Testing Setup and Methodology:

1. **Set-up:**
    a. The testing environment consists of a computer with sufficient memory and processing power to run the required memory allocation routines.
    b. The memory allocation routines (Mem_alloc, Mem_free, Mem_stats, and Mem_print) are implemented as instructed.
    c. The search policy (FIRST_FIT, BEST_FIT, or WORST_FIT), search location (HEAD_FIRST or ROVER_FIRST), and coalescing options are configurable via global variables.
    d. Memory allocation and deallocation operations are performed using a variety of allocation patterns and sizes.
2. **Methodology:**
    a. **Test Cases**: Multiple test cases are created to evaluate the performance of the memory allocation routines under various conditions.
    b. **Allocation Patterns**: Different allocation patterns, such as sequential, random, and interleaved allocations of varying sizes, are used to simulate differing realistic scenarios.
    c. **Metrics Collection**: Statistics such as the number of allocated chunks, minimum, maximum, and average chunk sizes, total memory usage, number of sbrk calls, and number of pages allocated are collected using the Mem_stats function after each test case execution.
    d. **Comparative Analysis**: The performance of different search policies, search starting locations, and coalescing options are compared and contrasted based on the collected statistics.

e. **Exploration of Combinations**: Various combinations of search policies, search locations, and coalescing options are explored to determine their impact on memory allocation performance and efficiency.

Comparison of Search Policy:

- **First Fit**: Allocates the first available block that is large enough to satisfy the request. Generally faster than best fit and worst fit but may lead to more fragmentation.
- **Best Fit**: Searches the entire free list to find the smallest block that is large enough to satisfy the request. Minimizes fragmentation but may require more time.
- **Worst Fit**: Allocates the largest available block. May result in more fragmentation but is faster than best fit due to simpler search logic.

Comparison of Search Starting Location:

- **Head First**: Starting the search from the head of the free list ensures that smaller blocks are allocated first, reducing fragmentation. However, it may take longer to find suitable blocks if the list is large.
- **Rover First**: Starting the search from the rover pointer allows for more efficient allocation of larger blocks, especially when the rover is moved dynamically based on allocation patterns. It may lead to less fragmentation but could potentially skip over smaller available blocks.

Comparison of Coalescing:

- **Enabled**: Coalescing combines adjacent free blocks into larger contiguous blocks, reducing fragmentation and improving memory utilization. However, it adds overhead to memory deallocation operations.

- **Disabled**: Without coalescing, free blocks remain separate, leading to increased fragmentation over time. However, memory deallocation operations are faster compared to when coalescing is enabled.

Impact of Combinations:

- Varying the combinations of search policy, search location, and coalescing can have significant effects on memory allocation performance and fragmentation. Certain combinations may be more suitable for specific use cases or allocation patterns.
- Experimentation with different combinations allows for optimizing memory allocation routines based on the application's requirements and constraints.