Name: Mahmoud Hany Fathalla **ID:20104375**
Name: Haneen Amr **ID: 20104448**

<u>ACPC grading system</u>

Python Code:

```
import random  # Import the random module for generating random numbers
import queue  # Import the queue module for creating a queue
import numpy as np  # Import the numpy module for numerical operations
import matplotlib.pyplot as plt  # Import the matplotlib module for plotting


def exponential_random(mean):
    """Generate a random number with exponential distribution."""
    return -mean * np.log(1 - random.random())  # Generate an exponential random vars



def simulate_acpc_grading_system(total_time, mean_interarrival, num_computers):
    """
    Simulates the ACPC grading system.

    Parameters:
        total_time (int): Total simulation time in seconds.
        mean_interarrival (float): Average interarrival time in seconds.
        num_computers (int): Number of available computers.
   Returns:
        dict: Simulation statistics.
    """
    print("Starting simulation...")  # Print the start of the simulation

    # Initialize simulation variables
    clock = 0  # Simulation clock
    computers = [0] * num_computers  # Tracks when each computer will be free
    task_queue = queue.Queue()  # Queue for waiting tasks
   # Data for visualization
    task_arrival_times = []  # List to store task arrival times
    task_completion_times = []  # List to store task completion times
    queue_lengths = []  # List to store queue lengths over time
    computer_usage = [[] for _ in range(num_computers)]  # Tracks computer utilization
    task_count_per_computer = [0] * num_computers #Tracks number of tasks per computer

    # Statistics
    total_delay_time = 0  # Total delay time for tasks
    total_waiting_time = 0  # Total waiting time for tasks
    total_tasks = 0  # Total number of tasks processed
    total_queue_length = 0  # Total length of the queue
    total_queue_updates = 0  # Total number of queue updates
   # Generate first task arrival time
    next_arrival_time = exponential_random(mean_interarrival)#Generate the first task
                                                                               arrival time
```

```
while clock < total_time:  # Run the simulation until the total time is reached
        # Determine the next event (arrival or service completion)
        next_free_computer_time = min([time for time in computers if time !=
        float('inf')], default=float('inf'))  # Find the next free computer time
        if next_arrival_time <= next_free_computer_time:
            # Process task arrival
            clock = next_arrival_time #Update the simulation clock to the next arrival
                                                                              time
            task_arrival_times.append(clock)  # Record the task arrival time
            print(f"Task arrived at time {clock:.2f}")  # Print the task arrival time

            # Random service time between 1 and 42 seconds to generate a new task
            service_time = random.uniform(1, 42)  # Generate a random service time

            if any(time == float('inf') for time in computers):
                # Check if any computer is free
                free_computer = computers.index(float('inf'))
               # Find the index of the free computer
                computers[free_computer] = clock + service_time
               # Assign the task to the free computer
                task_completion_times.append(computers[free_computer])
               # Record the task completion time
                computer_usage[free_computer].append((clock,computers[free_computer]))

                # Record the computer usage
                task_count_per_computer[free_computer] += 1  # Increment the task

                print(f"Task assigned to computer {free_computer} until time
               {computers[free_computer]:.2f}")  # Print the task assignment
                total_tasks += 1  # Increment the total number of tasks
                # Add the service time to the total waiting time
                total_waiting_time += service_time
            else:
                task_queue.put((clock, service_time))  # Add the task to the queue
                print(f"Task queued at time {clock:.2f}")  # Print the task queuing
            # Generate the next task arrival time
            next_arrival_time = clock + exponential_random(mean_interarrival)
        else:
            # Process service completion
            clock = next_free_computer_time  # Update the simulation clock to the next
                                                                 free computer time
            completed_computer = computers.index(next_free_computer_time)  # Find the
index of the completed computer
            print(f"Computer {completed_computer} completed a task at time
{clock:.2f}")  # Print the task completion

            if not task_queue.empty():
                arrival_time, service_time = task_queue.get()  # Get the next task
from the queue
                delay_time = clock - arrival_time  # Calculate the delay time for the
task
```

```
                total_delay_time += delay_time  # Add the delay time to the total
delay time
                total_waiting_time += delay_time + service_time  # Add the waiting
time to the total waiting time
                total_tasks += 1  # Increment the total number of tasks
                # Random service time for the task coming from the queue
                service_time = random.uniform(1, 42)  # Generate a random service time
                computers[completed_computer] = clock + service_time  # Assign the
task to the completed computer
                task_completion_times.append(computers[completed_computer])  # Record
the task completion time
                computer_usage[completed_computer].append((clock,
computers[completed_computer]))  # Record the computer usage
                task_count_per_computer[completed_computer] += 1  # Increment the task
count for the computer
                print(f"Task from queue assigned to computer {completed_computer}
until time {computers[completed_computer]:.2f}")  # Print the task assignment
            else:
                computers[completed_computer] = float('inf')  # Mark computer as idle
# Update queue stats
            total_queue_length += task_queue.qsize()  # Add the current queue size to
the total queue length
            total_queue_updates += 1  # Increment the total number of queue updates
            queue_lengths.append(task_queue.qsize())  # Record the current queue
length
 # Calculate averages
    avg_delay_time = total_delay_time / total_tasks if total_tasks > 0 else 0  #
Calculate the average delay time
    avg_waiting_time = total_waiting_time / total_tasks if total_tasks > 0 else 0  #
Calculate the average waiting time
    avg_queue_length = total_queue_length / total_queue_updates if
total_queue_updates > 0 else 0  # Calculate the average queue length
 print("Simulation completed.")  # Print the end of the simulation

    # Return statistics
    return {
        "Average Delay Time": avg_delay_time,
        "Average Waiting Time": avg_waiting_time,
        "Average Queue Length": avg_queue_length,
        "Total Tasks Processed": total_tasks,
        "task_arrival_times": task_arrival_times,
        "task_completion_times": task_completion_times,
        "queue_lengths": queue_lengths,
        "computer_usage": computer_usage,
        "task_count_per_computer": task_count_per_computer
    }



def visualize(results):
    """
```

```python
    Visualizes task start and end times for each computer, queue length over time,
    and computer utilization over time.
    """
    fig, axs = plt.subplots(2, 2, figsize=(15, 10))  # Create a 2x2 subplot for
visualization

    # Plot queue length over time
    axs[0, 0].plot([x for x in range(len(results["queue_lengths"]))],
results["queue_lengths"], label="Queue Length", color='r')  # Plot queue length over
time
    axs[0, 0].set_xlabel('Time Steps (x100)')  # Set x-axis label
    axs[0, 0].set_ylabel('Queue Length')  # Set y-axis label
    axs[0, 0].set_title('Queue Length Over Time')  # Set plot title
    axs[0, 0].grid(True)  # Enable grid
    axs[0, 0].legend()  # Show legend
 # Plot computer utilization over time
    for i, usage in enumerate(results["computer_usage"]):
        times = [start for start, end in usage]  # Get start times for each computer
        axs[0, 1].plot(times, [i] * len(times), marker='o', label=f"Computer {i}
Busy")  # Plot computer utilization over time
    axs[0, 1].set_xlabel('Time Steps')  # Set x-axis label
    axs[0, 1].set_ylabel('Computer ID')  # Set y-axis label
    axs[0, 1].set_title('Computer Utilization Over Time')  # Set plot title
    axs[0, 1].grid(True)  # Enable grid
    axs[0, 1].legend()  # Show legend
# Plot task completion times for each computer
    for i, task_times in enumerate(results["computer_usage"]):
        completion_times = [end for _, end in task_times]  # Get completion times for
each computer
        axs[1, 0].scatter(completion_times, [i] * len(completion_times),
label=f"Computer {i} Task Completion")  # Plot task completion times
    axs[1, 0].set_xlabel('Task Completion Time')  # Set x-axis label
    axs[1, 0].set_ylabel('Computer ID')  # Set y-axis label
    axs[1, 0].set_title('Task Completion Times for Each Computer')  # Set plot title
    axs[1, 0].grid(True)  # Enable grid
    axs[1, 0].legend()  # Show legend
# Plot computer utilization over time with task arrival (red) and completion (green)
    for i, usage in enumerate(results["computer_usage"]):
        task_arrival_times = [start for start, _ in usage]  # Get task arrival times
for each computer
        task_completion_times = [end for _, end in usage]  # Get task completion times
for each computer
# Plot task arrivals (red)
        axs[1, 1].scatter(task_arrival_times, [i] * len(task_arrival_times),
color='red', label=f"Computer {i} Task Arrival", marker='o')  # Plot task arrivals

        # Plot task completions (green)
        axs[1, 1].scatter(task_completion_times, [i] * len(task_completion_times),
color='green', label=f"Computer {i} Task Completion", marker='x')  # Plot task
completions
```

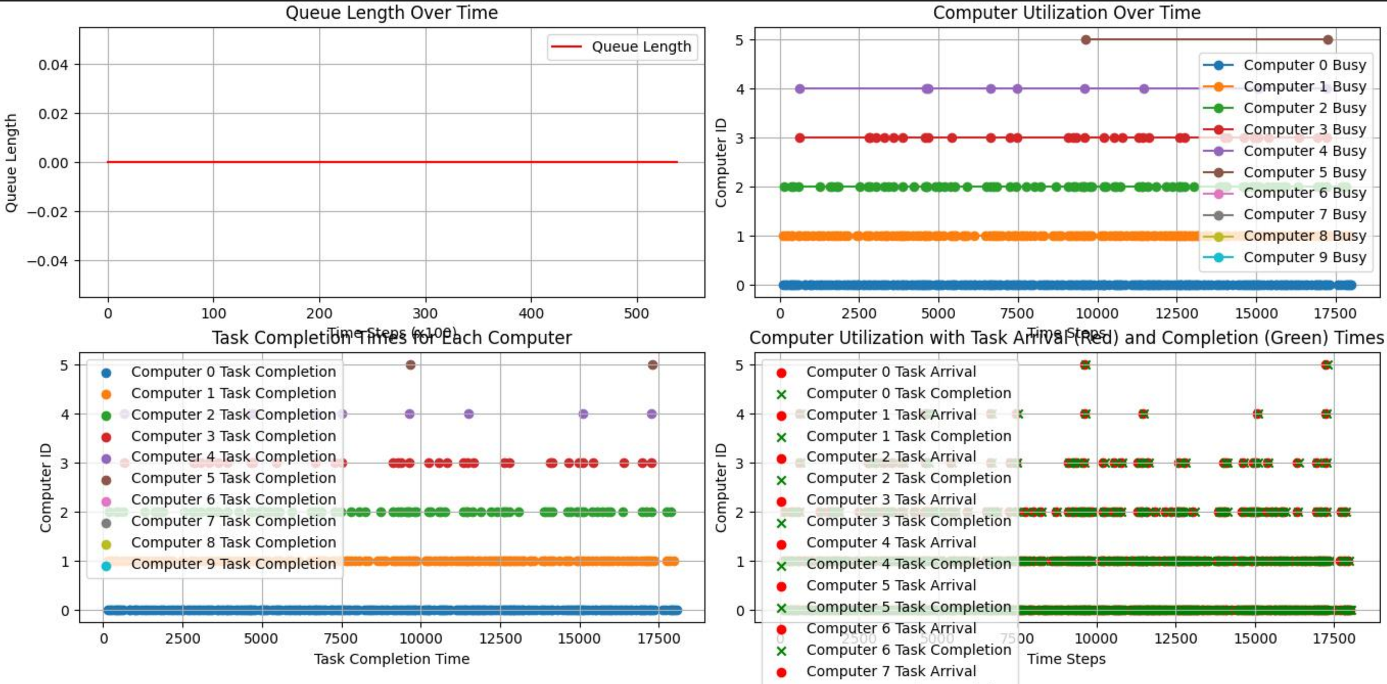
```
    axs[1, 1].set_xlabel('Time Steps')  # Set x-axis label
    axs[1, 1].set_ylabel('Computer ID')  # Set y-axis label
    axs[1, 1].set_title('Computer Utilization with Task Arrival (Red) and Completion
(Green) Times')  # Set plot title
    axs[1, 1].grid(True)  # Enable grid
    axs[1, 1].legend()  # Show legend
    plt.tight_layout()  # Adjust layout
    plt.show()  # Show the plot


if __name__ == "__main__":
    # Simulation parameters
    TOTAL_TIME = 5 * 60 * 60  # 5 hours in seconds
    MEAN_INTERARRIVAL = 35  # seconds
    NUM_COMPUTERS = 10  # Number of computers
    # Run simulation
    print("Initializing simulation parameters...")  # Print initialization message
    results= simulate_acpc_grading_system(TOTAL_TIME,MEAN_INTERARRIVAL, NUM_COMPUTERS)
   # Print results
    print("Simulation results:")  # Print results message
    for key, value in results.items():
        if isinstance(value, (float, int)):
            print(f"{key}: {value:.2f}")  # Print each result

    # Visualize task handling by each computer and queue length over time
    # Print task count per computer
    print("\nTotal Tasks Handled by Each Computer:")  # Print task count message
    for i, task_count in enumerate(results["task_count_per_computer"]):
        print(f"Computer {i}: {task_count} tasks")  # Print task count for each
computer
    visualize(results)  # Visualize the results
-------------------------------------------------------------------------------------
```

Code Output Analysis

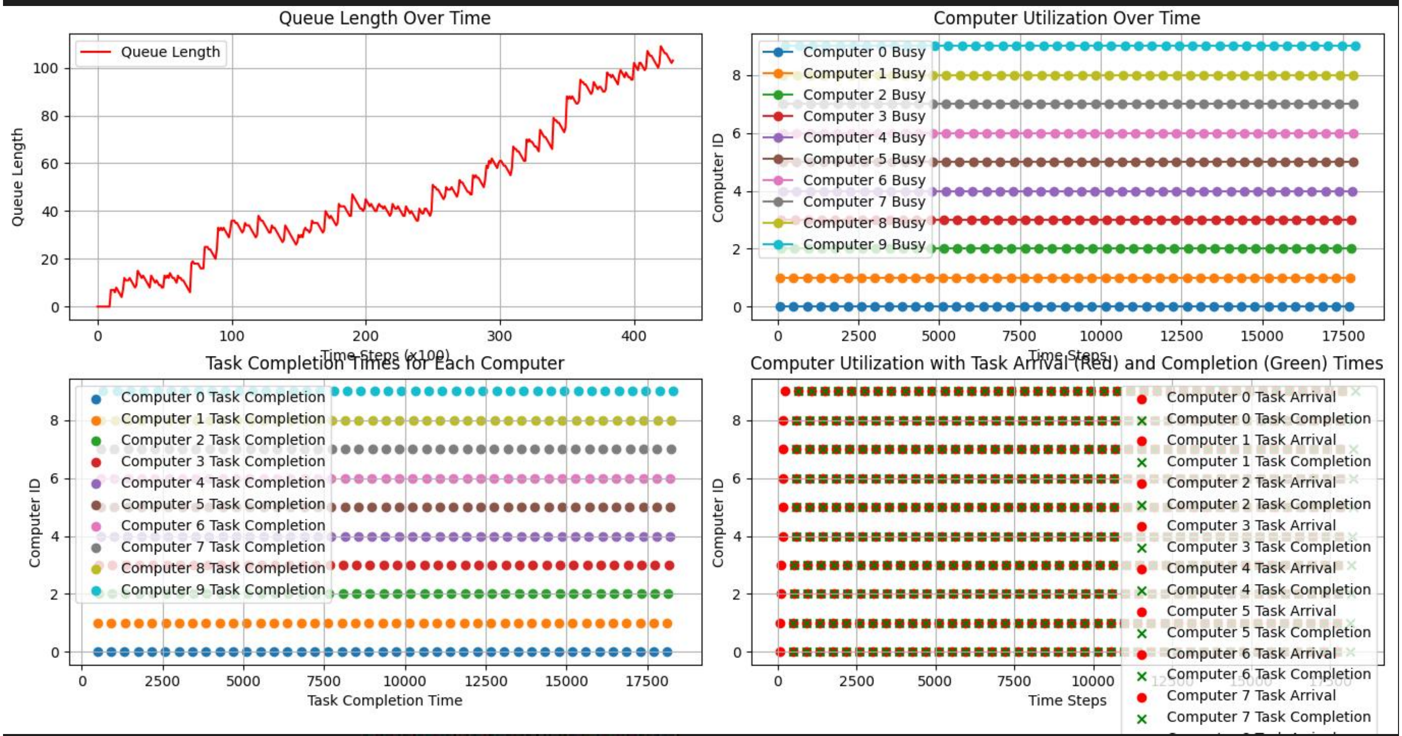
When service_time is 42 sec:




```
Simulation completed.
Simulation results:
Average Delay Time: 0.00
Average Waiting Time: 42.00
Average Queue Length: 0.00
Total Tasks Processed: 529.00

Total Tasks Handled by Each Computer:
Computer 0: 232 tasks
Computer 1: 168 tasks
Computer 2: 84 tasks
Computer 3: 34 tasks
Computer 4: 9 tasks
Computer 5: 2 tasks
Computer 6: 0 tasks
Computer 7: 0 tasks
Computer 8: 0 tasks
Computer 9: 0 tasks
```

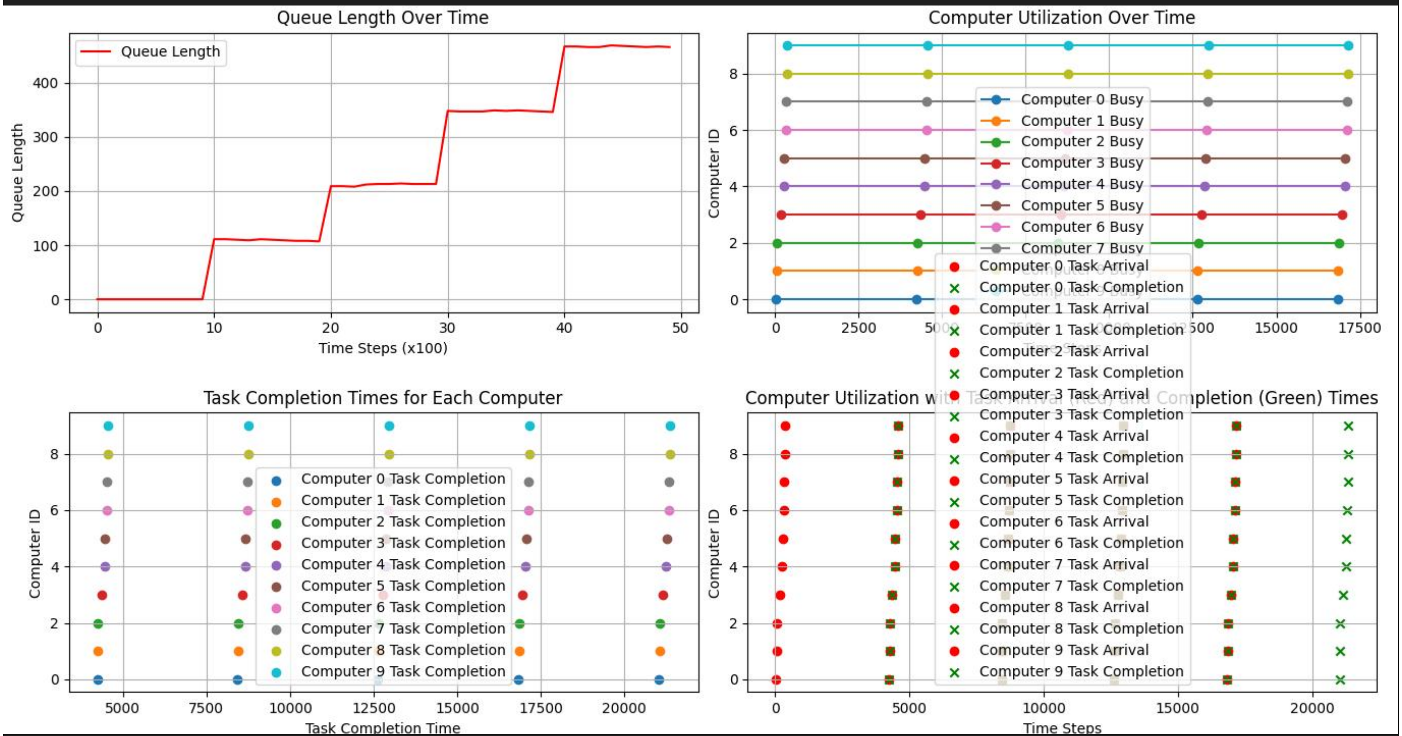
When service_time is 420 sec:




```
Simulation completed.
Simulation results:
Average Delay Time: 1513.80
Average Waiting Time: 1933.80
Average Queue Length: 41.77
Total Tasks Processed: 426.00

Total Tasks Handled by Each Computer:
Computer 0: 43 tasks
Computer 1: 43 tasks
Computer 2: 43 tasks
Computer 3: 43 tasks
Computer 4: 43 tasks
Computer 5: 43 tasks
Computer 6: 42 tasks
Computer 7: 42 tasks
Computer 8: 42 tasks
Computer 9: 42 tasks
```

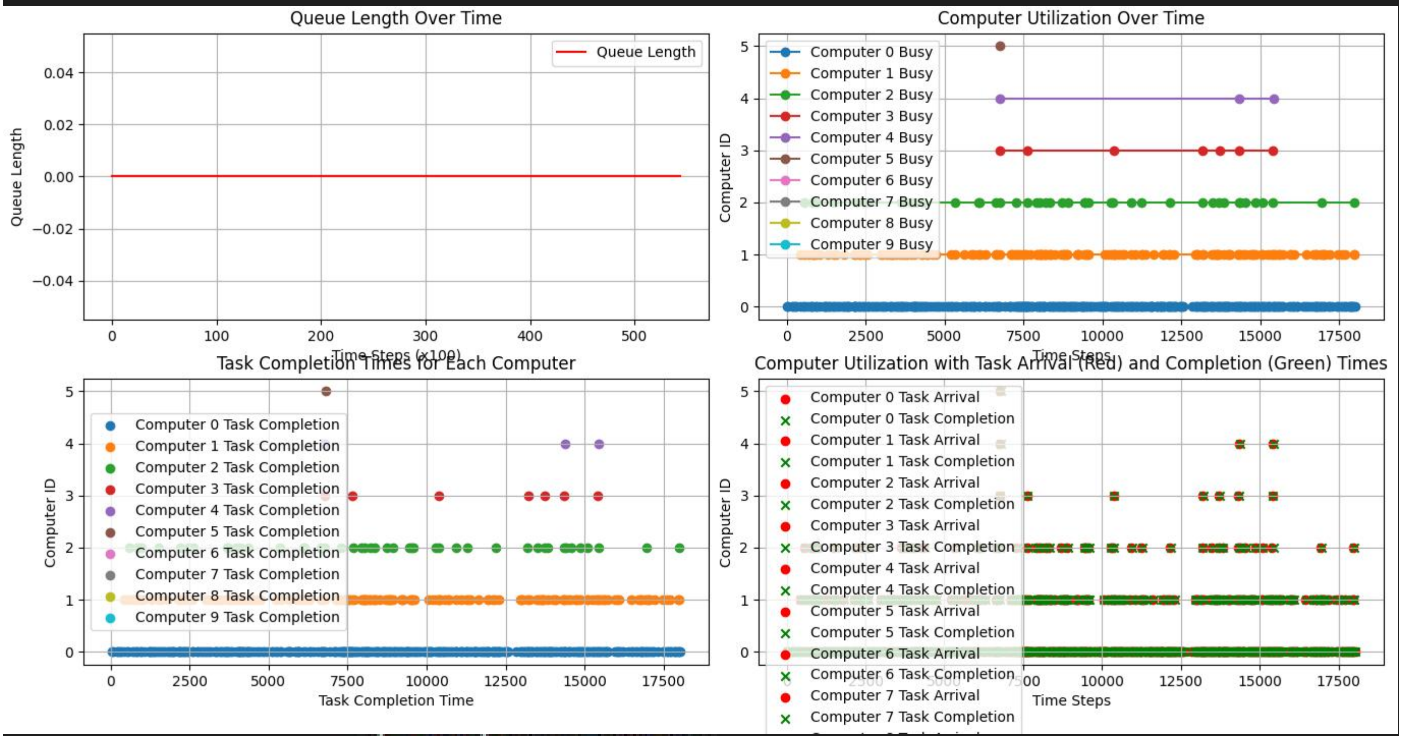
When service_time is 4200 sec:




```
Simulation completed.
Simulation results:
Average Delay Time: 7529.06
Average Waiting Time: 11729.06
Average Queue Length: 227.12
Total Tasks Processed: 50.00

Total Tasks Handled by Each Computer:
Computer 0: 5 tasks
Computer 1: 5 tasks
Computer 2: 5 tasks
Computer 3: 5 tasks
Computer 4: 5 tasks
Computer 5: 5 tasks
Computer 6: 5 tasks
Computer 7: 5 tasks
Computer 8: 5 tasks
Computer 9: 5 tasks
```

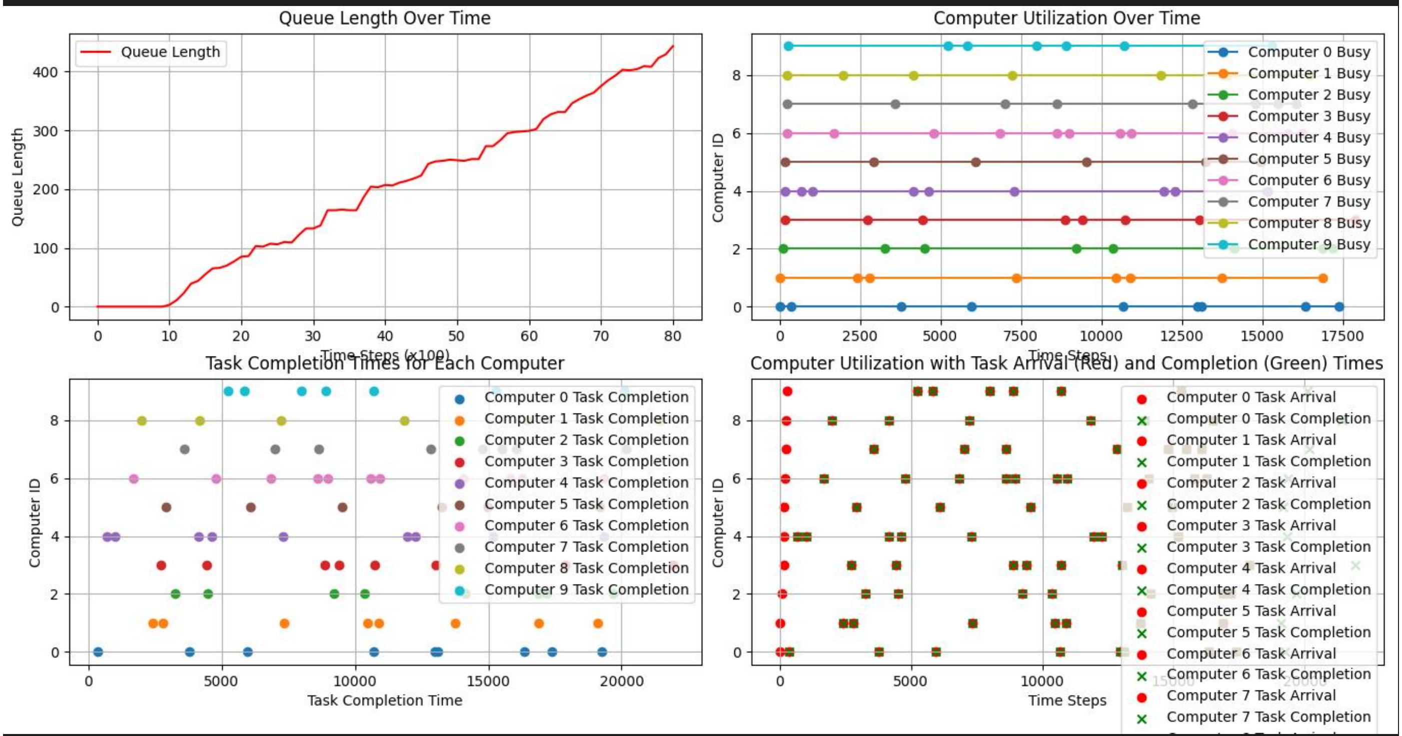
When service_time is Random between 1 to 42 sec:




```
Simulation completed.
Simulation results:
Average Delay Time: 0.00
Average Waiting Time: 21.38
Average Queue Length: 0.00
Total Tasks Processed: 536.00

Total Tasks Handled by Each Computer:
Computer 0: 325 tasks
Computer 1: 154 tasks
Computer 2: 46 tasks
Computer 3: 7 tasks
Computer 4: 3 tasks
Computer 5: 1 tasks
Computer 6: 0 tasks
Computer 7: 0 tasks
Computer 8: 0 tasks
Computer 9: 0 tasks
```

When service_time is Random between 1 to 5000 sec:




```
Simulation completed.
Simulation results:
Average Delay Time: 7081.21
Average Waiting Time: 9676.09
Average Queue Length: 197.42
Total Tasks Processed: 81.00

Total Tasks Handled by Each Computer:
Computer 0: 9 tasks
Computer 1: 8 tasks
Computer 2: 8 tasks
Computer 3: 8 tasks
Computer 4: 9 tasks
Computer 5: 6 tasks
Computer 6: 11 tasks
Computer 7: 8 tasks
Computer 8: 7 tasks
Computer 9: 7 tasks
```

Conclusion

The simulation results for the ACPC grading system server demonstrate the system's performance under various scenarios. For a service time of 42 seconds, the system efficiently processed 529 tasks with minimal delay, no queue buildup, and a balanced load distribution among the 10 computers. When the service time increased to 420 seconds, the average delay time, waiting time, and queue length surged dramatically, resulting in only 426 tasks being processed. For the longest service time of 4200 seconds, the system faced extreme congestion, with average delay and waiting times exceeding 7500 and 11700 seconds, respectively, and only 50 tasks processed. Additionally, when testing with a realistic random service time between 1 and 42 seconds, the system achieved optimal performance, processing 536 tasks using only 5 computers, with no delay, no queue buildup, and an average waiting time of 0. For a more challenging real-world scenario with service times randomly generated between 1 and 5000 seconds, the system processed 81 tasks, utilizing all 10 computers. In this case, the average delay time was 7081.2 seconds, the average waiting time was 9676.09 seconds, and the average queue length was 197.42. These results highlight the server's capability to handle shorter and moderate service times effectively while experiencing significant performance degradation under prolonged service times.