**SUMMARY**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M+N | Time in MS (Basic) | Time in MS (Efficient) | Memory in KB (Basic) | Memory in KB (Efficient) |
| 16 | 0.0000 | 0.0000 | 22584.0000 | 22808.0000 |
| 64 | 0.8969 | 1.6780 | 22640.0000 | 22760.0000 |
| 128 | 0.9987 | 2.0061 | 22748.0000 | 22836.0000 |
| 256 | 2.7406 | 7.0121 | 23252.0000 | 22768.0000 |
| 384 | 7.3285 | 16.0160 | 24088.0000 | 22900.0000 |
| 512 | 14.2281 | 27.5493 | 25272.0000 | 22936.0000 |
| 768 | 31.1458 | 60.3492 | 28496.0000 | 22760.0000 |
| 1024 | 59.9110 | 108.4754 | 33468.0000 | 22804.0000 |
| 1280 | 93.4994 | 177.5794 | 39016.0000 | 22832.0000 |
| 1536 | 135.7679 | 248.9016 | 46552.0000 | 23196.0000 |
| 2048 | 250.0134 | 455.3485 | 65368.0000 | 23120.0000 |
| 2560 | 385.5910 | 730.0720 | 88616.0000 | 23068.0000 |
| 3072 | 561.6295 | 1013.8209 | 116384.0000 | 23080.0000 |
| 3584 | 785.0158 | 1452.1732 | 153016.0000 | 23208.0000 |
| 3968 | 946.2602 | 1807.0965 | 179296.0000 | 23316.0000 |

## Datapoints

## Insights

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

A graph with a line and a blue line

Description automatically generated

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

Basic: Polynomial

Efficient: Linear

#### Explanation:

### As we would expect there is a stark contrast between the two algorithms’ abilities to conserve memory as the size of our input grows. The whole point of the Efficient algorithm was to lower the needed memory for more complex problems. Because the axes are not normalized, it is harder to clearly see trends, but the returned output data presented in our graph as the Efficient algorithm reveals a linear amount of memory usage even as the problem size grows. There is still an increase in memory usage as the input problem size increases, so we say that the efficient algorithm produces a linear space complexity of O(m), where m is the size of one of the input strings. This makes sense since the most memory intensive part of the efficient algorithm is when we create an m by 2 array when we divide the problem to find the optimal split point (m scales with respect to one of the string’s length).

### Note that the linearity becomes more apparent as problem size grows since the asymptotic relationship is defined when m approaches infinity.

### On the other hand we see that the basic algorithm’s memory usage grows at a polynomial rate as the size of the input problem grows so its space complexity would be O(m\*n) or quadratic if the sizes of the 2 input strings are equal. In the basic algorithm, the most memory intensive part of the algorithm is holding the entire contiguous m by n array in memory and storing all the values of the optimal solutions in order to do the top down pass to find the alignment.

### It’s fairly obvious that it would beneficial to move away from using the Basic algorithm and only use the Efficient algorithm due to how great a difference the space complexities are at larger problem sizes.

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

A graph with a line and a line

Description automatically generated

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

Basic: Polynomial

Efficient: Polynomial

#### Explanation:

## Strictly looking at the graph itself we can see that both graphs maintain a polynomial growth rate as the problem size increases. Delving more into the data itself the Efficient algorithm grows at roughly double the rate that the Basic algorithm grows.

## For our basic algorithm we’re getting a time complexity of O(mn). This is due to the fact that the time intensive portion of the algorithm involves the bottom up pass to fill each value of the m by n array, for O(mn). The top down pass is only O(m+n) and is dominated by O(mn).

## Conversely, the Efficient algorithm would be something closer to O(2mn). As described in lecture, the first divide step is the most expensive as it is the cost of the entire basic algorithm. However, in subsequent recursive calls, we’re discarding irrelevant sub problems each time thus drastically reducing problem size.

## It’s not a huge difference with smaller problem sizes but as shown it can make computing larger sequence alignments much more time consuming. Therefore, we can conclude that the Basic algorithm performs better with larger problem sizes if your optimization goal is to reduce CPU time.