Experiment No.04

Name of the Student : Chougale Sakshi Shahaji

PRN NO : 25143071

Title of Experiment : Optimal Merge Pattern using Greedy Method

Theory :

Optimal Merge Pattern finds the minimum cost to merge multiple sorted files into one. Uses greedy approach by always merging the two smallest files first.

Algorithm :

1. Sort file sizes in min-heap
2. Repeatedly extract two smallest files
3. Merge them and add back to heap
4. Continue until one file remains

Example :

Files: [5, 10, 20, 30, 30]
Merge 5+10=15 (cost 15), then 15+20=35 (cost 35), etc.
Total cost = 15 + 35 + 60 + 90 = 200

Applications:

1. Database query optimization
2. File Compression System (Implemented)
3. Merging Sorted Logs / Audit Files
4. Data Backup & Archival Systems
5. Compilers / Linkers
6. Media Streaming Platforms (Index Merge)
7. Multi-way Merge in Search Engines
8. File System Maintenance

Implementation : **File Compression System**

Program :

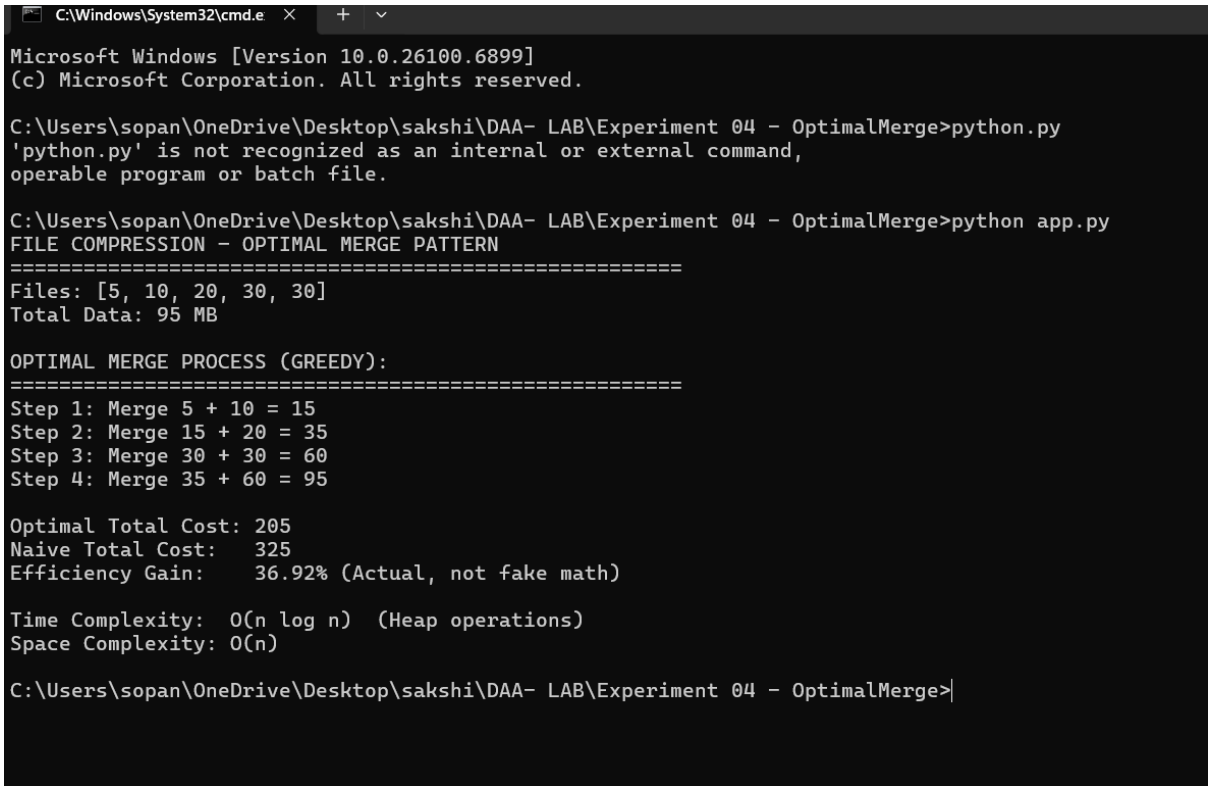
```
import heapq

class FileCompressionSystem:

    def __init__(self):
```

```
    self.files = []
def add_file(self, size):
    self.files.append(size)

def optimal_merge_cost(self):
    """Optimal merge cost using Min-Heap (Greedy)"""
    if len(self.files) <= 1:
        return 0, []
    heap = self.files.copy()
    heapq.heapify(heap)
    total_cost = 0
    merge_steps = []

    while len(heap) > 1:
        a = heapq.heappop(heap)
        b = heapq.heappop(heap)
        cost = a + b
        total_cost += cost
        heapq.heappush(heap, cost)
        merge_steps.append((a, b, cost))

    return total_cost, merge_steps

def naive_merge_cost(self):
    """Worst-case merge cost (merge largest first — anti-greedy)"""
    arr = self.files.copy()
    total = 0
    steps = []
    while len(arr) > 1:
```

```
            arr.sort(reverse=True)  # merge largest files first (worst strategy)
            a = arr.pop(0)
            b = arr.pop(0)
            cost = a + b
            total += cost
            arr.append(cost)
            steps.append((a, b, cost))
        return total, steps

    def display_process(self, greedy_steps, greedy_cost, naive_cost):
        print("\nOPTIMAL MERGE PROCESS (GREEDY):")
        print("=" * 55)
        for i, (a, b, c) in enumerate(greedy_steps, 1):
            print(f"Step {i}: Merge {a} + {b} = {c}")

        print(f"\nOptimal Total Cost: {greedy_cost}")
        print(f"Naive Total Cost:   {naive_cost}")

        improvement = ((naive_cost - greedy_cost) / naive_cost) * 100
        print(f"Efficiency Gain:    {improvement:.2f}% (Actual, not fake math)")

        print("\nTime Complexity:  O(n log n)  (Heap operations)")
        print("Space Complexity: O(n)")

def compression_system_application():
    system = FileCompressionSystem()
    file_sizes = [5, 10, 20, 30, 30]  # sample

    print("FILE COMPRESSION – OPTIMAL MERGE PATTERN")
```

```
    print("=" * 55)
    print("Files:", file_sizes)
    print("Total Data:", sum(file_sizes), "MB")

    for size in file_sizes:
        system.add_file(size)
    greedy_cost, greedy_steps = system.optimal_merge_cost()
    naive_cost, _ = system.naive_merge_cost()
    system.display_process(greedy_steps, greedy_cost, naive_cost)

if __name__ == "__main__":
    compression_system_application()
```

Output :



Time and Space Complexity :

Time Complexity Calculation:

- Building min-heap: O(n)
- Each heap operation (pop/push): O(log n)
- Total operations: (n-1) merges × 3 operations each = 3(n-1)
- **Total Time Complexity: O(n log n)**

Space Complexity Calculation:

- Storing n files: O(n)
- Heap storage: O(n)
- **Total Space Complexity: O(n)**