CS 317 Lab 2

Drake Song

Analysis of addInt, multInt, divInt, addDoub, multDoub, and divDoub

Generally, ints should be more efficient than doubles in terms of both time and memory since int uses less space than doubles. However, the difference in efficiency, especially in terms of time, is so small that it cannot be observed by using such simple operations from this lab.

Looking at the different operations that were performed, theoretically, addition should take the least amount of time. However, similar to above, the difference is so trivial that it cannot really be observed. There are some discrepancies between the operations as seen on the figures attached on the next page but due to the unreasonable nature of these fluctuations in the data where there seem to be no certain pattern nor explanations, it is safe to conclude that these fluctuations are happening because of the hardware, other programs occupying processing block, and/or other reasons that's preventing the program from utilizing the constant resources throughout the lab.

One thing that is certain as observed from the box plots is that as n becomes larger, the confidence of the data increases, meaning that the variance of the data for each of the operations decreases. By comparing the box plot when n = 10 to any other box plot, it is clear that the variance of data when n = 100 is far greater than any other. The size and the range of the box and whisker is far larger when n is small. This occurs because as the number of times increases, the accuracy of the runtime becomes more precise as outliers have diminished effect on the averaged runtime.

Analysis of sine, pwr, and print

All three of these operations take considerable more time to run than the operations discussed above. This is mostly due to the fact that these operations are not simple addition, multiplication, and division.

Although the computational operations between sine and power functions are not too discernible in the numpy library, the reason power function takes slightly longer than the sine function is that the power function has two inputs instead of one as done for sine function. In fact, the power function has to take two numbers into consideration when running and these numbers affect each other (one is the base while the other is the power) compared to the sine function where it's just one input that runs through the sine function. Print, on the other hand, is a completely different story. Print itself takes quite a long time (compared to other operations used in this lab) and the fact that there is a loop (although it is a simple one) that involves two runs of the print operation adds considerable time to the runtime.

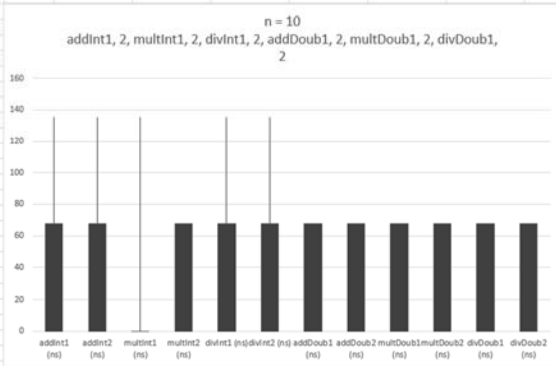
As mentioned above, as n increases, the precision of the data also improves. This is shown for these three operations too with the exception of print for n = 5000. The only explanation for this is that there is a big outlier or multiple outliers in the data that is skewing the shape of the box plot. As n increases, the box plots for all three operations grow smaller and smaller in size indicating that the data is becoming more accurate.

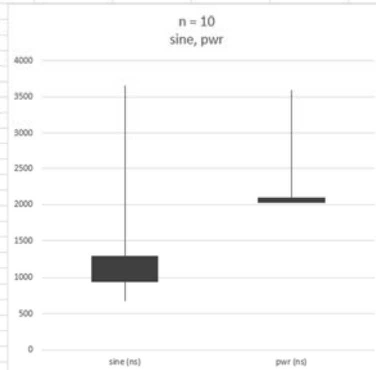
Gauss Analysis

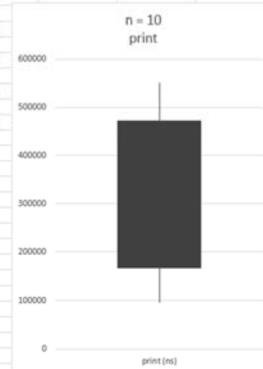
Theoretically, the runtime of Gaussian Elimination with Pivoting is $O(n^3)$. Looking at the algorithm step by step, let's say n = 6 (six columns and five rows). The algorithm first makes 5 comparisons to figure out the largest absolute value of the first column by going through the 5 rows. Then the algorithm swaps the rows. This swapping does not take too much time to run but rather consumes memory by creating a temp variable (memory is not a concern for this lab). The algorithm then goes to the rows below the first row and adds/subtracts after calculating the ratio that would make that row's first element 0 for every single row below the first. The entire process of this is then repeated 4 times so that a triangular matrix is outputted. To simply the algorithm further, there are 3 for-loops which results in a cubic runtime.

Plotting the log-log of the n vs runtime for Gaussian algorithm, the slope of the line is approximately 3. By fitting a cubic polynomial trendline on the actual data, it is clear to see that the trendline fits the data well. This confirms the fact that Gaussian algorithm has a cubic runtime. The coefficients, however, are fairly small. This can be explained by how the algorithm does not have a full cubic runtime. In fact, even though n = 6, the 3 for-loops are running 5 times (going through the 5 rows), 5 times (doing 5 compares of the absolute values), and 4 times (adding/subtracting the remaining rows to zero out the elements). To be clearer, the runtime should be O((n-1)(n-2)(n-2)). Because the runtime is O((n-1)(n-2)(n-2)) instead of O((n)(n)(n)), the coefficients for the cubic trendline are relatively small.

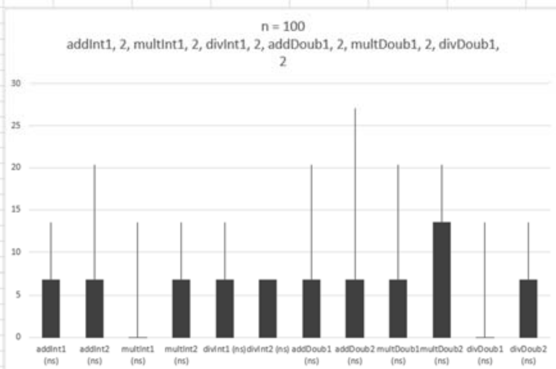
n = 10															
function	addInt1 (ns)	addInt2 (ns)	multInt1 (ns)	multInt2 (ns)	divInt1 (ns)	divInt2 (ns)	addDoub1 (ns)	addDoub2 (ns)	multDoub1 (ns)	multDoub2 (ns)	divDoub1 (ns)	divDoub2 (ns)	sine (ns)	pwr (ns)	print (ns)
Q3	67.72493653	67.72493653	0	67.72493653	67.72493653	67.72493653	67.72493653	67.72493653	67.72493653	67.72493653	67.72493653	67.72493653	1286.773794	2099.473032	472720.0569
Max	135.4498731	135.4498731	135.449873	67.72493653	135.449873	135.4498731	67.72493653	67.72493653	67.72493653	67.72493653	67.72493653	67.72493653	3657.146572	3589.421636	551348.7082
Min	0	0	0	0	0	0	0	0	0	0	0	0	677.2493652	2031.748096	96169.40987
Q1	0	0	0	0	0	0	0	0	0	0	0	0	948.1491113	2031.748096	167280.5932

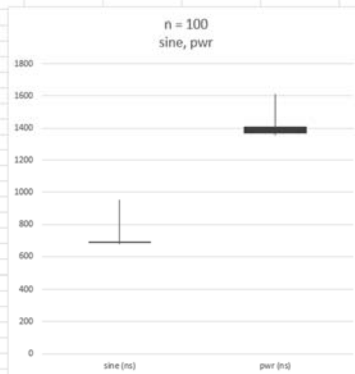


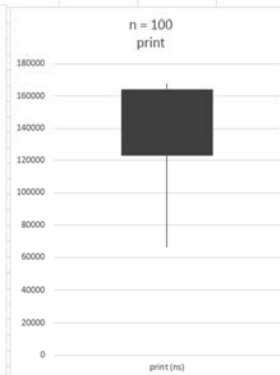




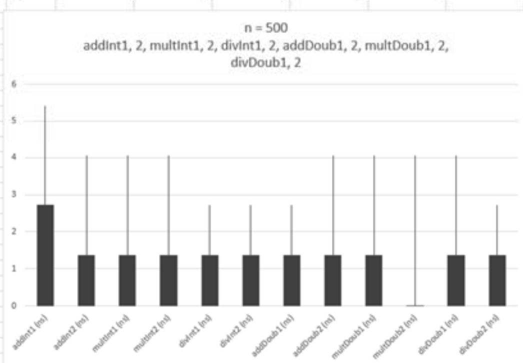
n=100															
function	addInt1 (ns)	addInt2 (ns)	multInt1 (ns)	multInt2 (ns)	divInt1 (ns)	divInt2 (ns)	addDoub1 (ns)	addDoub2 (ns)	multDoub1 (ns)	multDoub2 (ns)	divDoub1 (ns)	divDoub2 (ns)	sine (ns)	pwr (ns)	print (ns)
Q3	6.772493653	6.772493653	0	6.772493653	6.772493653	6.772493651	6.772493653	6.772493653	6.772493653	13.54498731	0	6.772493653	690.7943525	1408.67868	163968.8438
Max	13.54498731	20.31748096	13.54498731	13.54498731	13.54498731	6.772493653	20.31748096	27.08997461	20.31748096	20.31748096	13.5449873	13.54498731	954.921605	1611.853489	167923.9801
Min	0	0	0	0	0	0	0	0	0	0	0	0	677.249365	1354.49873	66377.21029
Q1	0	0	0	0	0	0	0	0	0	0	0	0	684.0218587	1368.043718	123008.8022

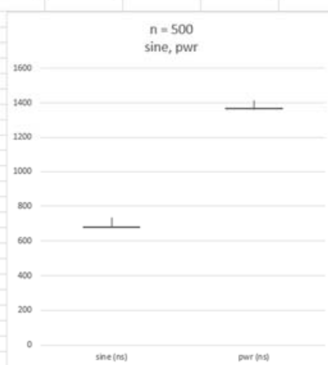


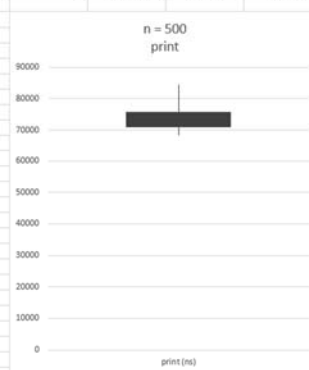




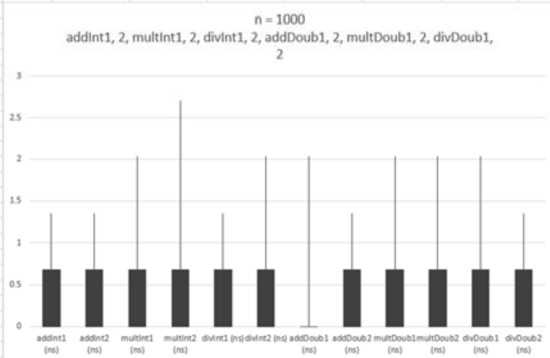
n=500															
function	addInt1 (ns)	addInt2 (ns)	multInt1 (ns)	multInt2 (ns)	divInt1 (ns)	divInt2 (ns)	addDoub1 (ns)	addDoub2 (ns)	multDoub1 (ns)	multDoub2 (ns)	divDoub1 (ns)	divDoub2 (ns)	sine (ns)	pwr (ns)	print (ns)
Q3	2.70899746	1.35449873	1.35449873	1.35449873	1.35449873	1.35449873	1.354498728	1.354498732	1.354498732	0	1.354498728	1.354498728	682.6673591	1369.398216	75604.05564
Max	5.41799492	4.06349619	4.06349619	4.063496192	2.70899746	2.708997462	2.70899746	4.063496192	4.063496192	4.063496192	4.063496192	2.70899746	734.1383109	1412.742175	84512.59379
Min	0	0	0	0	0	0	0	0	0	0	0	0	677.2493641	1361.271224	68304.66198
Q1	0	0	0	0	0	0	0	0	0	0	0	0	679.9583616	1363.980221	71057.0034

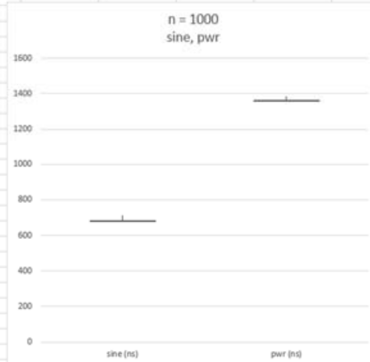




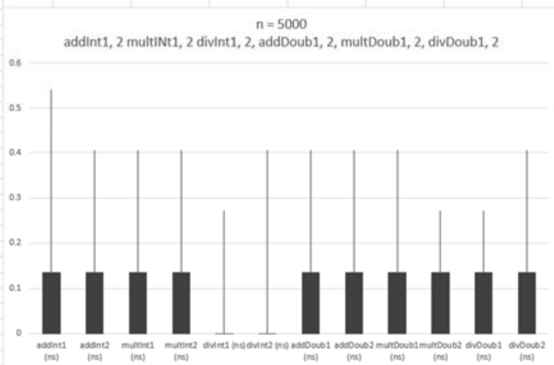


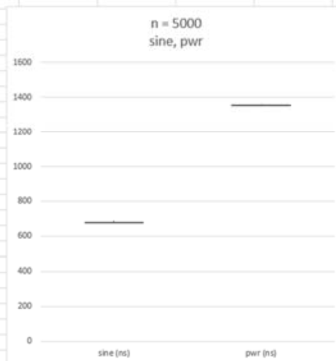
n=1000															
function	addInt1 (ns)	addInt2 (ns)	multInt1 (ns)	multInt2 (ns)	divInt1 (ns)	divInt2 (ns)	addDoub1 (ns)	addDoub2 (ns)	multDoub1 (ns)	multDoub2 (ns)	divDoub1 (ns)	divDoub2 (ns)	sine (ns)	pwr (ns)	print (ns)
Q3	0.677249364	0.677249366	0.677249364	0.677249364	0.677249368	0.677249364	0	0.677249364	0.677249364	0.677249364	0.677249364	0.677249364	681.9901062	1361.948471	65124.29896
Max	1.354498732	1.35449873	2.031748096	2.70899746	1.354498728	2.031748092	2.031748096	1.354498732	2.031748096	2.031748096	2.031748092	1.354498732	713.8208264	1384.2977	65657.29421
Min	0	0	0	0	0	0	0	0	0	0	0	0	677.2493606	1357.884975	55684.79731
Q1	0	0	0	0	0	0	0	0	0	0	0	0	679.2811087	1359.239474	64566.92273

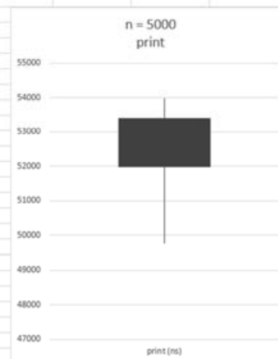




n=5000															
function	addInt1 (ns)	addInt2 (ns)	multInt1 (ns)	multInt2 (ns)	divInt1 (ns)	divInt2 (ns)	addDoub1 (ns)	addDoub2 (ns)	multDoub1 (ns)	multDoub2 (ns)	divDoub1 (ns)	divDoub2 (ns)	sine (ns)	pwr (ns)	print (ns)
Q3	0.135449875	0.135449875	0.135449869	0.135449864	0	0	0.135449875	0.135449875	0.135449875	0.135449875	0.135449864	0.135449864	678.0620173	1356.124035	53391.36006
Max	0.541799494	0.406349619	0.406349619	0.406349614	0.27089975	0.406349614	0.406349614	0.406349614	0.406349625	0.27089975	0.27089975	0.406349614	687.6789583	1362.35473	53977.04531
Min	0	0	0	0	0	0	0	0	0	0	0	0	677.249318	1354.904986	49766.04421
Q1	0	0	0	0	0	0	0	0	0	0	0	0	677.5202178	1355.175886	51984.03588

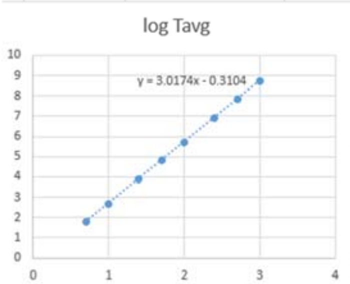


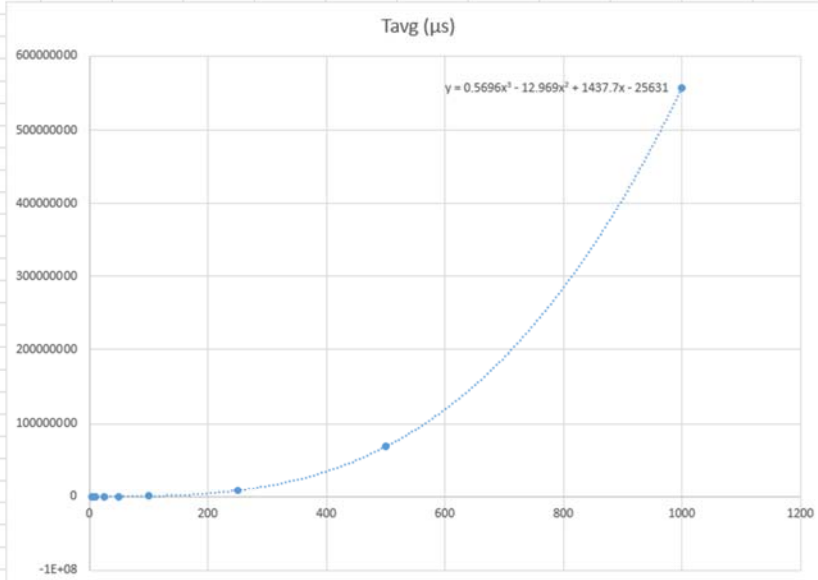




n	Tavg (µs)
5	65.68751219
10	497.472373
25	7994.059701
50	63746.65148
100	528879.5305
250	8453374.972
500	68638747.58
1000	558018188.9

log n	log Tavg
0.698970004	1.817482814
1	2.696768967
1.397940009	3.902767387
1.698970004	4.804457377
2	5.723356759
2.397940009	6.927030134
2.698970004	7.836569351
3	8.746648355





```
# CS 317 Algorithm Anaylsis Lab 2
# Drake Song
# Python 3.6

import numpy as np
import timeit
import os
import pdb

def clear():
    os.system('cls')

def addInt1():
    return 23 + 38

def addInt2():
    return 7261852 + 4917528

def multInt1():
    return 23 * 28

def multInt2():
    return 7261852 * 4917528

def divInt1():
    return 23 / 38

def divInt2():
    return 7261852 / 4917528

def addDoub1():
    return 23.3 + 38.1

def addDoub2():
    return 7261852.6 + 4917528.9

def multDoub1():
    return 23.3 * 38.1

def multDoub2():
    return 7261852.6 * 4917528.9

def divDoub1():
    return 23.3 / 38.1

def divDoub2():
    return 7261852.6 / 4917528.9
```

```
def sine():
    return np.sin(1.23)

def pwr():
    return np.power(3.13, 2.78)

def printStuff():
    for i in range(2):
        print(i)

def runFunction(n, x):
    time = []
    minimum = 0
    q1 = 0
    q3 = 0
    maximum = 0

    for i in range(n):
        for j in range(21):
            tic = timeit.default_timer()
            x()
            toc = timeit.default_timer()
            time.append((toc-tic)*1000000000)
        sorted_time = sorted(time)
        minimum += sorted_time[0]
        q1 += sorted_time[5]
        q3 += sorted_time[15]
        maximum += sorted_time[20]

    minimum /= n
    q1 /= n
    q3 /= n
    maximum /= n

    selected_time = [minimum, q1, q3, maximum]
    print(selected_time)

def runPrintStuff(n):
    time = []
    minimum = 0
    q1 = 0
    q3 = 0
    maximum = 0

    for i in range(n):
        for j in range(21):
            tic = timeit.default_timer()
            printStuff()
```

```
            toc = timeit.default_timer()
            time.append((toc-tic)*1000000000)

        sorted_time = sorted(time)
        minimum += sorted_time[0]
        q1 += sorted_time[5]
        q3 += sorted_time[15]
        maximum += sorted_time[20]

    clear()
    minimum /= n
    q1 /= n
    q3 /= n
    maximum /= n

    selected_time = [minimum, q1, q3, maximum]

    runFunction(n, addInt1)
    runFunction(n, addInt2)
    runFunction(n, multInt1)
    runFunction(n, multInt2)
    runFunction(n, divInt1)
    runFunction(n, divInt2)
    runFunction(n, addDoub1)
    runFunction(n, addDoub2)
    runFunction(n, multDoub1)
    runFunction(n, multDoub2)
    runFunction(n, divDoub1)
    runFunction(n, divDoub2)
    runFunction(n, sine)
    runFunction(n, pwr)

    print(selected_time)

n = int(input("Please enter a positive integer: "))
runPrintStuff(n)
```

```
# CS 317 Algorithm Anaylsis Lab 2
# Drake Song
# Python 3.6

import numpy as np
import timeit
import os

os.system('cls')
time = []
ans = []

n = int(input("Enter a number greater than 2: "))
matrix = np.random.randint(-100, 100, size=(n-1, n)).tolist()

# print("\nInitial matrix:")
# print(np.matrix(matrix))

for a in range(5):
    tic = timeit.default_timer()
    for k in range(n-2):
        largest = 0
        row = 0
        for i in range(k, n-1):
            if np.absolute(matrix[i][k]) > largest:
                largest = np.absolute(matrix[i][k])
                row = i

        temp = matrix[k]
        matrix[k] = matrix[row]
        matrix[row] = temp

        # print("\nMatrix after swap:")
        # print(np.matrix(matrix))

        for i in range(k+1, n-1):
            ratio = matrix[i][k]/matrix[k][k]
            temp = [np.absolute(matrix[k][j]*ratio) for j in range(n)]
            if matrix[i][k] > 0:
                matrix[i] = [matrix[i][j] - temp[j] for j in range(n)]
            else:
                matrix[i] = [matrix[i][j] + temp[j] for j in range(n)]

        # print("\nMatrix after subtraction:")
        # print(np.matrix(matrix))

    toc = timeit.default_timer()
    time.append((toc-tic)*1000000)
```

```
print(sum(time)/5)
```