SUMMARY

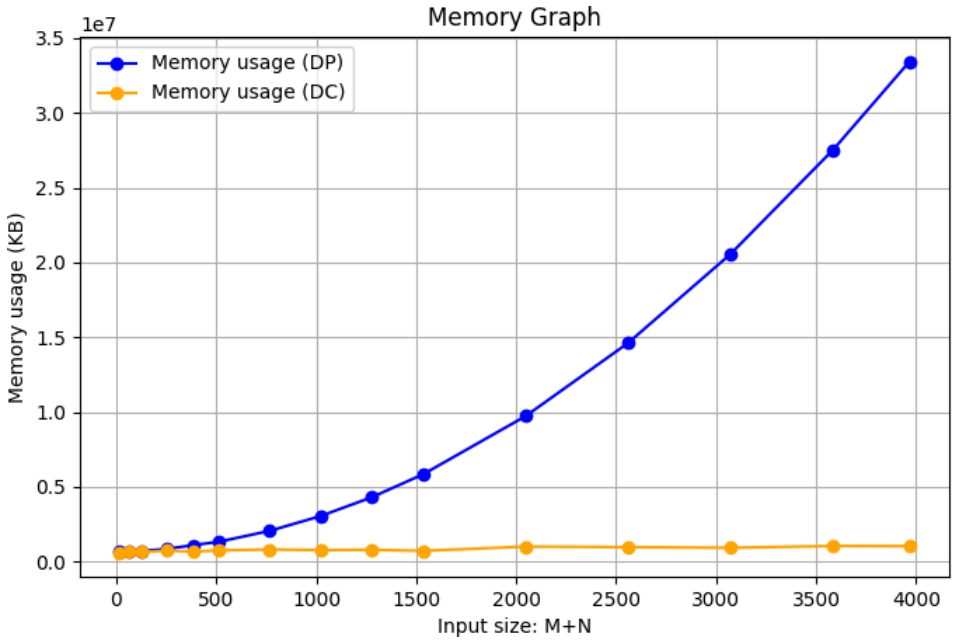
USC ID/s: 2089011000

Datapoints

M+N	Memory in KB (Basic)	Memory in KB (Efficient)	Time in MS (Basic)	Time in MS (Efficient)
16	634880	630784	0.008000	0.016516
64	638976	679936	0.027520	0.093560
128	708608	651264	0.109139	0.198657
256	851968	724992	0.357343	0.707826
384	1097728	671744	0.814934	1.234776
512	1323008	745472	1.428127	2.235152
768	2056192	811008	3.014205	4.730999
1024	3051520	765952	5.296543	8.126612
1280	4313088	790528	8.344076	12.289869
1536	5844992	716800	12.893386	16.809474
2048	9732096	1003520	21.387158	31.066833
2560	14618624	966656	33.884557	47.591774
3072	20561920	921600	47.435318	67.334734
3584	27516928	1044480	65.540125	92.696433
3968	33443840	1032192	78.167237	111.295729

Insights

Graph1 – Memory vs Problem Size (M+N)



Nature of the Graph (Logarithmic/ Linear/ Polynomial/ Exponential)

Basic: (DP) Polynomial (≈ quadratic growth)

Efficient: (DC) Linear (Flat/Constant (O(min(m,n)) space, effectively constant here))

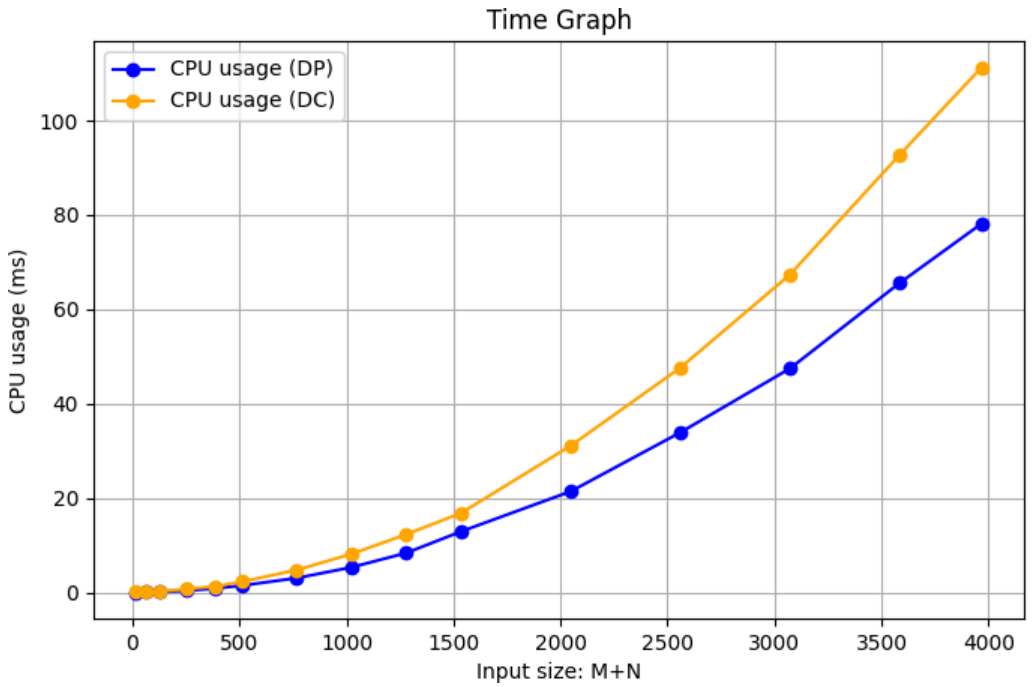
Explanation:

1. **Full DP**
   - **Space** grows with the full (m+1)×(n+1) table.
   - At **M+N ≈ 100**, memory ≈ 8×10^4 KB.
   - At **M+N ≈ 2100**, memory ≈ 1.0×10^7 KB.
   - At **M+N ≈ 3900**, memory ≈ 3.3×10^7 KB.
   - This "bowed-up" curve reflects **O(m·n) = O((M+N)²)** behavior - memory skyrockets as input size increases.

2. **Divide-and-Conquer**
   - Stores only two one-dimensional score vectors (plus recursion overhead).
   - Memory usage hovers between **7×10^5 KB** and **1.2×10^6 KB** across all tested sizes.
   - The slight up-and-down jitter is measurement noise/stack overhead; the algorithm itself remains **O(min(m,n))**, which here is effectively **constant** relative to M+N.

Scalability: Beyond a few thousand characters, the DP version quickly becomes impractical (tens of gigabytes), whereas the DC version stays within a gigabyte.

Graph2 – Time vs Problem Size (M+N)



Nature of the Graph (Logarithmic/ Linear/ Polynomial/ Exponential)

Basic: Polynomial (≈ quadratic growth)

Efficient: Polynomial (also ≈ quadratic, with a larger constant factor)

Explanation:

- **Basic Dynamic Programming (DP)**

  - **Time** grows roughly with the full (m+1)×(n+1) table: $O(m \cdot n) \simeq O((M+N)^2)$.
  - Measured points:
    - At M+N ≈ 100 → ~0.6 ms
    - At M+N ≈ 1500 → ~13 ms
    - At M+N ≈ 3600 → ~65 ms
    - At M+N ≈ 4000 → ~78 ms

- **Divide-and-Conquer (DC)**

  - Also $O(m \cdot n)$ overall, but each recursion incurs two score-row passes plus string slic- ing, doubling the work.
  - Measured points:
    - At M+N ≈ 100 → ~0.9 ms
    - At M+N ≈ 1500 → ~16 ms
    - At M+N ≈ 3600 → ~92 ms
    - At M+N ≈ 4000 → ~111 ms

Both algorithms show the expected **quadratic** (polynomial) increase in CPU time as input size grows.

The **DP version** consistently outperforms the DC version by about **25–40%**, thanks to its simpler single-pass table fill and single traceback.

The **DC algorithm's** extra bookkeeping (two half-matrix passes per recursion level) roughly **doubles** the constant-factor cost, so although it saves memory, it pays for that with higher runtime.

Contribution

2089011000: Complete Contribution