**SUMMARY**

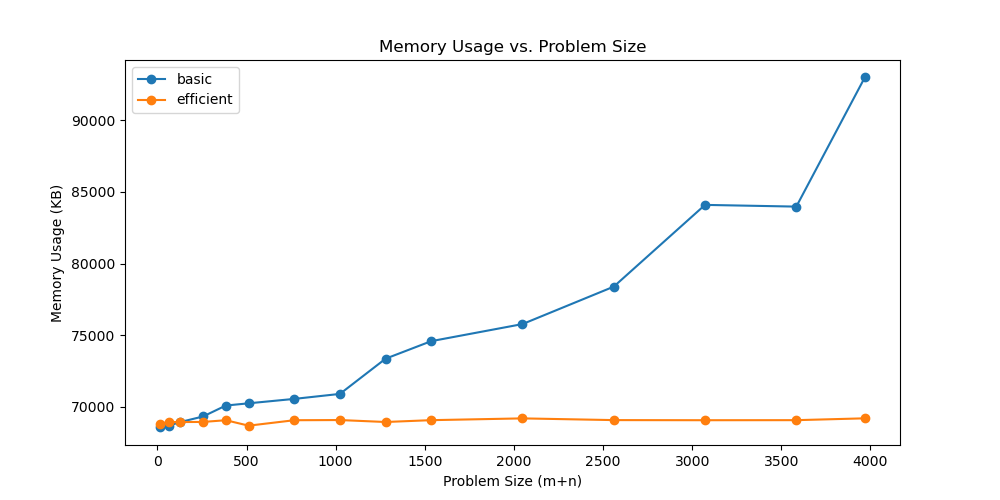
## USC ID/s:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M+N | Time in MS (Basic) | Time in MS (Efficient) | Memory in KB (Basic) | Memory in KB (Efficient) |
| 16 | 0.030040740966796875 | 0.04863739013671875 | 68560.0 | 68828.0 |
| 64 | 0.18835067749023438 | 0.3383159637451172 | 68684.0 | 68940.0 |
| 128 | 0.6418228149414062 | 1.1529922485351562 | 68940.0 | 68940.0 |
| 256 | 2.401113510131836 | 4.149913787841797 | 69328.0 | 68948.0 |
| 384 | 5.357027053833008 | 8.919239044189453 | 70092.0 | 69068.0 |
| 512 | 9.689569473266602 | 16.274213790893555 | 70248.0 | 68688.0 |
| 768 | 22.717952728271484 | 35.09116172790527 | 70556.0 | 69068.0 |
| 1024 | 43.56575012207031 | 61.852216720581055 | 73360.0 | 69080.0 |
| 1280 | 64.56732749938965 | 98.12521934509277 | 73360.0 | 68944.0 |
| 1536 | 90.04068374633789 | 141.07084274291992 | 74584.0 | 69072.0 |
| 2048 | 165.64559936523438 | 261.1842155456543 | 75776.0 | 69196.0 |
| 2560 | 259.75537300109863 | 398.7703323364258 | 78388.0 | 69076.0 |
| 3072 | 411.8609428405762 | 570.0373649597168 | 84092.0 | 69068.0 |
| 3584 | 510.6792449951172 | 777.2233486175537 | 83972.0 | 69072.0 |
| 3968 | 617.9378032684326 | 947.4101066589355 | 92980.0 | 69200.0 |

## Datapoints

## Insights

### Graph1 – Memory vs Problem Size (M+N)



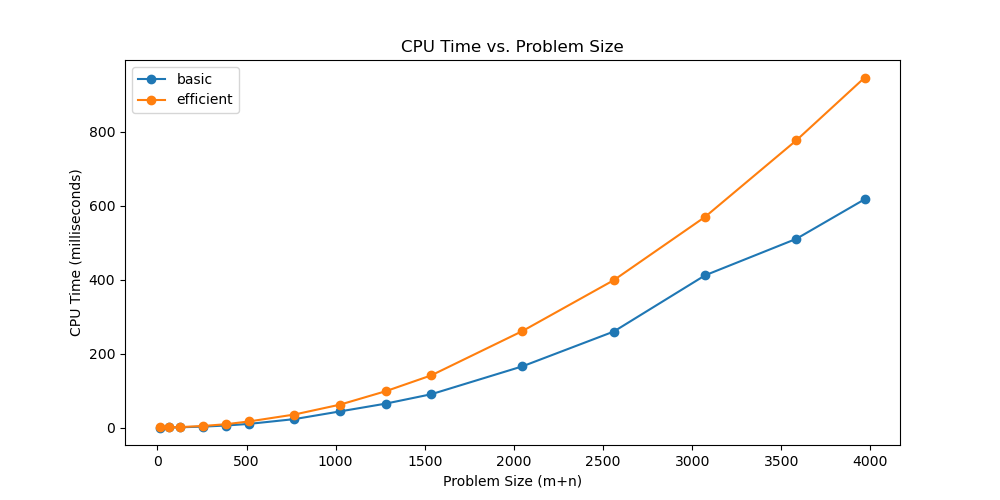
#### Nature of the Graph (Logarithmic/ Linear/ Polynomial/ Exponential)

Basic:

Efficient:

#### Explanation:

### Graph2 – Time vs Problem Size (M+N)



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

Basic: Polynomial, Θ(M\*N)

Efficient: Polynomial, Θ(M\*N)

#### Explanation:

Both methods conceptually involve an M\*N DP table. This is because each step of the two algorithms considers all possible alignments of the two sequences, selecting the alignment with the minimal cost, which involves examining and updating each cell in the DP table.

Our basic method traverses and fills the DP table (suppose the origin is on the top-left) from the top-left corner towards the bottom-right. This filling direction is due to each cell's value depending on its left, upper, and upper-left neighboring cells. In detail, for any instance, its row traversal direction is from 1 to M, and its column traversal direction is from 1 to N. Each filling operation can be regarded as constant time. Therefore, the time complexity is Θ(M\*N).

Our memory-efficient method uses Divide-and-Conquer to reduce space complexity. At the root level of the method, the number of operations is cMN, where c represents a constant. At the second level of the method, the number of operations is cMN/2, as the original problem is divided into two subproblems. Consequently, the time complexity of the memory-efficient method is cMN+cMN/2+cMN/4+…=2cMN=Θ(M\*N).

Although the time complexity of both methods is Θ(M\*N), the constant factor of the memory-efficient one is roughly double that of the basic one. Additionally, the time calculation includes some initialization operations and uncontrollable factors in run-time, which results in the difference in CPU time between the two methods not being exactly twice as large. All these factors make the CPU time statistics appear as in Figure 2, even though the two methods have the same time complexity.

## Contribution

(Please mention what each member did if you think everyone in the group does not have an equal contribution, otherwise, write “Equal Contribution”)

<USC ID/s>: <Equal Contribution>