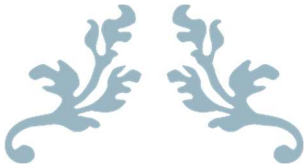

HOSPITAL PATIENT RECORDS: A COMPARISON OF BUBBLE SORT AND MERGE SORT

By Brandon Everett

FEBRUARY 2, 2025

CSC-506

Module #3

Hospital Patient Records System: A Comparison of Bubble Sort and Merge Sort

Sorting patient records is a big deal in hospitals because it helps doctors and nurses find the information they need quickly. This paper compares two ways to sort records: Bubble Sort and Merge Sort to see which one works better for large datasets. To test this, I created a PatientRecord class in Python that stores each patient's ID, name, and admission date. I then tested both sorting methods using Python to see how fast they can organize these patient records.

```python
# Patient class
class PatientRecord:
    def __init__(self, id, name, admission_date):
        self.id = id
        self.name = name
        self.admission_date = admission_date

    def __str__(self):
        return f"Patient {self.name} (ID: {self.id}) admitted on {self.admission_date}"
```

(Patient Class)

Sorting Performance and Test Data: At first, I tried sorting just a few records, and both methods seemed about the same. But when I tested them with 1000 patient records, the difference became clear. I used Python's Faker library to create realistic patient data, including IDs, names, and admission dates (LearnPython, 2024).

Here's what happened:

```
Trying Bubble Sort...
Patient Eric Costa (ID: 391) admitted on 01/20/2025
Patient Daniel Small (ID: 437) admitted on 01/29/2025
Patient Joseph Curtis (ID: 551) admitted on 01/12/2025
Patient Stephen Jones (ID: 671) admitted on 01/06/2025
Patient Kathleen Taylor (ID: 1426) admitted on 01/12/2025
Bubble Sort took 0.095848 seconds

Trying Merge Sort...
Patient Eric Costa (ID: 391) admitted on 01/20/2025
Patient Daniel Small (ID: 437) admitted on 01/29/2025
Patient Joseph Curtis (ID: 551) admitted on 01/12/2025
Patient Stephen Jones (ID: 671) admitted on 01/06/2025
Patient Kathleen Taylor (ID: 1426) admitted on 01/12/2025
Merge Sort took 0.001999 seconds
```

(results with 1000 patients)

Bubble Sort: This method took about 0.096 seconds to sort the records. It works by comparing and swapping adjacent records until everything is in order.

```python
# Bubble Sort
def bubble_sort(patients):
    start = time.time()
    for i in range(len(patients)):
        for j in range(len(patients) - 1):
            if patients[j].id > patients[j + 1].id:
                patients[j], patients[j + 1] = patients[j + 1], patients[j]
    end = time.time()
    return patients, end - start
```

(Bubble Sort Code)

Merge Sort: This method was much faster, taking only 0.002 seconds. It works by splitting the records into smaller groups, sorting them, and then merging them back together.

```
def merge(left_half, right_half):
    left_index = 0
    right_index = 0
    tmp_list = []

    while left_index < len(left_half) and right_index < len(right_half):
        if left_half[left_index].id > right_half[right_index].id:
            tmp_list.append(right_half[right_index])
            right_index += 1
        else:
            tmp_list.append(left_half[left_index])
            left_index += 1

    # Add remaining elements from left half
    tmp_list.extend(left_half[left_index:])
    # Add remaining elements from right half
    tmp_list.extend(right_half[right_index:])

    return tmp_list
```

```
def merge_sort_wrapper(patients):
    start = time.time()

    def merge_sort(arr):
        if len(arr) <= 1:
            return arr

        mid = len(arr) // 2
        left_half = merge_sort(arr[:mid])
        right_half = merge_sort(arr[mid:])

        return merge(left_half, right_half)

    sorted_patients = merge_sort(patients)
    end = time.time()
    return sorted_patients, end - start
```

(Merge Sort Code)

Results: The results show that Merge Sort is a better choice for hospitals with lots of records because it's much faster.

Challenges in Sorting Patient Records: Each record has important details like names, dates, and possibly test results. If the sorting isn't done right, it could lead to mistakes, like mixing up patients or delaying care (Nelson, 2017).

```
# Generate test patients
def generate_patients(num_patients=10000):
    patients = []
    for _ in range(num_patients):
        patient_id = randint(100, 999)
        name = fake.name()
        admission_date = fake.date_between(start_date="-30d", end_date="today").strftime('%m/%d/%Y')
        patients.append(PatientRecord(patient_id, name, admission_date))
    return patients
```

(Data Structure List: Using Faker to Create Patients)

When I first tested my code, I only used 20 patients. The problem was both sorting methods happened so fast that I couldn't really tell which one was faster. I knew I needed way more test data to see any real difference. After some Googling, I found a library called Faker that lets you make up fake names and dates. This was perfect because I could create tons of fake patient data that looked real, with unique names and admission dates. It made testing the sorting methods much easier since I could test with thousands of patients instead of just 20.

```
def test_sorting():
    patients = generate_patients()
    print("Patients before sorting:")
    for patient in patients[:5]:
        print(patient)

    print("\nTrying Bubble Sort...")
    sorted_bubble, bubble_time = bubble_sort(patients.copy())
    for patient in sorted_bubble[:5]:
        print(patient)
    print(f"Bubble Sort took {bubble_time:.6f} seconds")

    print("\nTrying Merge Sort...")
    sorted_merge, merge_time = merge_sort_timer(patients.copy())
    for patient in sorted_merge[:5]:
        print(patient)
    print(f"Merge Sort took {merge_time:.6f} seconds")
```

(Testing Sorting Algorithms)

Another challenge was understanding how Merge Sort works because it's a bit more complicated than Bubble Sort. But once I broke it down into smaller steps, it made more sense.

When to Use Each Method: Bubble Sort is simple and works fine for small clinics with only a few records. But as the number of records grows, it becomes slow and inefficient. With 10,000 records it took over 10 seconds while Merge Sort, on the other hand took .02, which would be much faster and better for large hospitals with thousands of records. It's a bit more complex, but speed makes it worth it.

Time Complexity and Practical Implications: Bubble Sort has a time complexity of $O(n^2)$, which means it gets slower as the dataset grows. For example, if you double the number of records, it could take four times as long to sort them. Merge Sort, on the other hand, has a time complexity of O(n log n), which means it stays fast even with large datasets (GeeksforGeeks, 2024).

However, Merge Sort uses more memory because it needs extra space to merge the sorted groups. This is something to keep in mind if the system has limited resources. (GeeksforGeeks, 2024).

Conclusion: In conclusion, Merge Sort is the better choice for hospitals with large datasets due to its faster performance. While it uses more memory, the trade-off is worth it for the time saved

in sorting patient records.

References

"How to Check the Execution Time of Python Script." *GeeksforGeeks*, 26 Apr. 2023, www.geeksforgeeks.org/how-to-check-the-execution-time-of-python-script/. Accessed 2 Feb. 2025.

Nelson, G. S. *A Practical Guide to Healthcare Data: Tips, Traps, and Techniques*. ThotWave Technologies, 2017.

"Generate Test Data in Python." *LearnPython*, 2024, learnpython.com/blog/generate-test-data-in-python/. Accessed 2 Feb. 2025.

Nguyen, Dong. "3.4 Merge Sort." CSC506 - Design and Analysis of Algorithms, Module 3, Colorado State University Global, 2025, csuglobal.instructure.com/courses/104935/pages/3-dot-4-merge-sort?module_item_id=5432068. Accessed 2 Feb. 2025.