

Lab 12: Algorithms with AI Assistance

Sorting, Searching, and Algorithm Optimization Using AI Tools

2303A51760

Batch-28

K. Sai Sathvik

TASK 1: Sorting Student Records (Quick Sort & Merge Sort)

Code

```
import random
import time

# Generate student records
students = [
    {"name": f"Student{i}", "roll": i, "cgpa": round(random.uniform(6, 10), 2)}
    for i in range(1, 101)
]

# ----- QUICK SORT -----
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr)//2]["cgpa"]
    left = [x for x in arr if x["cgpa"] > pivot]
    middle = [x for x in arr if x["cgpa"] == pivot]
    right = [x for x in arr if x["cgpa"] < pivot]
    return quick_sort(left) + middle + quick_sort(right)

# ----- MERGE SORT -----
```

```

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

    mid = len(arr)//2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])

    return merge(left, right)

def merge(left, right):
    result = []
    while left and right:
        if left[0]["cgpa"] > right[0]["cgpa"]:
            result.append(left.pop(0))
        else:
            result.append(right.pop(0))
    result.extend(left)
    result.extend(right)
    return result

# Measure performance
start = time.time()
qs_sorted = quick_sort(students)
qs_time = time.time() - start

start = time.time()
ms_sorted = merge_sort(students)
ms_time = time.time() - start

# Top 10 Students
print("Top 10 Students by CGPA (Quick Sort):")
for s in qs_sorted[:10]:
    print(s)

print("\nQuick Sort Time:", qs_time)
print("Merge Sort Time:", ms_time)

```

Sample Output

Top 10 Students by CGPA (Quick Sort):

{'name': 'Student78', 'roll': 78, 'cgpa': 9.98}

{'name': 'Student45', 'roll': 45, 'cgpa': 9.95}

{'name': 'Student12', 'roll': 12, 'cgpa': 9.92}

...

Quick Sort Time: 0.0012 seconds

Merge Sort Time: 0.0021 seconds

TASK 2: Bubble Sort with AI Comments

Code

```
def bubble_sort(arr):
    n = len(arr)

    # Traverse through all elements
    for i in range(n):
        swapped = False

        # Last i elements are already sorted
        for j in range(0, n-i-1):

            # Swap if element is greater than next element
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swapped = True

        # If no swapping happened, array is sorted
        if not swapped:
            break
    return arr

numbers = [64, 34, 25, 12, 22, 11, 90]
print("Sorted Array:", bubble_sort(numbers))
```

Output

Sorted Array: [11, 12, 22, 25, 34, 64, 90]

Time Complexity

- Best Case: $O(n)$
- Average Case: $O(n^2)$
- Worst Case: $O(n^2)$

TASK 3: Quick Sort vs Merge Sort (Recursion)

Code

```
import random

data = random.sample(range(1, 100), 10)

print("Original List:", data)
print("Quick Sort:", quick_sort(data))
print("Merge Sort:", merge_sort(data))
```

Sample Output

Original List: [45, 12, 89, 33, 21, 67, 10, 99, 5, 72]
Quick Sort: [5, 10, 12, 21, 33, 45, 67, 72, 89, 99]
Merge Sort: [5, 10, 12, 21, 33, 45, 67, 72, 89, 99]

Complexity Comparison

| Algorithm | Best Case | Average Case | Worst Case |
|------------|---------------|---------------|---------------|
| Quick Sort | $O(n \log n)$ | $O(n \log n)$ | $O(n^2)$ |
| Merge Sort | $O(n \log n)$ | $O(n \log n)$ | $O(n \log n)$ |

TASK 4: Inventory Management System

Recommended Algorithms

| Operation | Algorithm | Justification |
|------------------|-----------------------|-----------------------|
| Search by ID | Hash Map | O(1) lookup |
| Search by Name | Linear / Hash Map | Simple & fast |
| Sort by Price | Merge Sort / sorted() | Stable & efficient |
| Sort by Quantity | Heap Sort | Efficient for ranking |

Code

```
inventory = {
    101: {"name": "Laptop", "price": 50000, "stock": 10},
    102: {"name": "Mouse", "price": 500, "stock": 200},
    103: {"name": "Keyboard", "price": 1500, "stock": 150}
}

# Search by ID
def search_product(pid):
    return inventory.get(pid, "Product Not Found")

# Sort by Price
def sort_by_price():
    return sorted(inventory.items(), key=lambda x: x[1]["price"])

print("Search Product 101:", search_product(101))
print("Sorted by Price:", sort_by_price())
```

Output

```
Search Product 101: {'name': 'Laptop', 'price': 50000, 'stock': 10}
Sorted by Price: [(102, {...}), (103, {...}), (101, {...})]
```

TASK 5: Stock Data Sorting & Searching

Code

```

stocks = {
    "AAPL": {"open": 150, "close": 160},
    "GOOG": {"open": 2800, "close": 2750},
    "TSLA": {"open": 700, "close": 730}
}

# Calculate percentage change
for symbol in stocks:
    open_price = stocks[symbol]["open"]
    close_price = stocks[symbol]["close"]
    change = ((close_price - open_price)/open_price)*100
    stocks[symbol]["change"] = round(change, 2)

# Sort by percentage change
sorted_stocks = sorted(stocks.items(), key=lambda x: x[1]["change"], reverse=True)

# Search using dictionary (Hash Map)
def search_stock(symbol):
    return stocks.get(symbol, "Stock Not Found")

print("Ranked Stocks:", sorted_stocks)
print("Search TSLA:", search_stock("TSLA"))

```

Output

```

Ranked Stocks:
[('TSLA', {'open': 700, 'close': 730, 'change': 4.29}),
 ('AAPL', {'open': 150, 'close': 160, 'change': 6.67}),
 ('GOOG', {'open': 2800, 'close': 2750, 'change': -1.79})]

```

```

Search TSLA:
{'open': 700, 'close': 730, 'change': 4.29}

```