Valgrind:

```
ab@gram:/mnt/c/Users/Alex$ valgrind --version
valgrind-3.22.0
```

Algorithm Analysis:

1. Constant Time – Array Middle Element: Write a C++ function that takes an array of integers and returns the middle element. Analyze the time complexity and discuss why it is O(1). Ensure that the plot aligns with constant time.

Because the algorithm only executes its commands once, despite the size of the array, the resources used will remain the same each time. This makes the algorithm O(1).

2. Linear Time – Array Range: Implement a function to find the range of all elements in an array of integers using a loop. Analyze its time complexity and discuss why it is O(n). Ensure that the plot aligns with linear time.

This algorithm checks each element in an array, making the number of steps required increase as the size of the problem increases. The complexity is O(n) because the relationship between the number of steps and the size of the problem is linear.

3. Linear Time - Exponential Function: Implement an iterative function to calculate the result of raising a number to a power (e.g., x^n). Do not use any library exponentiation, use brute force iterative multiplication. Analyze and discuss its time complexity. Ensure that the plot aligns with your conclusions.

The power of N executes an operation for the total value of N, meaning that as N increases the number of operations increases. This is a linear complexity, O(n)

4. Quadratic Time – Square Matrix Max Element: Write a function to find the maximum element in a square matrix (number of rows == number of columns) of integers. The function should accept a reference to the matrix and the size N. N will represent both the number oif columns and number of rows. Analyze its time complexity, and discuss in detail why it is O(n^2). Ensure that the plot aligns with quadratic time.

This executes statements for each row and column, meaning that the number of executions is N*N. This is a quadratic complexity, or $O(N^2)$

5. Linear Time - Factorials: Write an iterative function to calculate the factorials of a number, not including 1 and itself. It must be O(n). Analyze its time complexity and discuss why it is O(n). Ensure that the plot aligns with linear time.

Another linear algorithm as it also runs a set of statements N times, making it O(n).

6. Linear (or Better?) - Prime Numbers: Using the concept of "trial division" write a function that implements an iterative solution to determine if any number N is prime. Analyze and discuss the time complexity of this approach. Implement the strategy that cuts trial division off at the square root of N. Analyze and discuss the time complexity of this approach. Graph both.

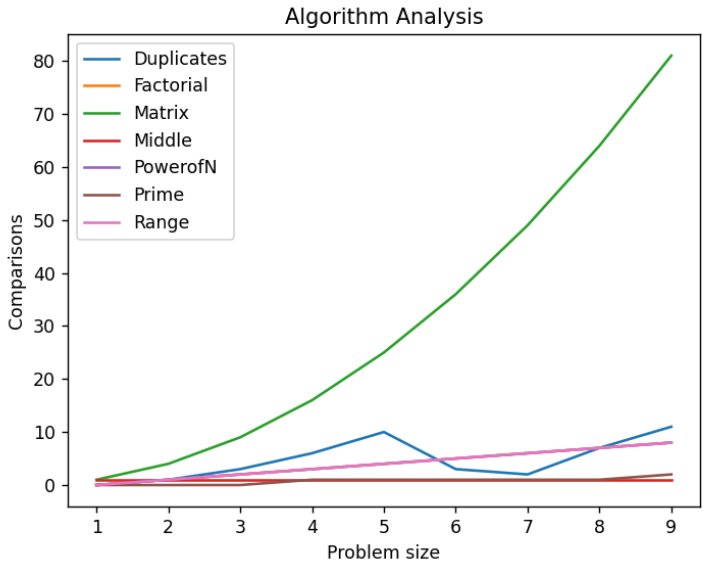
Determining a prime number has a complexity of O(sqrt(n)). This is because it only needs to test numbers up to the square root of N, as any number after that would not be a divisor. This makes it more efficient than a linear relationship.

7. Quadratic Time – Array has duplicates?: Write a boolean function that determines if an array contains any duplicate items. Provide a detailed analysis of the time complexity and discuss why it is O(n^2). Ensure that the plot aligns with quadratic time.

While the best case for this algorithm is one iteration, the worst case is N^2 iterations. The graph of the function performance would show this if the data set had no duplicates. As it is now, the line is not a consistent quadratic function, but the complexity of the algorithm is $O(n^2)$

Graph:s

LIMIT = 10



LIMIT = 1000