COT 6417 - Algorithms on Strings and Sequences -Fall 2021
Homework Assignment 3
Kusma Thummagunta – 5335477

Hirschberg's algorithm :

1) **Description :**

Given strings X and Y i.e |X| = n and |Y| = m also given (m<n) . To compute the optimal global alignment cost in linear time, we need to fill out (n x m) matrix M, we need information present in the current row and the previous row but by just doing this, we cannot construct an optimal alignment directly. However, hirschberg's algorithm allows us to construct an optimal global alignment in O(nm) space but using only O(m) space.

Let M[i,j] denote the maximum score of aligning string X[1,i] with string Y[1,j] . Let X^R and Y^R denote the reverse of strings X and Y . Let M^R[i,j] denote the maximum score of aligning string X^R[1,i] with string Y^R[1,j] . Optimal cost of aligning X and Y is same as the optimal cost of aligning X^R and Y^R

Recursive approach to obtain an alignment for with X[1,n] Y[1,m]

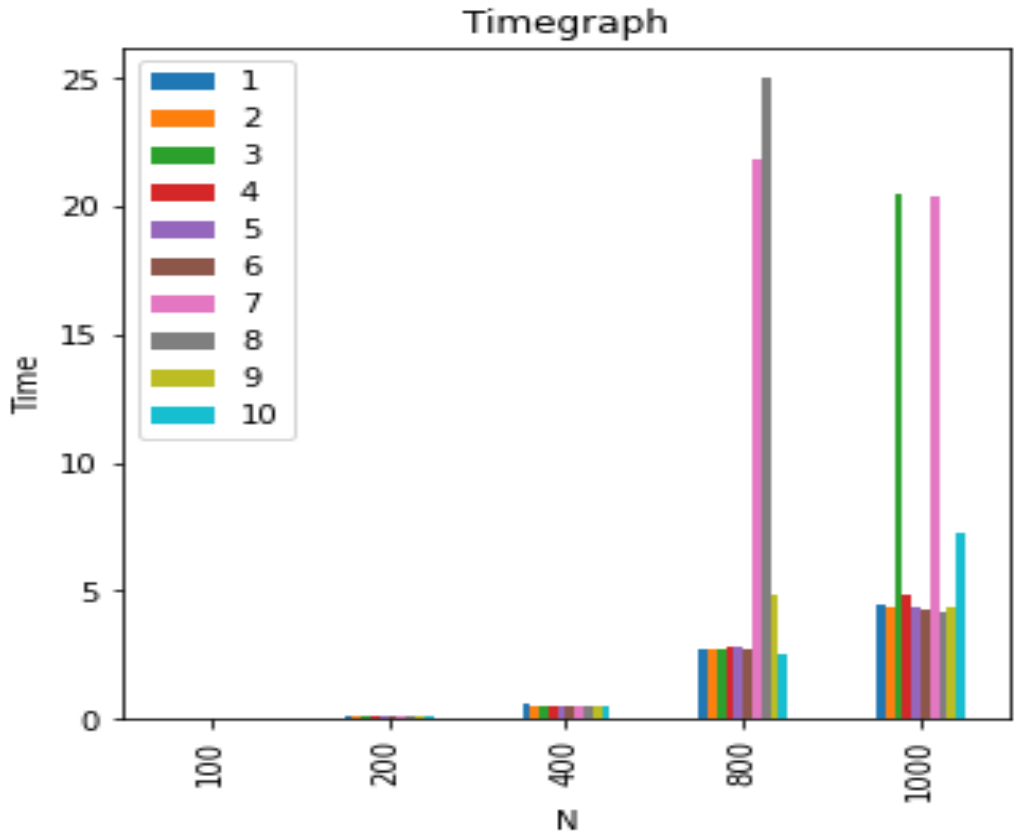
- We need to identify index in string Y such that k* M[n, m] = M[(n/2), k*]+ M^R[(n/2), m−k*]
- In other words, identify index k* in string Y such that is M[(n/2), k*]+ M^R[(n/2), m−k*] maximum
- For both M and M^R we only need the last row (which is the $(n/2)^{th}$ row)
- We can find k* in O(nm) time using O(m) space.
- We will cut the string Y at this index value k* and then
- Construct the alignment for X[1,(n/2)] with Y[1, k*]
- Construct the alignment for X[(n/2) + 1,n] with Y[k* + 1,m]
- Combine these 2 disjoint (i.e non overlapping) sets of alignments to obtain the alignment for X[1,n] with Y[1,m]

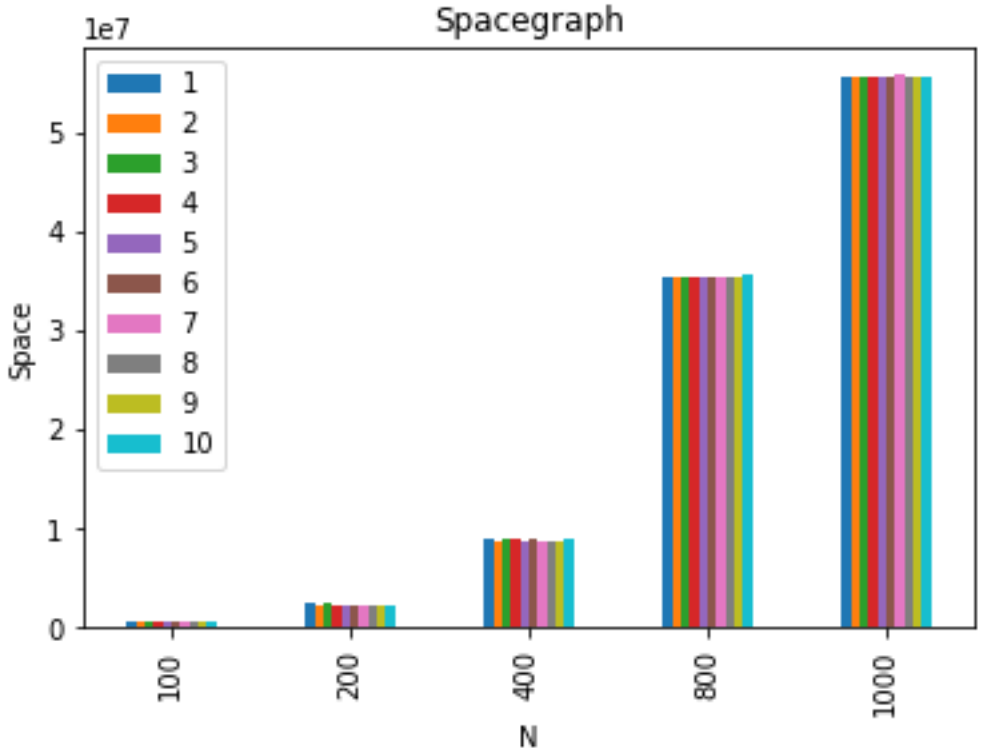
2) **Pseudo -code :**

```
def main(N):
        #start time and space

        X, Y = generaterandomString of value N
        Y^R = Y[::-1]
        |X| = n, |Y| =m
        x = X [:n//2 + 1] , x' = X [n//2 + 1:]
        x^R = x [::-1]  , (x')^R = x' [::-1]
        G = globalAlign(x, Y)
        G^R = globalAlign((x')^R, Y^R)
        Gsum = G[-1] and G^R [-1]
        max1 = max(Gsum)
        K* = Gsum.index(max1)
        y = Y[:k*+1] ,  y'= Y[k*+1:]
        G1 = globalAlign(x, y)
        G2 = globalAlign(x', y')

    return time, space
(Global align function to compute the matrix using match, gap , insertion, deletion costs)
```

COT 6417 - Algorithms on Strings and Sequences -Fall 2021
Homework Assignment 3
Kusma Thummagunta – 5335477

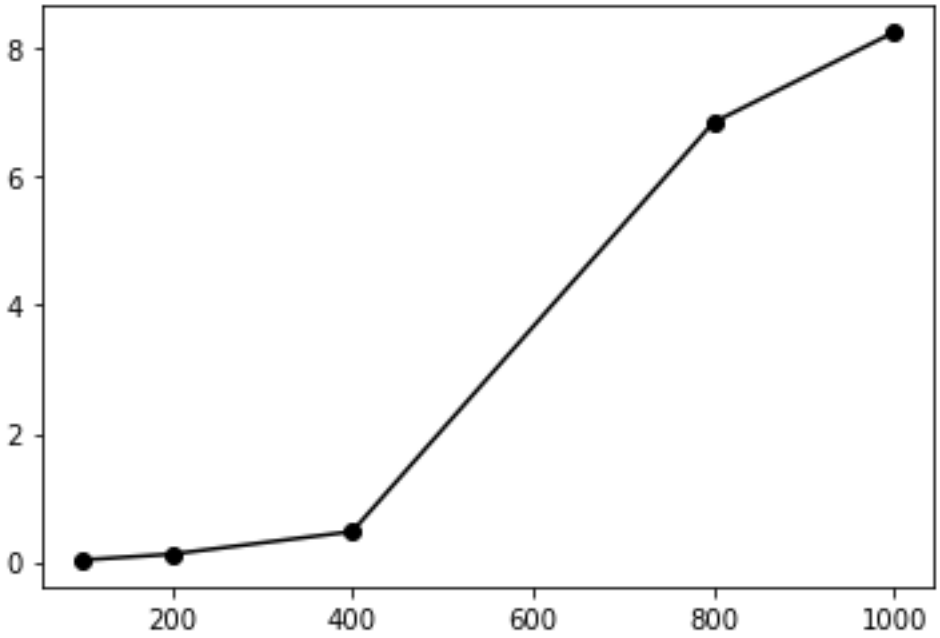
3) Simulation Results :



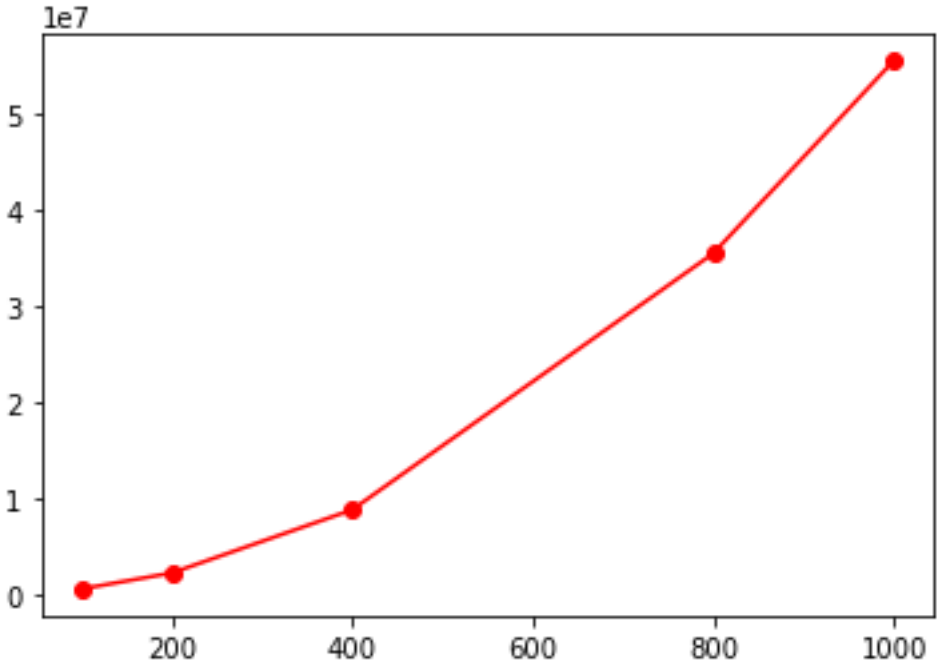


COT 6417 - Algorithms on Strings and Sequences -Fall 2021
Homework Assignment 3
Kusma Thummagunta – 5335477

Graphs of the Average time Values vs N values



Graph of the Average space Values vs N values



COT 6417 - Algorithms on Strings and Sequences -Fall 2021
Homework Assignment 3
Kusma Thummagunta – 5335477

```
In [55]: runfile('C:/Users/kusma/Desktop/hw_3.py', wdir='C:/Users/kusma/Desktop')
N = 100, iterations = 10

Average time taken for 100 : 0.027716231346130372
Average space taken for 100 : 603859.4

N = 200, iterations = 10

Average time taken for 200 : 0.11952109336853027
Average space taken for 200 : 2263427.4

N = 400, iterations = 10

Average time taken for 400 : 0.5121010303497314
Average space taken for 400 : 8840125.1

N = 800, iterations = 10

Average time taken for 800 : 5.391849327087402
Average space taken for 800 : 35503795.5

N = 1000, iterations = 10

Average time taken for 1000 : 9.117580723762511
Average space taken for 1000 : 55543227.1
```

4)

Yes, the evaluations definitely match with the theory as we can see from the first graph demonstrating time taken for each iteration of N values change depending on the random string generated while the memory graph plotted against each iteration is same for a given N value. It takes linear space regardless of the given string. I even demonstrated the avg time graph next by plotting the avg time taken for a given N and we can see it grows with the size again with respect to the string , sometimes can take lesser time for a large of N or takes longer time for smaller N value (refer to first graph comparing the 8^{th} iteration(grey line) for N =800,1000) Whereas, the avg space graph calculated for each value of N grows **linearly.** Using these results, I can conclude that the implementation obeys the theory.