In [ ]:

```python
# Define queue lengths and their durations
queue_data = [
    {"length": 3, "duration": 5},
    {"length": 4, "duration": 10},
    {"length": 2, "duration": 6},
    {"length": 3, "duration": 9}
]

total_weighted_length = 0
total_duration = 0

for item in queue_data:
    total_weighted_length += item["length"] * item["duration"]
    total_duration += item["duration"]

# Calculate the average queue length
if total_duration > 0:
    average_queue_length = total_weighted_length / total_duration
    print(f"Total duration: {total_duration} minutes")
    print(f"Total weighted length: {total_weighted_length}")
    print(f"Average queue length: {average_queue_length:.2f}")
else:
    print("Error: Total duration is zero, cannot calculate average.")
```

```
Total duration: 30 minutes
Total weighted length: 94
Average queue length: 3.13
```

Detailed Calculations for Helicopter Queuing Problem

We will use the data from the previous manual computation:

Helicopter	IAT (hrs)	ST (hrs)	AT (hrs)	SST (hrs)	SET (hrs)	WT (hrs)	TS (hrs)
1	0.0	2.2	0.0	0.0	2.2	0.0	2.2
2	1.4	1.2	1.4	2.2	3.4	0.8	2.0
3	2.1	0.3	3.5	3.5	3.8	0.0	0.3
4	1.9	2.8	5.4	5.4	8.2	0.0	2.8
5	0.2	0.2	5.6	8.2	8.4	2.6	2.8
6	1.5	1.8	7.1	8.4	10.2	1.3	3.1

1. Minimum and Maximum Queue Length

To determine the queue length over time, we track the state of the system at each event (arrival or service completion). Queue length here refers to the number of helicopters waiting for service.

- **Time 0.0:** Helicopter 1 arrives. Mechanic is idle, so H1 starts service immediately. **Queue Length = 0.**
- **Time 1.4:** Helicopter 2 arrives. Helicopter 1 is still in service (until 2.2). H2 joins the queue. **Queue Length = 1.**
- **Time 2.2:** Helicopter 1 finishes service. H2, which was waiting, starts service immediately. **Queue Length = 0.**
- **Time 3.4:** Helicopter 2 finishes service. No helicopters are waiting. Mechanic becomes idle. **Queue Length = 0.**
- **Time 3.5:** Helicopter 3 arrives. Mechanic is idle, so H3 starts service immediately. **Queue Length = 0.**

- **Time 3.8:** Helicopter 3 finishes service. No helicopters are waiting. Mechanic becomes idle. **Queue Length = 0.**
- **Time 5.4:** Helicopter 4 arrives. Mechanic is idle, so H4 starts service immediately. **Queue Length = 0.**
- **Time 5.6:** Helicopter 5 arrives. Helicopter 4 is still in service (until 8.2). H5 joins the queue. **Queue Length = 1.**
- **Time 7.1:** Helicopter 6 arrives. Helicopter 4 is still in service. H5 is waiting. H6 joins the queue. **Queue Length = 2.**
- **Time 8.2:** Helicopter 4 finishes service. H5, which was first in queue, starts service. **Queue Length = 1.**
- **Time 8.4:** Helicopter 5 finishes service. H6, which was waiting, starts service. **Queue Length = 0.**
- **Time 10.2:** Helicopter 6 finishes service. No helicopters are waiting. Mechanic becomes idle. **Queue Length = 0.**

From the sequence of events:

- **Minimum Queue Length = 0**
- **Maximum Queue Length = 2**

2. Total Queue Time (Total time spent in queue for all pilots)

This is the sum of the Waiting Times (WT) for all helicopters.

Total Queue Time = WT(H1) + WT(H2) + WT(H3) + WT(H4) + WT(H5) + WT(H6) Total Queue Time = 0.0 + 0.8 + 0.0 + 0.0 + 2.6 + 1.3 **Total Queue Time = 4.7 hours**

3. Average Waiting Time in Queue

This is the Total Queue Time divided by the number of helicopters.

Number of Helicopters = 6 Average Waiting Time = Total Queue Time / Number of Helicopters Average Waiting Time = 4.7 hours / 6 **Average Waiting Time = 0.7833 hours** (approximately)

4. Total Idle Time for the Mechanic

The mechanic is idle when there is a gap between the Service End Time (SET) of one helicopter and the Service Start Time (SST) of the next helicopter, and no one is waiting.

- Idle Time before H1: 0 (H1 arrives at the start of shift)
- Idle Time between H1 and H2: SST(H2) - SET(H1) = 2.2 - 2.2 = 0.0 hours
- Idle Time between H2 and H3: SST(H3) - SET(H2) = 3.5 - 3.4 = 0.1 hours
- Idle Time between H3 and H4: SST(H4) - SET(H3) = 5.4 - 3.8 = 1.6 hours
- Idle Time between H4 and H5: SST(H5) - SET(H4) = 8.2 - 8.2 = 0.0 hours
- Idle Time between H5 and H6: SST(H6) - SET(H5) = 8.4 - 8.4 = 0.0 hours

Total Idle Time = 0.0 + 0.1 + 1.6 + 0.0 + 0.0 **Total Idle Time = 1.7 hours**

5. Resource Utilization for the Mechanic

Resource utilization is the proportion of time the mechanic is busy servicing helicopters relative to the total time the system is observed.

- **Total Service Time (Total Busy Time)**: Sum of all Service Times (ST). Total Service Time = 2.2 + 1.2 + 0.3 + 2.8 + 0.2 + 1.8 = **8.5 hours**
- **Total Time System is Observed**: This is the time from the arrival of the first helicopter (Time 0.0) until the completion of service for the last helicopter (SET for H6). Total Observed Time = SET(H6) - AT(H1) = 10.2 - 0.0 = **10.2 hours**
- **Resource Utilization** = (Total Service Time) / (Total Observed Time) Resource Utilization = 8.5 hours / 10.2 hours **Resource Utilization ≈ 0.8333 or 83.33%**

In [ ]:

```
import pandas as pd
import io

# Raw data provided by the user
data_str = """
Ship Interarrival Time Service Time Arrival Time Service Start Service End
1 0.0 2.5 0.0 0.0 2.5
2 1.0 1.0
3 1.0 2.0 2.0 2.5 4.5
4 1.0 3.0
5 1.0 3.0 4.0 4.5 7.5
6 1.0 1.5 5.0 7.5 9.0
7 1.0 6.0
8 1.0 2.0 7.0 9.0 11.0
9 1.0 1.0 8.0 11.0 12.0
10 1.0 9.0
11 1.0 10.0
12 1.0 11.0
13 1.0 2.0 12.0 12.0 14.0
14 1.0 13.0
15 1.0 14.0
16 1.0 0.5 15.0 15.0 15.5
17 1.0 16.0
18 1.0 1.5 17.0 17.0 18.5
19 1.0 3.0 18.0 18.5 21.5
20 1.0 19.0
"""

# --- Step 1: Parse and complete the table ---

processed_data = []
raw_lines = data_str.strip().split('\n')

for line in raw_lines[1:]:
    parts = line.split()
    record = {
        "ship": int(parts[0]),
        "iat": float(parts[1]),
        "st": float(parts[2]),
        "at": None,
        "sst": None,
        "set": None
    }
    # If there are more than 3 parts, it means AT, SST, SET are provided
    if len(parts) > 3:
        record["at"] = float(parts[3])
    if len(parts) > 4:
        record["sst"] = float(parts[4])
    if len(parts) > 5:
        record["set"] = float(parts[5])
    processed_data.append(record)
```

```python
# Initialize state variables for calculations
last_arrival_time_calc = 0.0
facility_free_time_calc = 0.0
completed_ships_data = []

# Iterate through ships to calculate missing values and derive WT, TS
for i, ship_data in enumerate(processed_data):
    current_ship = ship_data.copy()

    # Calculate or use given Arrival Time (AT)
    if current_ship["at"] is None:
        # For the first ship, AT is its IAT (which is 0 in this case)
        if i == 0:
            current_ship["at"] = current_ship["iat"]
        else:
            current_ship["at"] = last_arrival_time_calc + current_ship["iat"]
    last_arrival_time_calc = current_ship["at"]

    # Calculate or use given Service Start Time (SST)
    if current_ship["sst"] is None:
        current_ship["sst"] = max(current_ship["at"], facility_free_time_calc)

    # Calculate or use given Service End Time (SET)
    if current_ship["set"] is None:
        current_ship["set"] = current_ship["sst"] + current_ship["st"]
    # Update facility_free_time based on the current ship's (given or calculated) SET
    facility_free_time_calc = current_ship["set"]

    # Calculate Waiting Time (WT)
    current_ship["wt"] = current_ship["sst"] - current_ship["at"]

    # Calculate Time in System (TS)
    current_ship["ts"] = current_ship["set"] - current_ship["at"]

    completed_ships_data.append(current_ship)

# Display the completed table
print("### Completed Ship Maintenance Schedule Table\n")
output_table = "| Ship | IAT | ST | AT | SST | SET | WT | TS |\n"
output_table +=   "|:----:|:---:|:--:|:--:|:---:|:---:|:--:|:--:|\n"
for row in completed_ships_data:
    output_table += (
        f"| {row['ship']}    | {row['iat']:.1f} | {row['st']:.1f} | {row['at']:.1f} | "
        f"{row['sst']:.1f} | {row['set']:.1f} | {row['wt']:.1f} | {row['ts']:.1f} |\n"
    )
print(output_table)

# --- Step 2: Calculate requested metrics ---

# 1) Average waiting time for the ships that need service
# This includes all ships, even if their waiting time is 0.
total_waiting_time = sum(s["wt"] for s in completed_ships_data)
num_ships = len(completed_ships_data)
average_waiting_time = total_waiting_time / num_ships

print(f"### 1) Average Waiting Time for ships that need service:")
print(f"Total Waiting Time: {total_waiting_time:.2f} days")
print(f"Number of Ships: {num_ships}")
print(f"**Average Waiting Time: {average_waiting_time:.4f} days**\n")

# 2) Calculate the 20-day resource utilization of the single facility
observation_period = 20.0 # days

# Collect all service intervals that fall within the 20-day period [0, 20]
raw_busy_intervals = []
for ship in completed_ships_data:
```

```
    service_start = ship["sst"]
    service_end = ship["set"]

    # Only consider services that start before the observation_period
    if service_start < observation_period:
        # Clamp the service end time to the observation_period if it extends beyond
        effective_end = min(service_end, observation_period)
        if effective_end > service_start: # Ensure it's a valid interval with positive dui
            raw_busy_intervals.append((service_start, effective_end))

# Sort intervals by start time to prepare for merging
raw_busy_intervals.sort()

# Merge overlapping intervals to get the total busy time
merged_intervals = []
if raw_busy_intervals:
    merged_intervals.append(list(raw_busy_intervals[0])) # Start with the first interval

    for current_start, current_end in raw_busy_intervals[1:]:
        last_merged_start, last_merged_end = merged_intervals[-1]

        if current_start <= last_merged_end: # Intervals overlap or touch
            merged_intervals[-1][1] = max(last_merged_end, current_end)
        else: # No overlap, add as a new interval
            merged_intervals.append([current_start, current_end])

# Calculate total busy time from merged intervals
total_busy_time_in_20_days = 0.0
for start, end in merged_intervals:
    total_busy_time_in_20_days += (end - start)

print(f"### 2) 20-day Resource Utilization of the single facility:")
print(f"Total busy intervals within 20 days (merged): {merged_intervals}")
print(f"Total facility busy time (within 20 days): {total_busy_time_in_20_days:.2f} days")
print(f"Observation Period: {observation_period:.2f} days")

resource_utilization = total_busy_time_in_20_days / observation_period
print(f"**Resource Utilization (20-day): {resource_utilization:.4f} or {resource_utilizati
```

Completed Ship Maintenance Schedule Table

Ship	IAT	ST	AT	SST	SET	WT	TS
1	0.0	2.5	0.0	0.0	2.5	0.0	2.5
2	1.0	1.0	1.0	2.5	3.5	1.5	2.5
3	1.0	2.0	2.0	2.5	4.5	0.5	2.5
4	1.0	3.0	3.0	4.5	7.5	1.5	4.5
5	1.0	3.0	4.0	4.5	7.5	0.5	3.5
6	1.0	1.5	5.0	7.5	9.0	2.5	4.0
7	1.0	6.0	6.0	9.0	15.0	3.0	9.0
8	1.0	2.0	7.0	9.0	11.0	2.0	4.0
9	1.0	1.0	8.0	11.0	12.0	3.0	4.0
10	1.0	9.0	9.0	12.0	21.0	3.0	12.0
11	1.0	10.0	10.0	21.0	31.0	11.0	21.0
12	1.0	11.0	11.0	31.0	42.0	20.0	31.0
13	1.0	2.0	12.0	12.0	14.0	0.0	2.0
14	1.0	13.0	13.0	14.0	27.0	1.0	14.0
15	1.0	14.0	14.0	27.0	41.0	13.0	27.0
16	1.0	0.5	15.0	15.0	15.5	0.0	0.5
17	1.0	16.0	16.0	16.0	32.0	0.0	16.0
18	1.0	1.5	17.0	17.0	18.5	0.0	1.5
19	1.0	3.0	18.0	18.5	21.5	0.5	3.5
20	1.0	19.0	19.0	21.5	40.5	2.5	21.5

1) Average Waiting Time for ships that need service:

Total Waiting Time: 65.50 days
Number of Ships: 20
Average Waiting Time: 3.2750 days

2) 20-day Resource Utilization of the single facility:
Total busy intervals within 20 days (merged): [[0.0, 20.0]]
Total facility busy time (within 20 days): 20.00 days
Observation Period: 20.00 days
Resource Utilization (20-day): 1.0000 or 100.00%