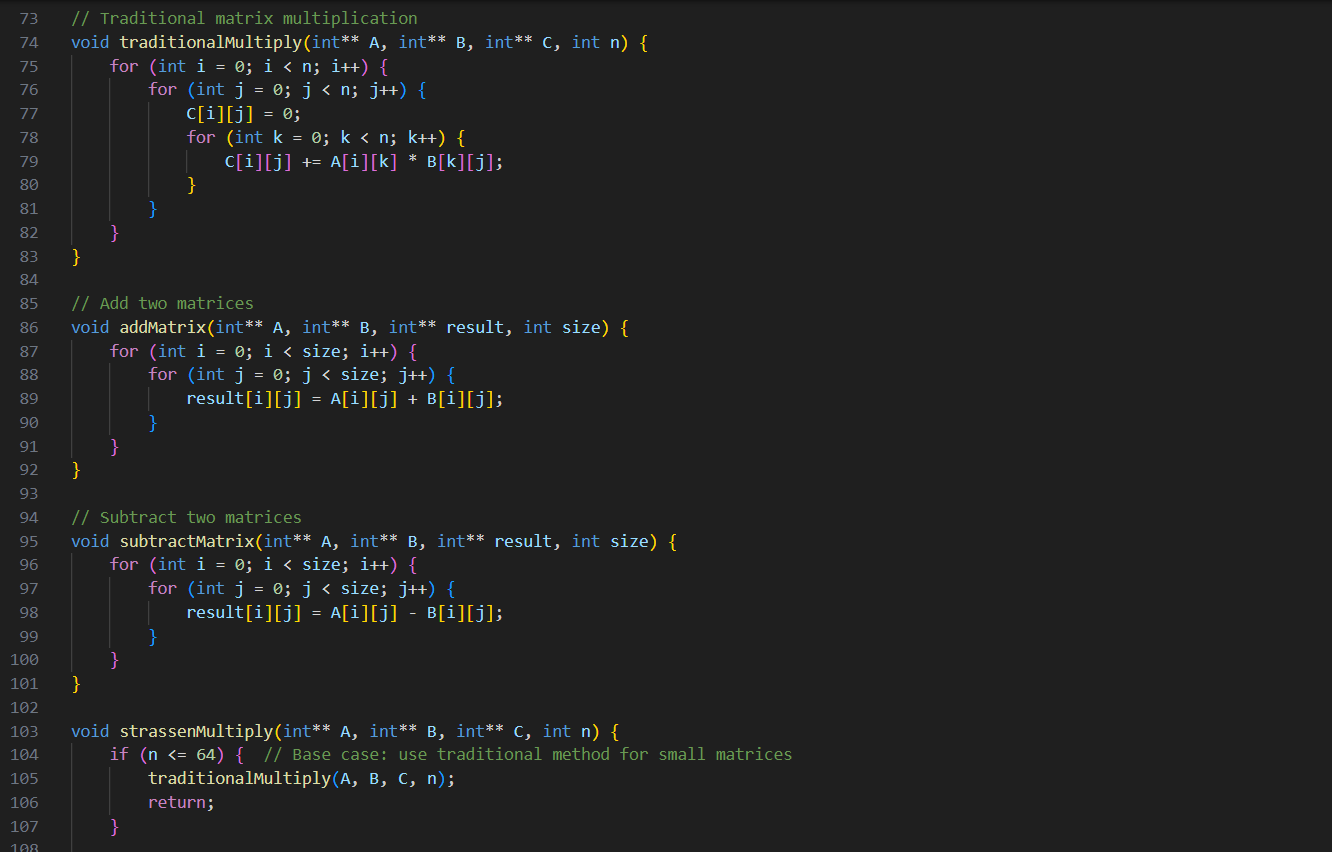
**EXPERIMENT 3**

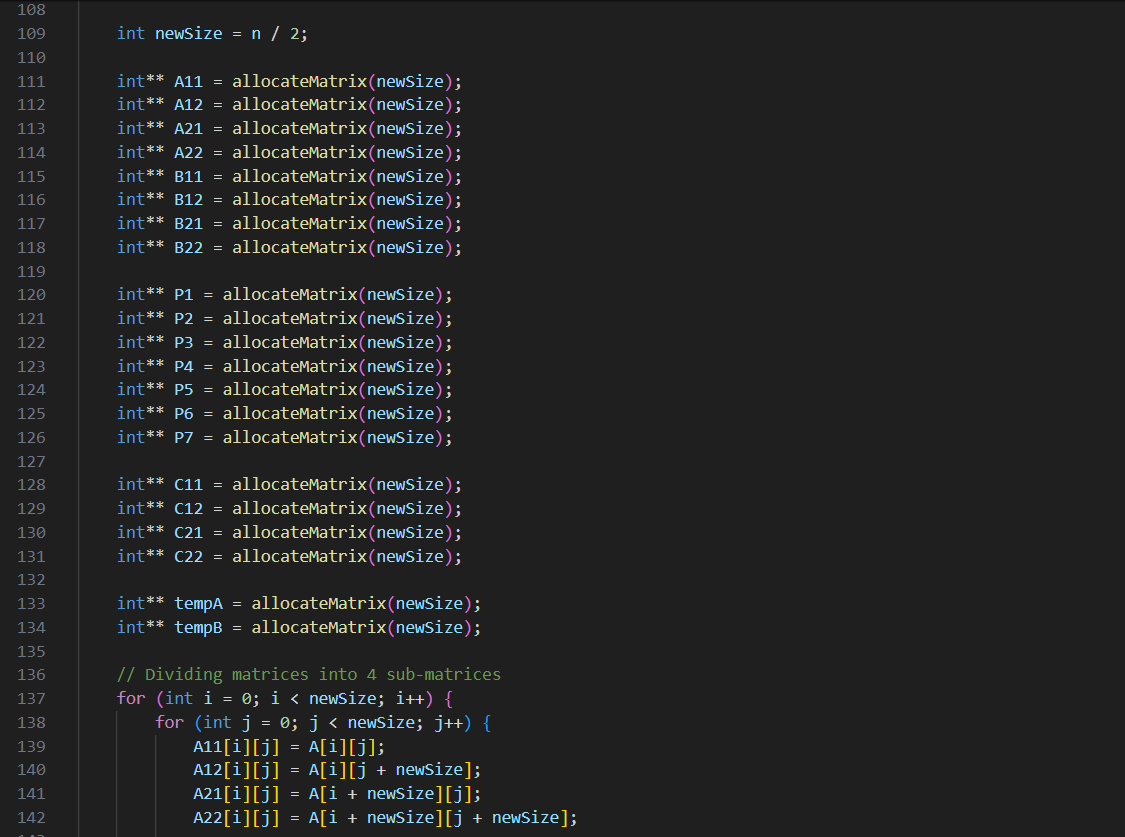
**Compare the performance of Strassen method of matrix multiplication with traditional way of matrix multiplication.**

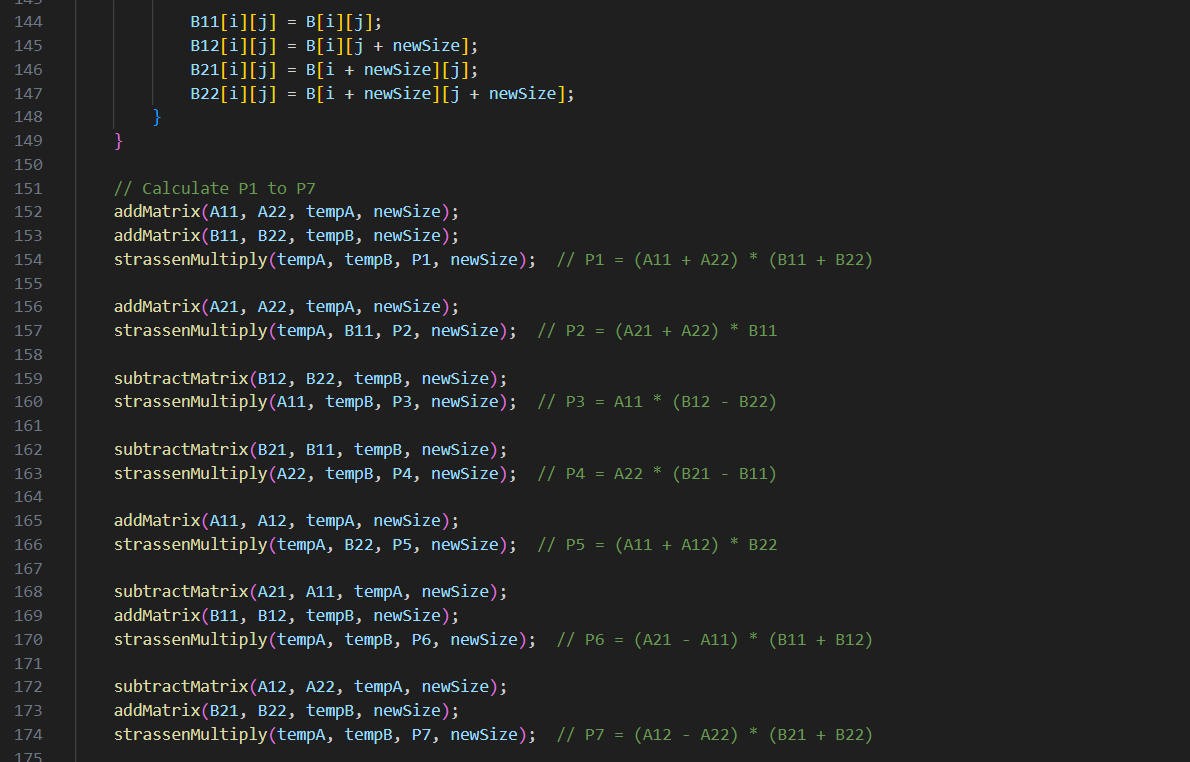
**CODE:**

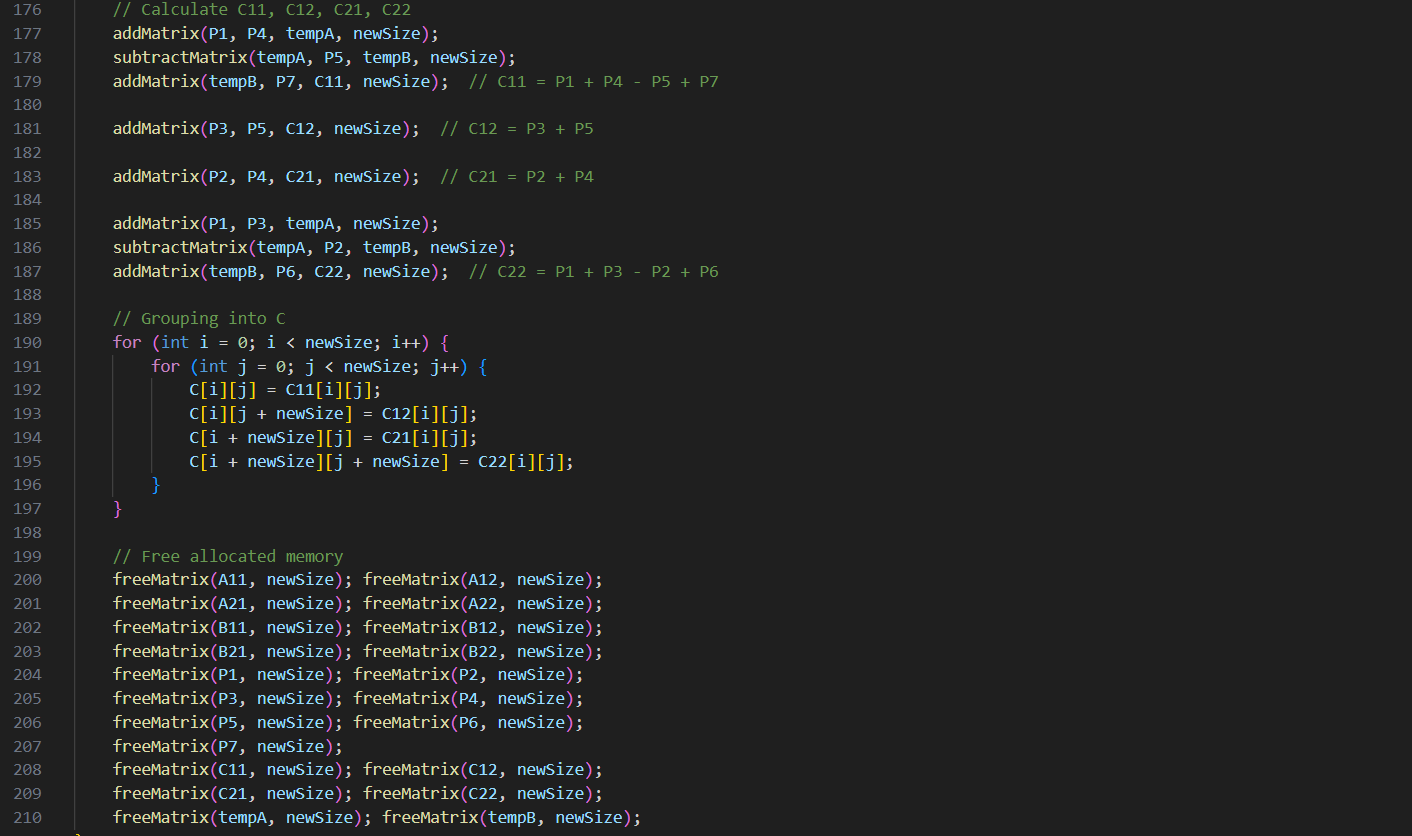
****

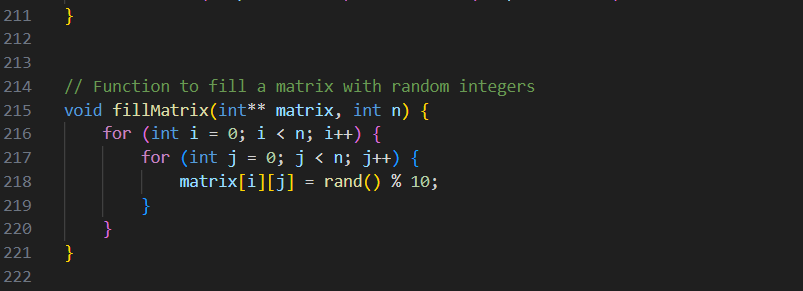
****

****

****

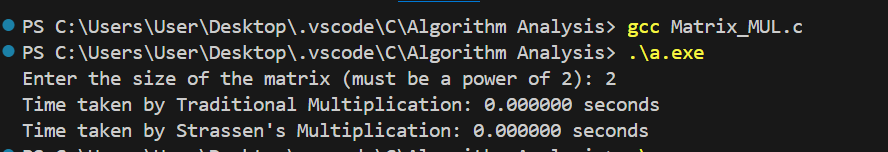
****

****

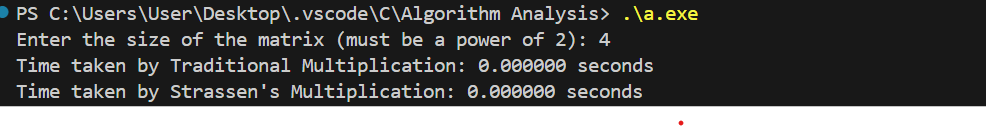
****

**OUTPUTS:**

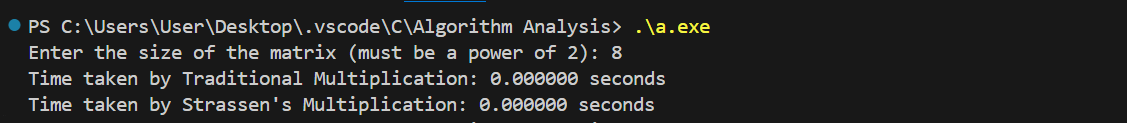
**1.**



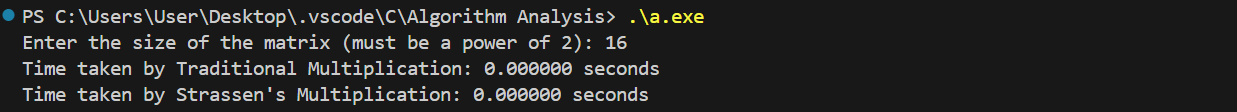
**2.**



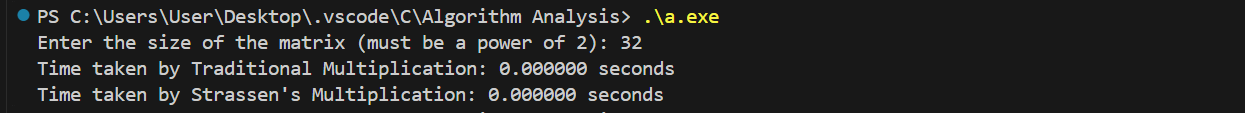
**3.**



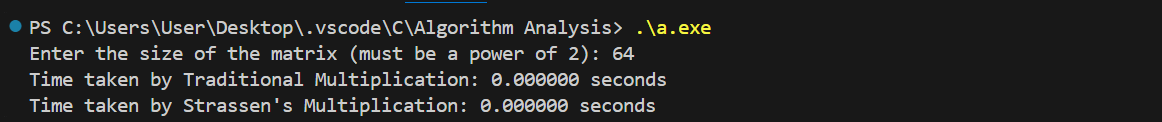
**4.**



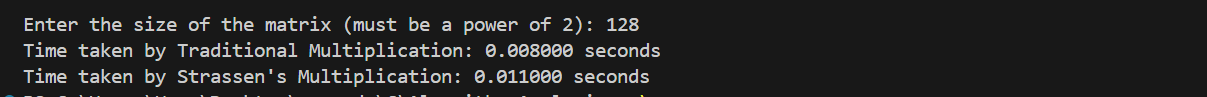
**5.**



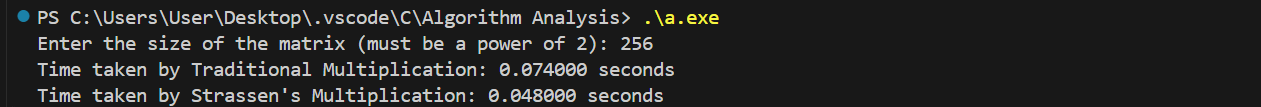
**6.**



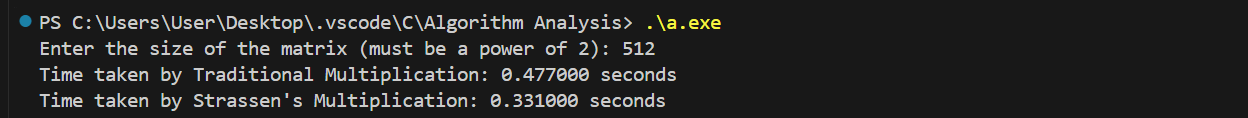
**7.**



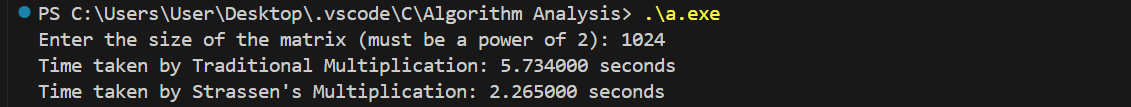
**8.**



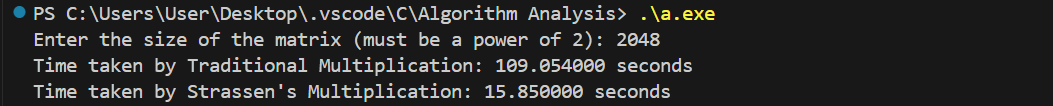
**9.**



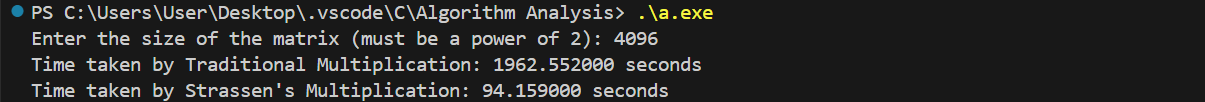
**10.**



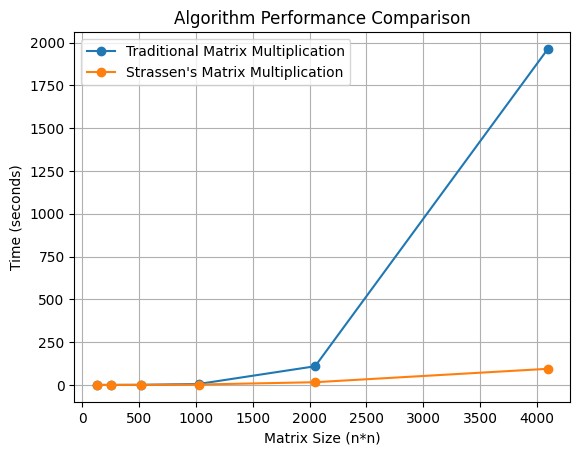
**11.**



**12.**



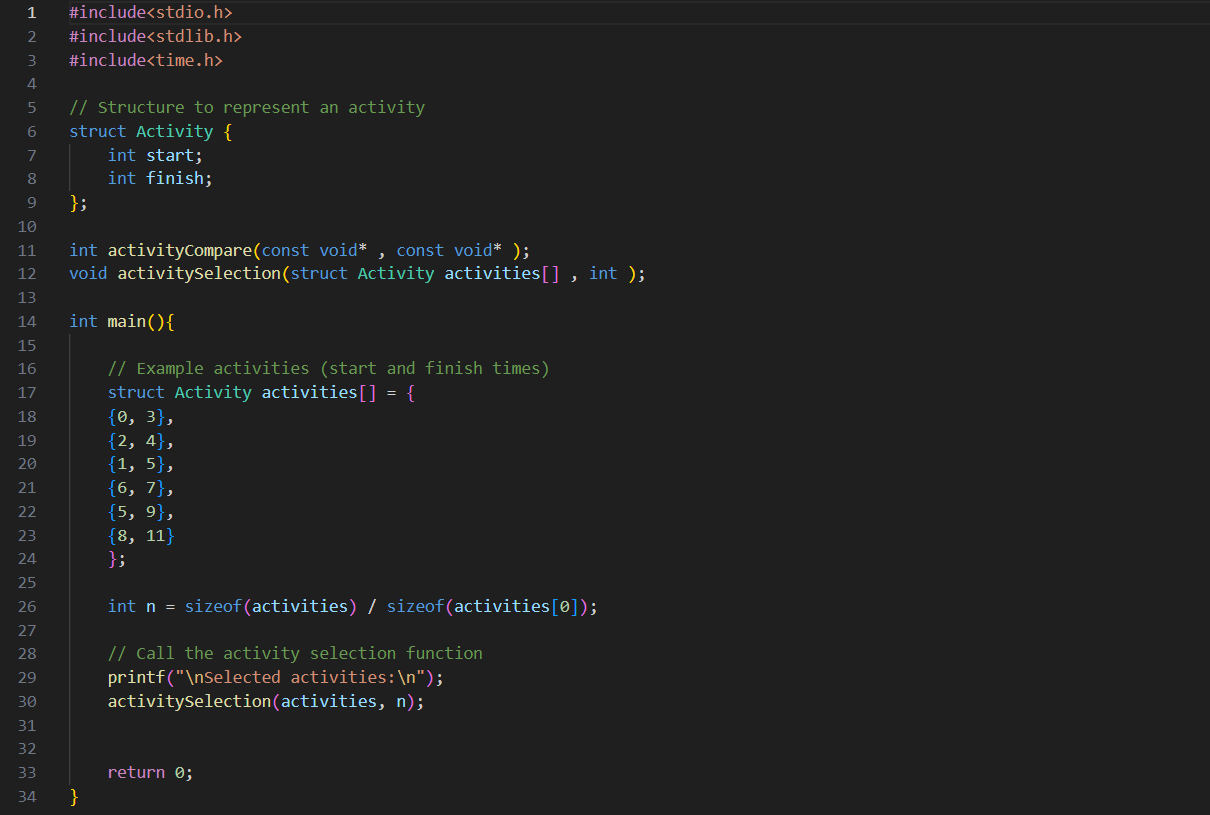
**GRAPH ANALYSIS:**

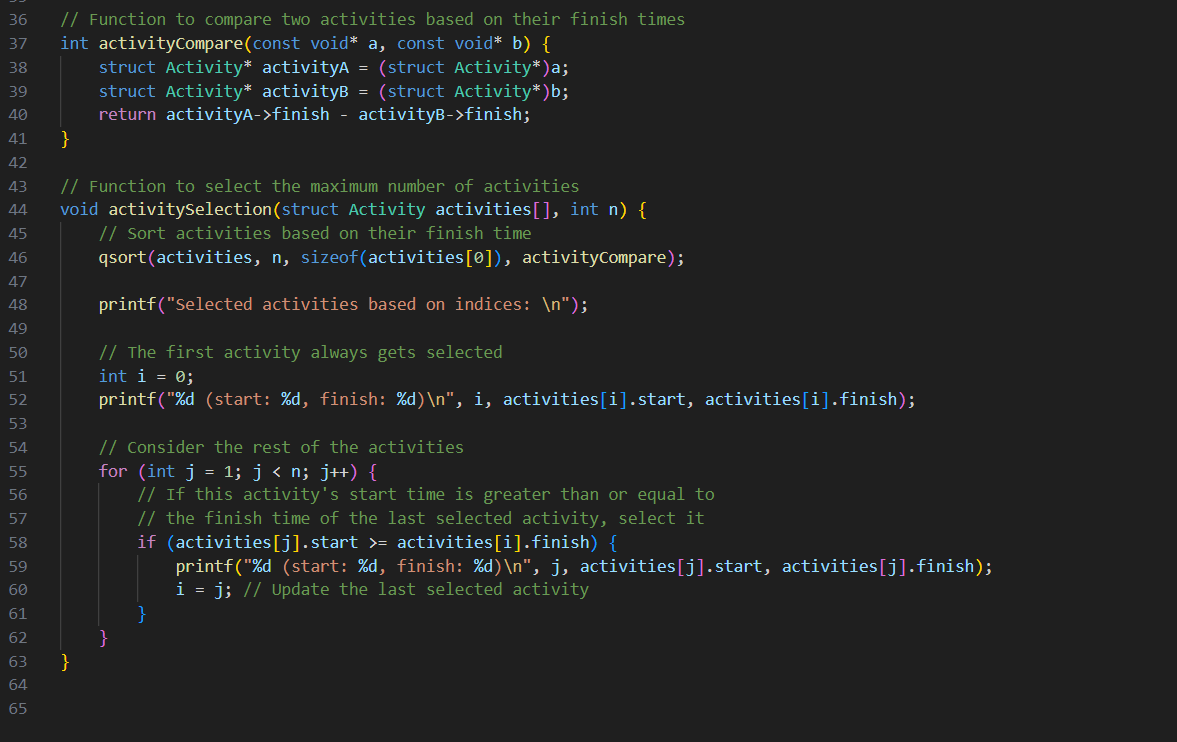
****

**EXPERIMENT 4**

**Implement the activity selection problem to get a clear understanding of greedy approach.**

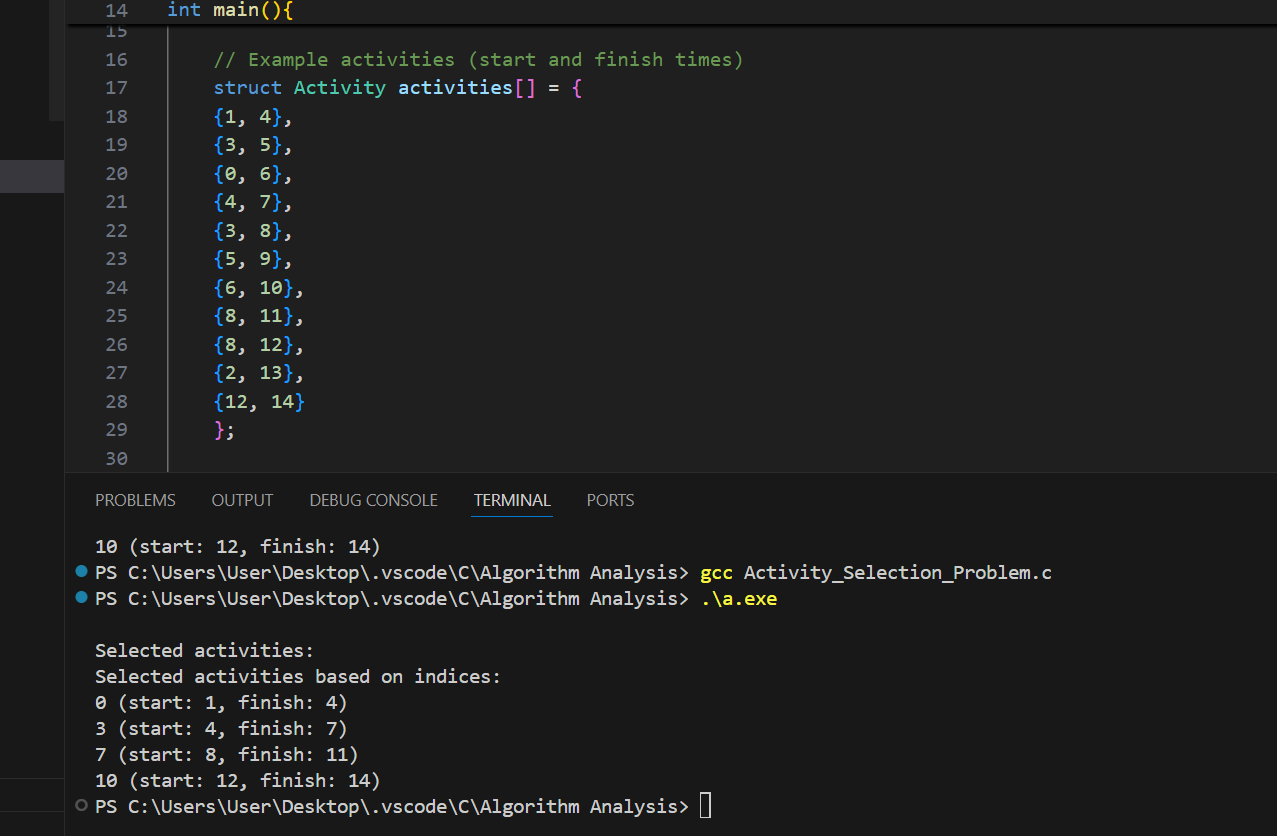
**CODE:**

****

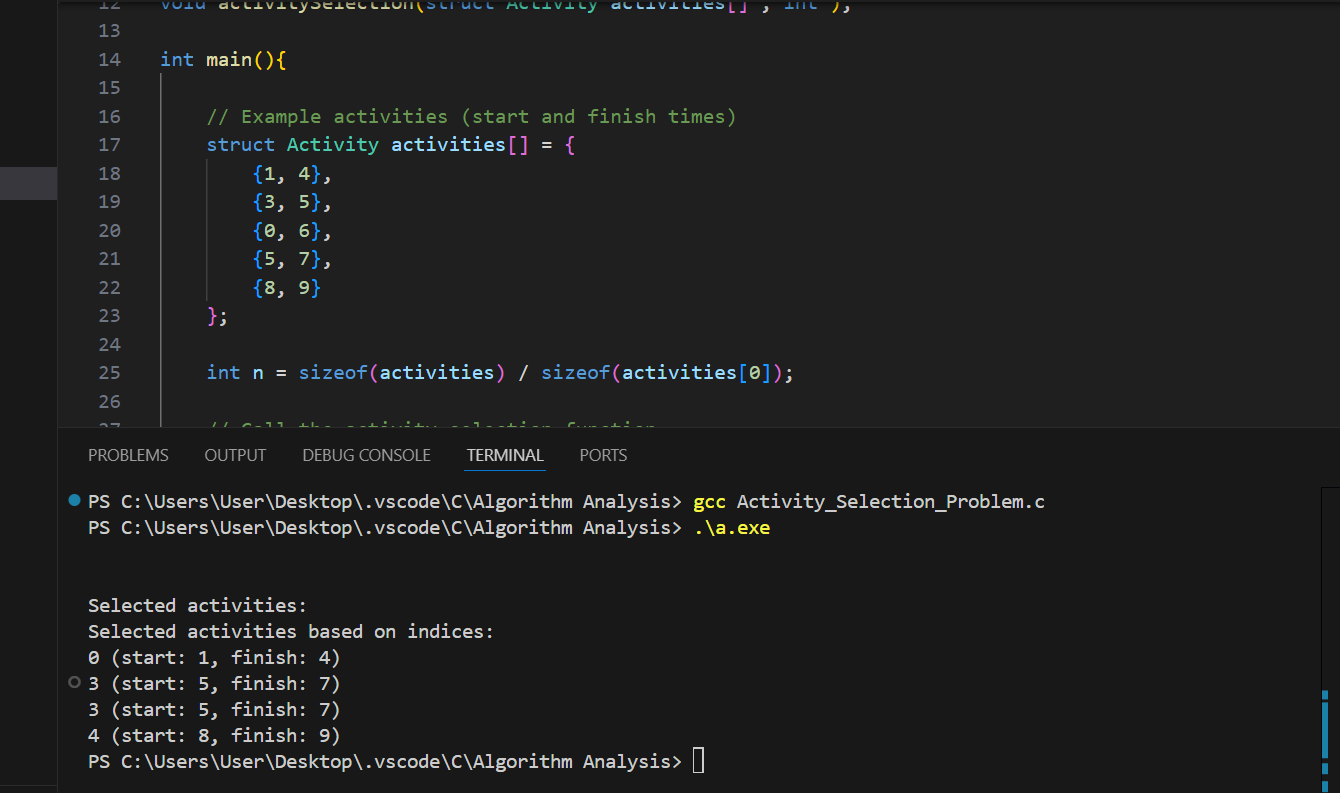
****

**OUTPUTS:**

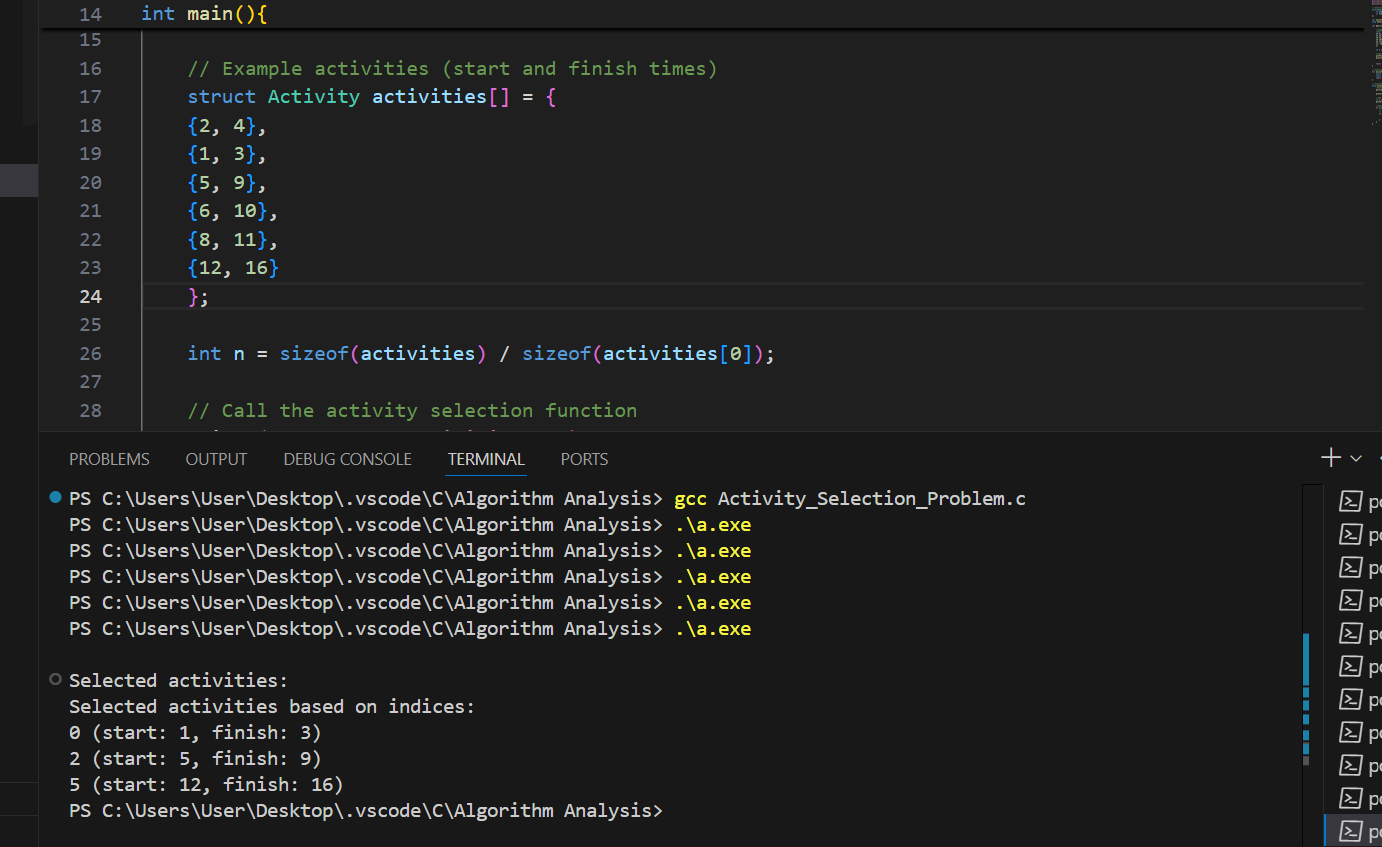
**1.**



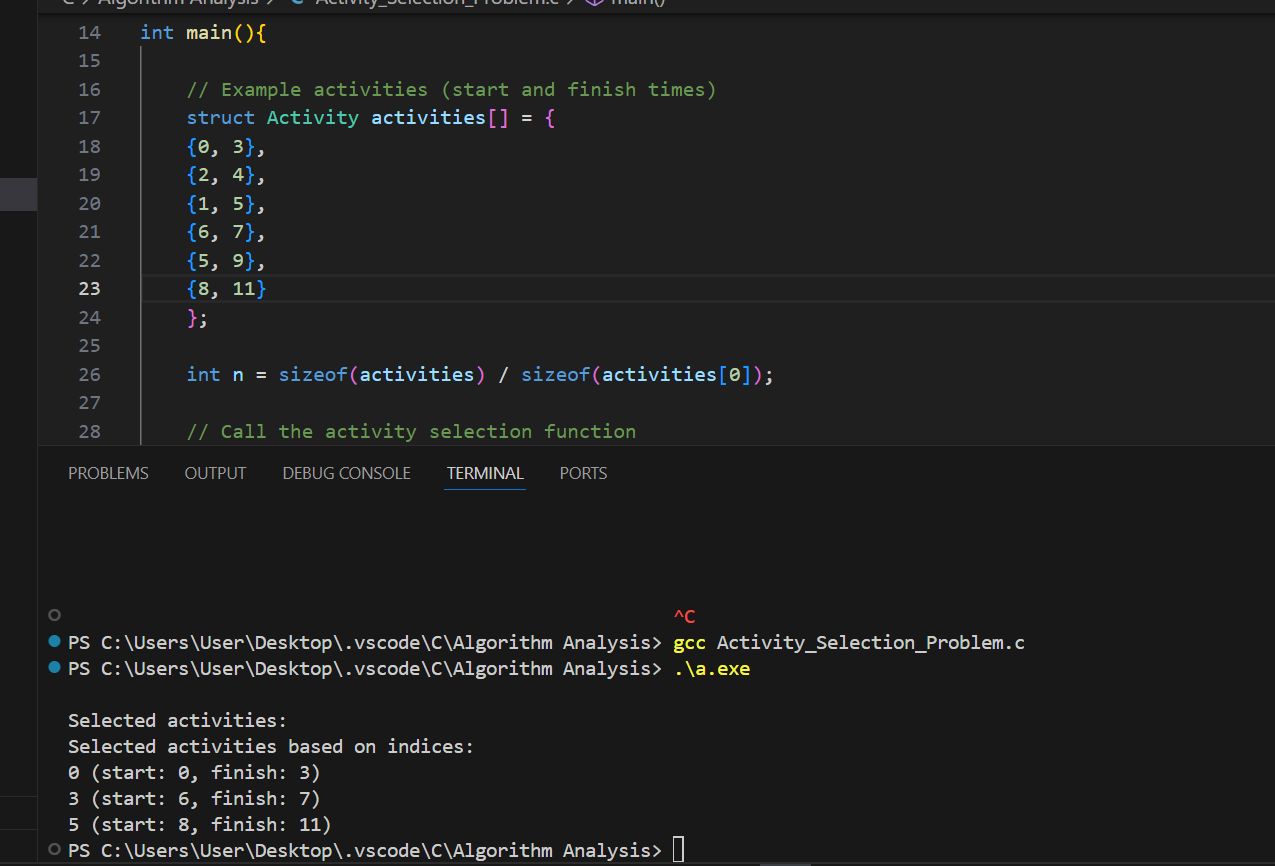
**2.**



**3.**

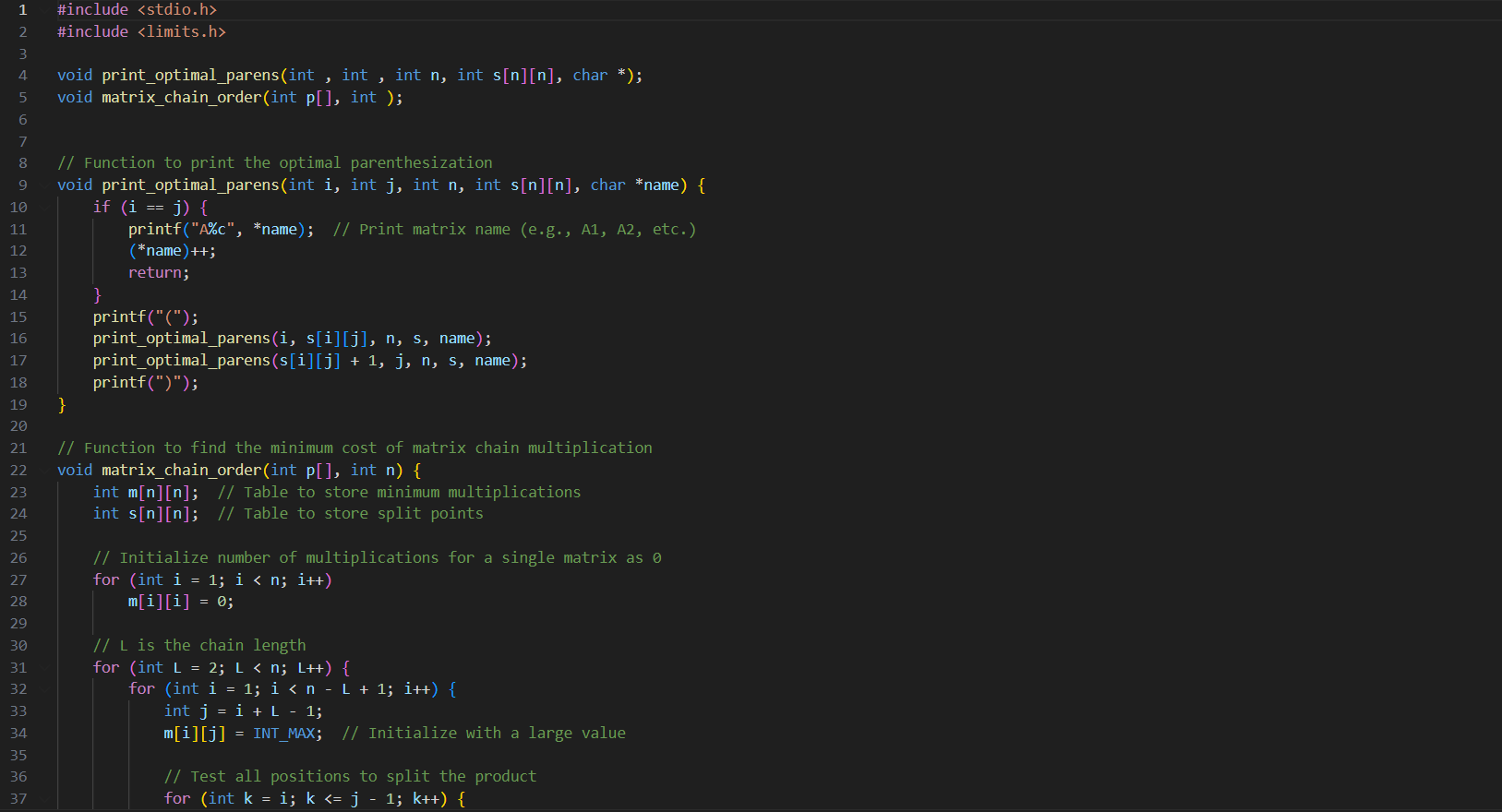


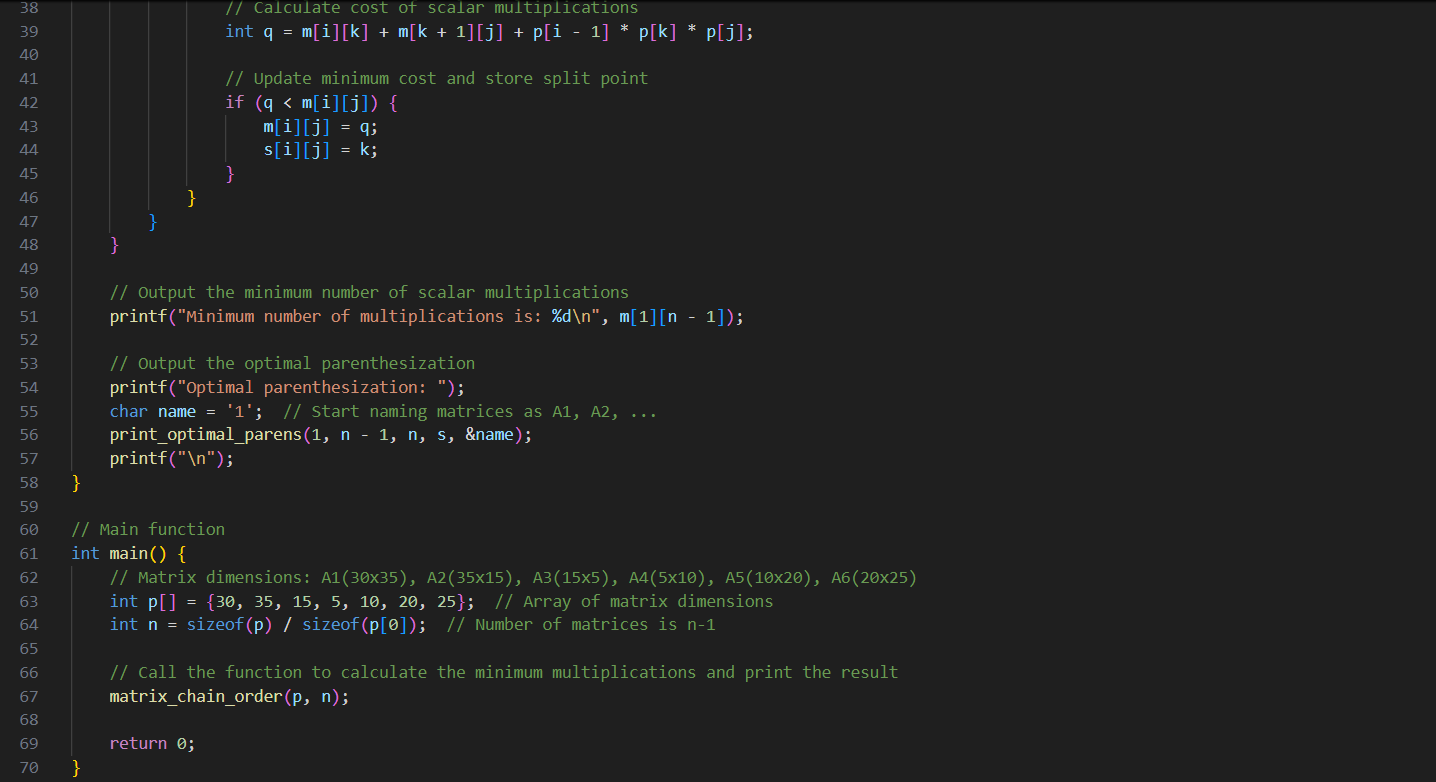
**4.**



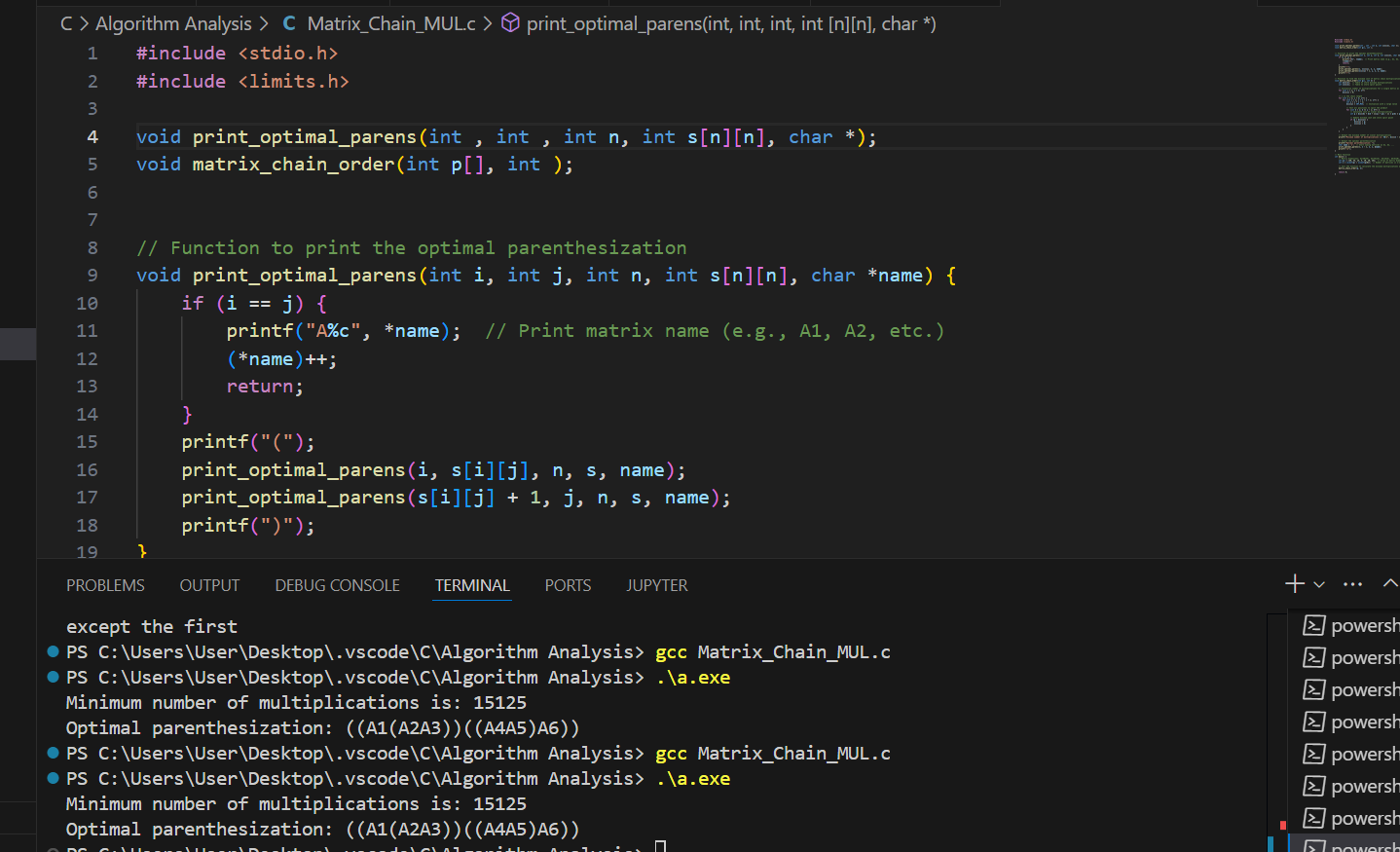
**EXPERIMENT 5**

**Get a detailed insight of dynamic programming approach by the implementation of Matrix Chain Multiplication problem and see the impact of parenthesis positioning on time requirements for matrix multiplication.**

**CODE:  
**

****

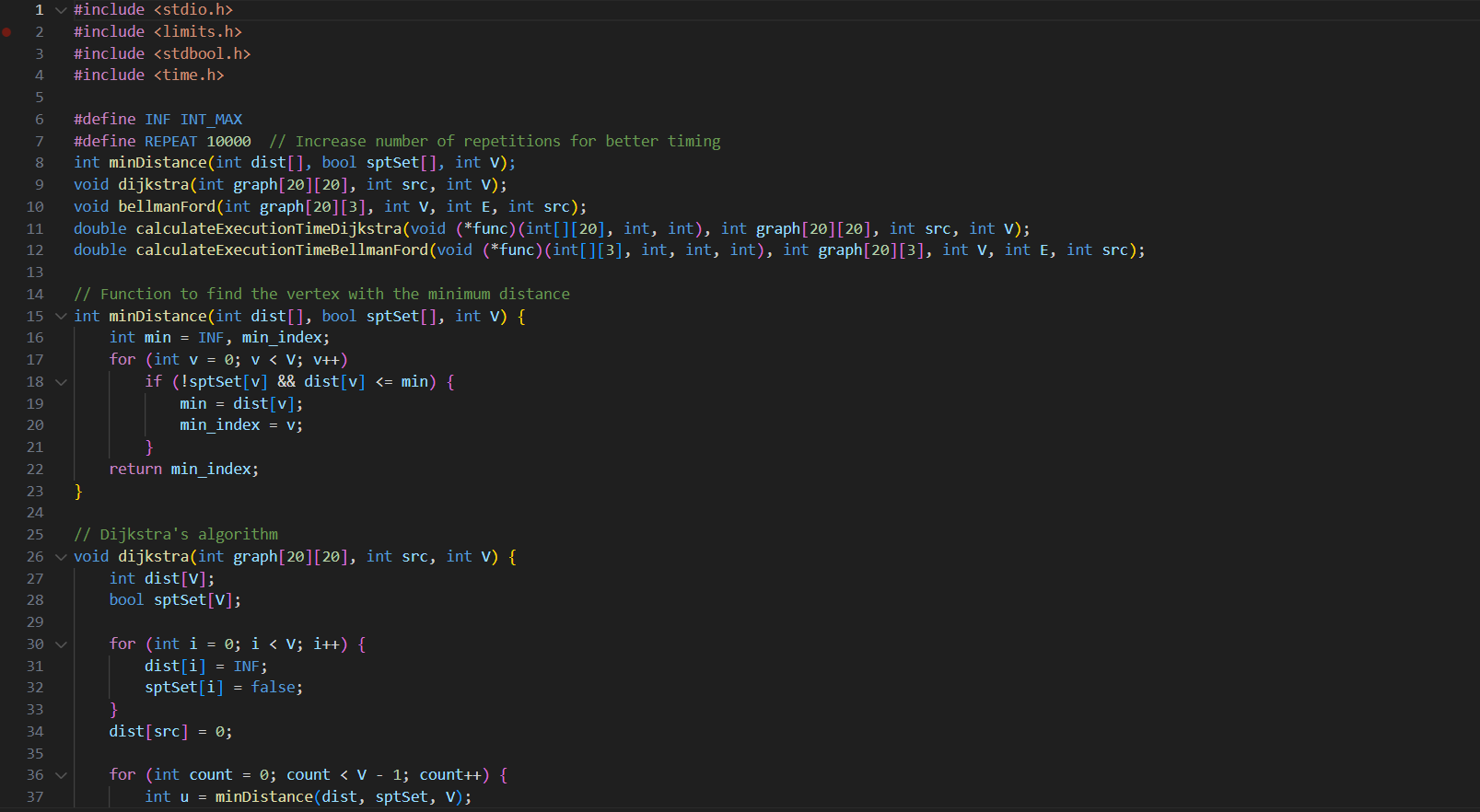
**OUTPUT:**

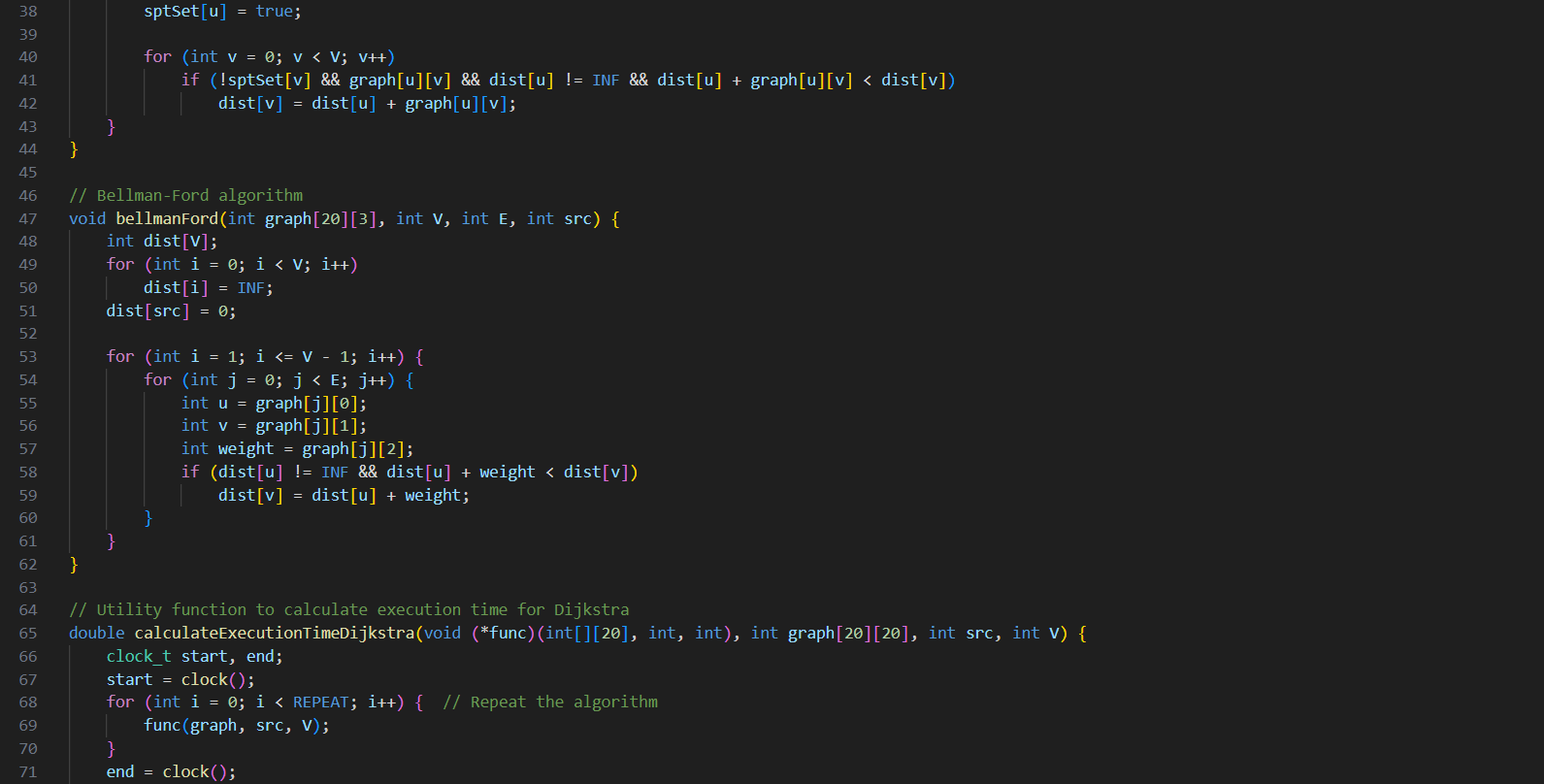


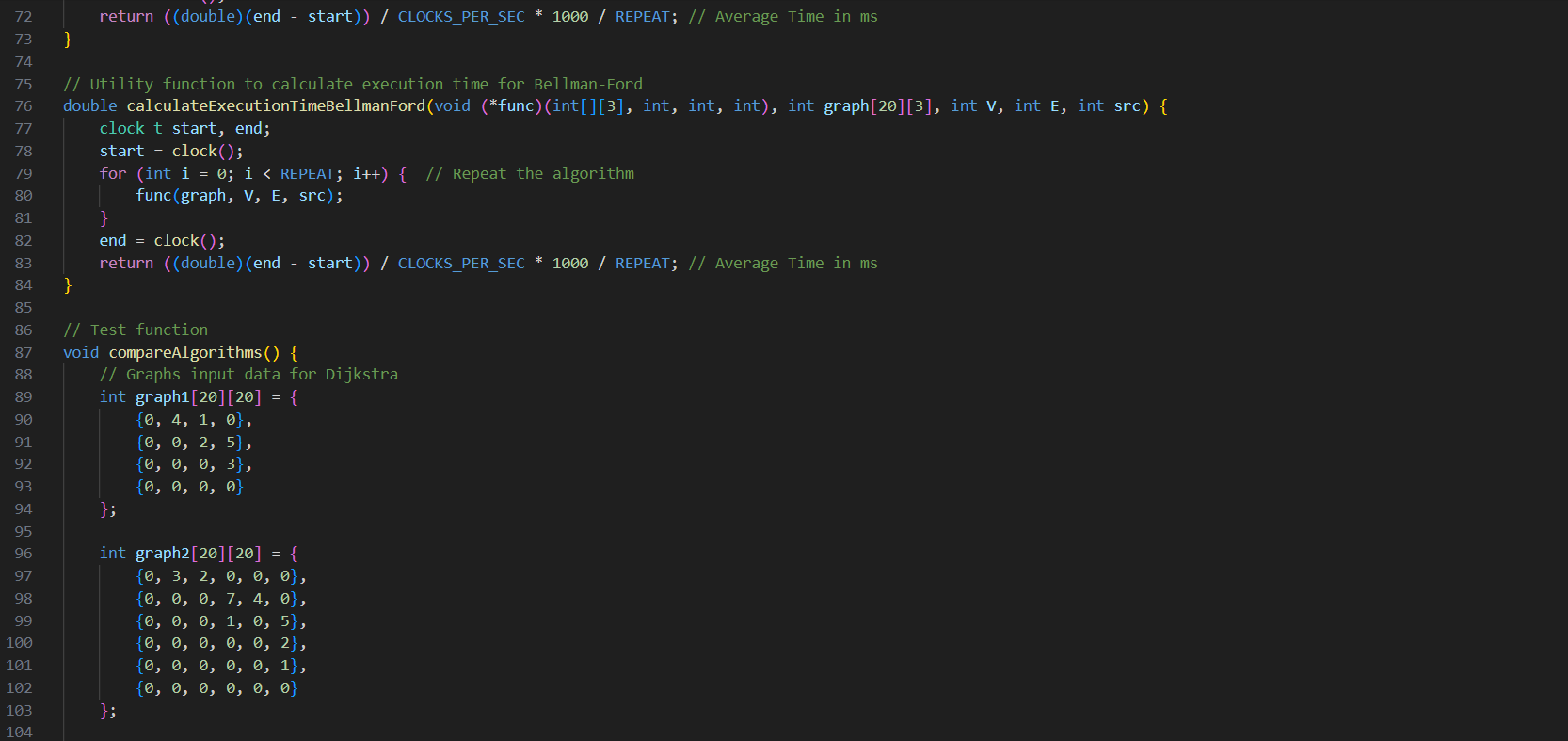
**EXPERIMENT 6**

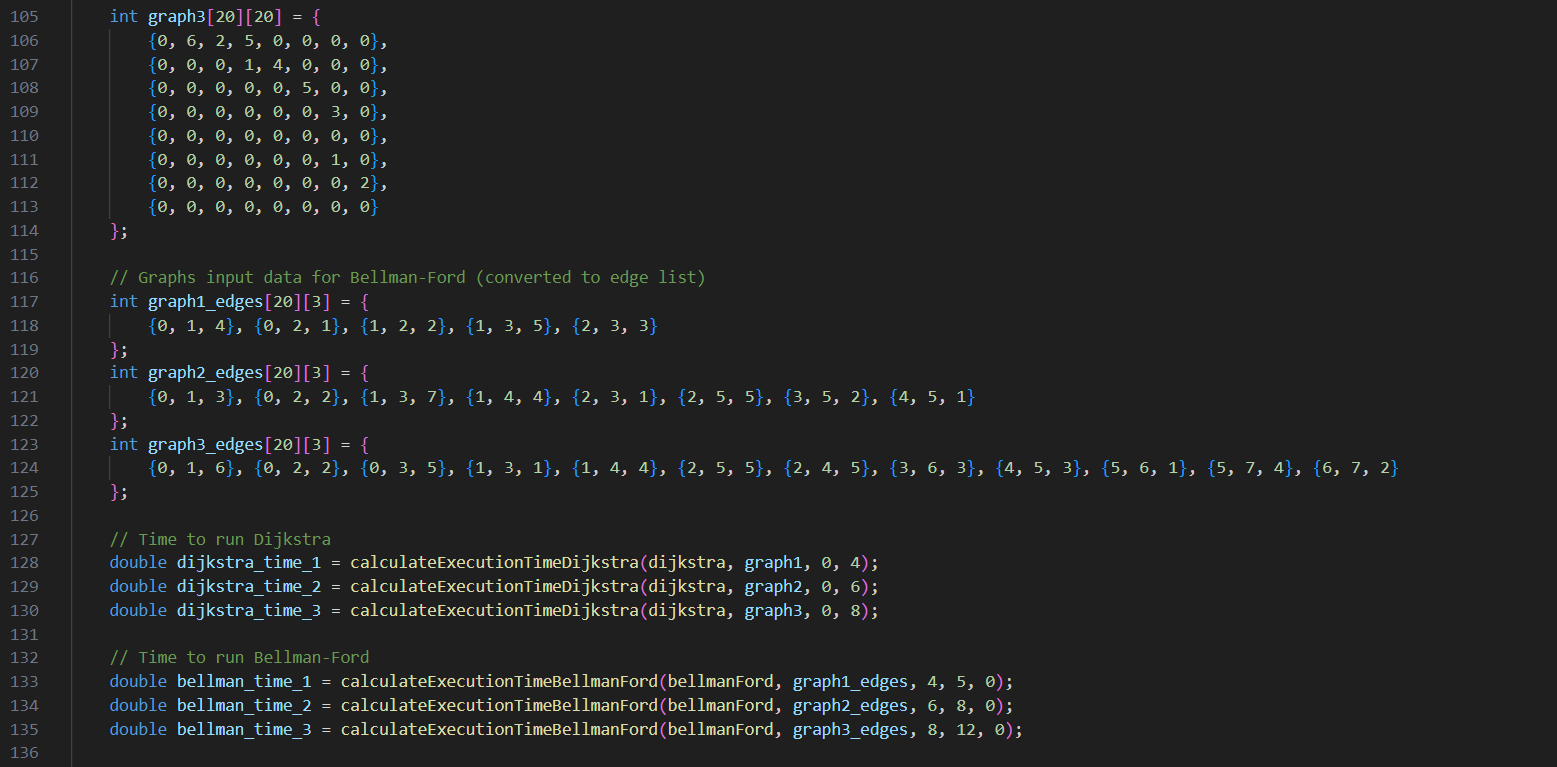
**Compare the performance of Dijkstra and Bellman ford algorithm for the single source shortest path problem.**

**CODE:**

****

****

****

****

**OUTPUT:**

**GRAPH ANALYSIS:**

**EXPERIMENT 7**

**Through 0/1 Knapsack problem, analyse the greedy and dynamic programming approach for the same dataset.**

**CODE:**

**OUTPUTS:  
1.**

**2.**

**3.**

**4.**

**5.**

**EXPERIMENT 8**

**Implement the sum of subset.**

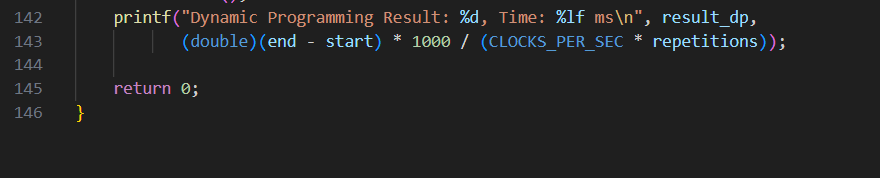
**CODE:**

**OUTPUT:**

**EXPERIMENT 9**

**Compare the Backtracking and Branch & Bound Approach by the implementation of 0/1 Knapsack problem. Also compare the performance with dynamic programming approach.**

**CODE:**

****

**OUTPUTS:**

**1.**

**2.**

**3.**

**GRAPH ANALYSIS:**

**EXPERIMENT 10**

**Compare the performance of Rabin-Karp, Knuth-Morris-Pratt and naive string-matching algorithms.**

**CODE:**

**OUTPUTS:**

**1.**

**2.**

**3.**

**4.**

**5.**

**GRAPH ANALYSIS:**

**MY GITHUB LINK:**

https://github.com/COBR-A/Algorithm\_lab\_3\_sem\_500120575