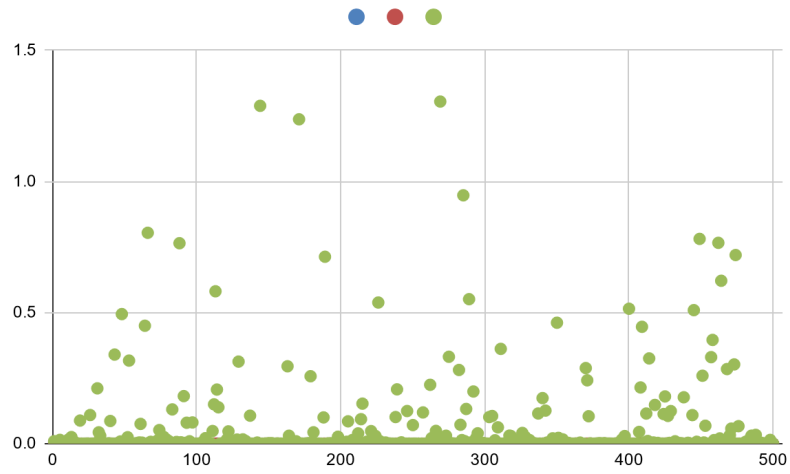
Comparison of Binomial Algorithms

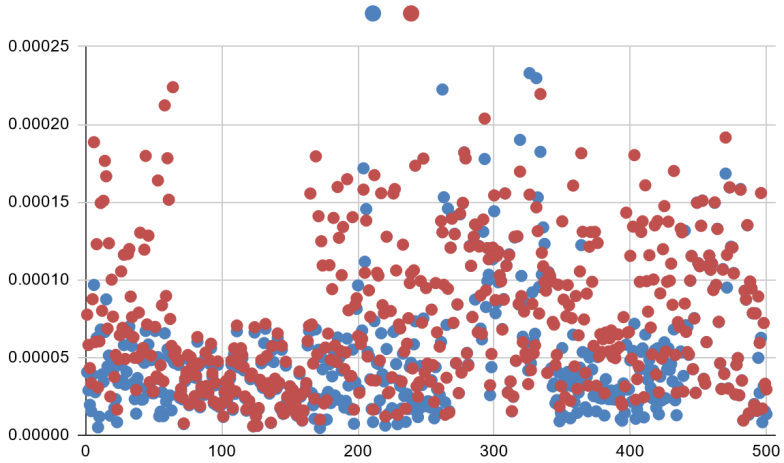
The experiment consisted of several key features. These features include three different binomial coefficient algorithms; a divide-and-conquer algorithm, bottom-up dynamic algorithm, and a memoization algorithm. Alongside these algorithms, a way to generate random binomial coefficient problems and output the results of each problem to a readable format was included as well.

This was achieved by creating a function called nkVals() that creates a list of 500 different tuples for (n,k) where n is less than 25. This does mean that the problems aren't as intensive, but in testing of the simple divide-and-conquer algorithm, it took upward of 1 second for each calculation where n approaches 30. To save on run-time, 25 was decided to be the limit. Even after this limitation, the divide-and-conquer algorithm was still significantly slower than the dynamic programming and memoization approaches.



As seen above, the graph is a scatterplot of the solve times for each of the 500 problems measured in seconds. The x axis is the problems from 1-500, and the y axis is the amount of time, in seconds, that it took for each algorithm to solve that problem. The green dots are the divide-and-conquer, red is memoization, and blue is dynamic programming. It is clear the

divide-and-conquer algorithm is the slowest, and dwarfs any meaningful comparison between the other two algorithms.



This is the same scatter plot without the divide-and-conquer algorithm, and while a little jumbled, it is clear that for any given problem, the dynamic programming approach is usually faster for any given problem.

The total runtime and averages for each of the algorithms are as follows;

Recursion Sum (s)	Dynam Sum (s)	Memo Sum (s)
26.7400539	0.0256575	0.0369153
Recursion Avg (s)	Dynam Avg (s)	Memo Avg (s)
0.0534801078	0.000051315	0.0000738306

As seen above, the dynamic programming approach is marginally faster than the memoization algorithm due to the fact that it has to solve less sub-problems than the other. The divide-and-conquer approach is clearly the slowest algorithm due to the fact that it has to solve each sub-problem over and over again for any given problem, drastically increasing run-time.

Computer Specifications

Processor Intel(R) Core(TM) i3-8145U CPU @ 2.10GHz 2.30 GHz

Installed RAM 4.00 GB (3.78 GB usable)

System type 64-bit operating system, x64-based processor

Binomial.py

```
import xlwt
from xlwt import Workbook

import random
from random import seed, randint

import time

# Sets seed
seed(42)


# Basic Recursion
def binCoeffRecursion(n, k):

    # Base Cases
    if k==0 or k==n:
        return 1

    return binCoeffRecursion(n-1, k) + binCoeffRecursion(n-1, k-1)


# Dynamic Bottom-Top
def binCoeffDynamic(n, k):
    C = [[0 for x in range(k+1)] for x in range(n+1)]

    for i in range(n+1):
        for j in range(min(i,k)+1):

            # Base Cases
            if j==0 or j==i:
                C[i][j]=1

            C[i][j] = C[i-1][j-1]+C[i-1][j]

    return C[n][k]
```

```
# Both functions below use Top-Bottom Memoization approach
def binCoeffMemoTable(n, k, dupe):

    # Checks if value is in look up table
    # If so, return that value
    if dupe[n][k] != -1:
        return dupe[n][k]

    # Base Cases
    if k==0 or k==n:
        dupe[n][k]=1
        return dupe[n][k]

    # Else, put new value in table
    dupe[n][k] = binCoeffMemoTable(n-1,k-1, dupe) + binCoeffMemoTable(n-1,
k, dupe)

    return dupe[n][k]


def binCoeffMemo(n,k):
    dupe = [[-1 for i in range(k+1)] for j in range(n+1)]
    return binCoeffMemoTable(n,k,dupe)


# Generates a list of 500 pairings of n and k where n<25
def nkVals():

    # Declare variables
    lst = []
    nk = [0,0]

    # Get two random numbers from 2 to 25
    # Put them in the nk list and add that list to the end of lst
    # Repeat until 500 data points
    for i in range(500):
        vals = randint(2,25), randint(2,25)
        nk[0] = max(vals)
        nk[1] = min(vals)
        print(nk)
```

```
        lst.append(nk[:])
    print(lst)


    return lst



# The below three functions all compute the time it takes for each
binomial call
def recurTime(wb, lst):

    counter = 0
    # Creates Excell Sheets
    sheet1.write(1, 1, "Recursion Times (s)")

    # Calculates the time it takes for each function call and adds them to
the data sheet
    for i in range(len(lst)):

        t0 = time.perf_counter()
        binCoeffRecursion(lst[i][0],lst[i][1])
        t1 = time.perf_counter()

        counter += 1

        timeTaken = t1 - t0
        sheet1.write(counter+1, 1, timeTaken)

def dynamTime(wb, lst):

    counter = 0
    # Creates Excel Sheets
    sheet1.write(1,2,"Dynamic Times (s)")

    # Calculates the time it takes for each function call and adds them to
the data sheet
    for i in range(len(lst)):
```

```
        t0 = time.perf_counter()
        binCoeffDynamic(lst[i][0],lst[i][1])
        t1 = time.perf_counter()

        counter += 1

        timeTaken = t1 - t0
        sheet1.write(counter+1, 2, timeTaken)

def memoTime(wb, lst):

    counter = 0
    # Creates Excel Sheets
    sheet1.write(1,3, "Memoization Times (s)")
    # Calculates the time it takes for each function call and adds them to
the data sheet
    for i in range(len(lst)):

        t0 = time.perf_counter()
        binCoeffDynamic(lst[i][0],lst[i][1])
        t1 = time.perf_counter()

        counter += 1

        timeTaken = t1 - t0
        sheet1.write(counter+1, 3, timeTaken)

# Driver Code
if __name__ == "__main__":

    wb = Workbook()
    sheet1 = wb.add_sheet('Sheet 1', cell_overwrite_ok= True)



    lst = nkVals()

    t0 = time.perf_counter()
    dynamTime(wb, lst)
    t1 = time.perf_counter()
```

```
    print("Dynamic done in " + str(t1-t0) + " seconds!")
    t0 = time.perf_counter()
    memoTime(wb, lst)
    t1 = time.perf_counter()
    print("Memo done in " + str(t1-t0) + " seconds!")
    t0 = time.perf_counter()
    recurTime(wb, lst)
    t1 = time.perf_counter()
    print("Recursion done in " + str(t1-t0) + " seconds!")

    wb.save('BinomialData.xls')
```