Computer Organization and Architecture

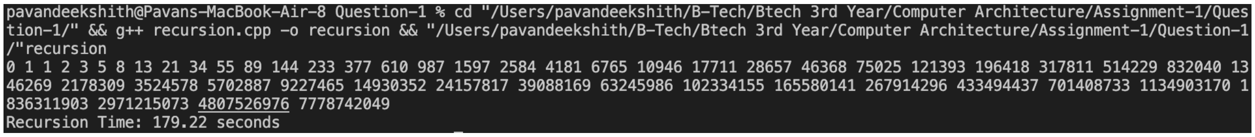
Pavan Deekshith - 22110190

Question -1:

a) Recursion

Code:

```
#include <iostream>
#include <ctime>

using namespace std;

long long fibonacci_recursion(int n) {
    if (n <= 1) return n;
    return fibonacci_recursion(n - 1) + fibonacci_recursion(n - 2);
}

int main() {
    struct timespec start, end;
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start);

    for (int i = 0; i < 50; ++i) {
        cout << fibonacci_recursion(i) << " ";
    }

    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);

    double time_taken = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec)
/ 1e9;
    cout << "\nRecursion Time: " << time_taken << " seconds\n";

    return 0;
}
```



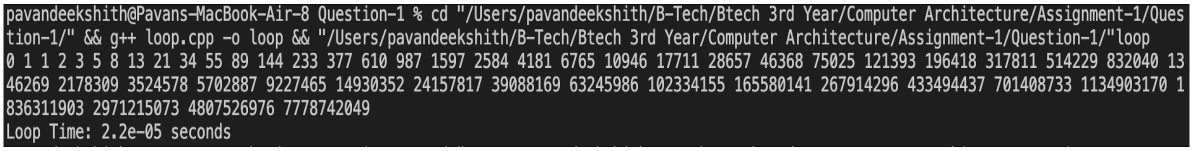
```
pavandeekshith@Pavans-MacBook-Air-8 Question-1 % cd "/Users/pavandeekshith/B-Tech/Btech 3rd Year/Computer Architecture/Assignment-1/Ques
tion-1/" && g++ recursion.cpp -o recursion && "/Users/pavandeekshith/B-Tech/Btech 3rd Year/Computer Architecture/Assignment-1/Question-1
/"recursion
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 317811 514229 832040 13
46269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986 102334155 165580141 267914296 433494437 701408733 1134903170 1
836311903 2971215073 4807526976 7778742049
Recursion Time: 179.22 seconds
```

Speedup = 1; Taking recursion time as base line

b) Loop

Code:

```
#include <iostream>
#include <ctime>
using namespace std;
void fibonacci_loop(int n) {
    long long a = 0, b = 1, next;
    for (int i = 0; i < n; ++i) {
        cout << a << " ";
        next = a + b;
        a = b;
        b = next;
    }
}

int main() {
    struct timespec start, end;
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start);

    fibonacci_loop(50);

    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);

double time_taken = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec)/ 1e9;
    cout << "\nLoop Time: " << time_taken << " seconds\n";

    return 0;
}
```



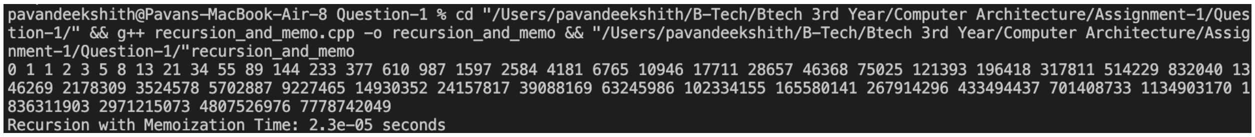
```
pavandeekshith@Pavans-MacBook-Air-8 Question-1 % cd "/Users/pavandeekshith/B-Tech/Btech 3rd Year/Computer Architecture/Assignment-1/Ques
tion-1/" && g++ loop.cpp -o loop && "/Users/pavandeekshith/B-Tech/Btech 3rd Year/Computer Architecture/Assignment-1/Question-1/"loop
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 317811 514229 832040 13
46269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986 102334155 165580141 267914296 433494437 701408733 1134903170 1
836311903 2971215073 4807526976 7778742049
Loop Time: 2.2e-05 seconds
```

Speed up = 81.36 * 10^5 times faster than baseline

c) Recursion and memoization

Code:

```
#include <iostream>
#include <vector>
#include <ctime>

using namespace std;

vector<long long> memo(51, -1);

long long fibonacci_recursion_memo(int n) {
    if (n <= 1) return n;
    if (memo[n] != -1) return memo[n];
    return memo[n] = fibonacci_recursion_memo(n - 1) + fibonacci_recursion_memo(n - 2);
}

int main() {
    struct timespec start, end;
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start);

    for (int i = 0; i < 50; ++i) {
        cout << fibonacci_recursion_memo(i) << " ";
    }

    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);

    double time_taken = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;
    cout << "\nRecursion with Memoization Time: " << time_taken << " seconds\n";

    return 0;
}
```



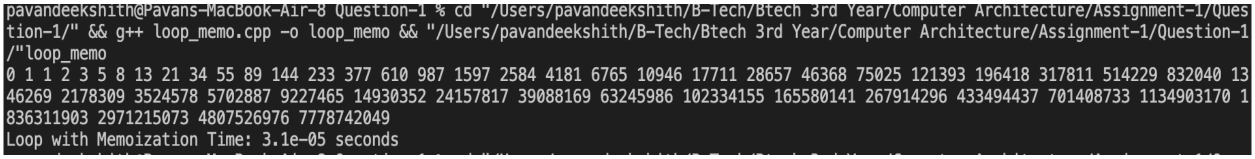
```
pavandeekshith@Pavans-MacBook-Air-8 Question-1 % cd "/Users/pavandeekshith/B-Tech/Btech 3rd Year/Computer Architecture/Assignment-1/Ques
tion-1/" && g++ recursion_and_memo.cpp -o recursion_and_memo && "/Users/pavandeekshith/B-Tech/Btech 3rd Year/Computer Architecture/Assig
nment-1/Question-1/"recursion_and_memo
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 317811 514229 832040 13
46269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986 102334155 165580141 267914296 433494437 701408733 1134903170 1
836311903 2971215073 4807526976 7778742049
Recursion with Memoization Time: 2.3e-05 seconds
```

Speed up = 77.8 * 10^5 times faster than baseline

d) Loop and memoization

Code:

```
#include <iostream>
#include <vector>
#include <ctime>

using namespace std;

void fibonacci_loop_memo(int n) {
    vector<long long> fib(n, 0);
    fib[1] = 1;
    for (int i = 2; i < n; ++i) {
        fib[i] = fib[i - 1] + fib[i - 2];
    }
    for (int i = 0; i < n; ++i) {
        cout << fib[i] << " ";
    }
}

int main() {
    struct timespec start, end;
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start);

    fibonacci_loop_memo(50);

    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);

    double time_taken = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;
    cout << "\nLoop with Memoization Time: " << time_taken << " seconds\n";

    return 0;
}
```



```
pavandeekshith@Pavans-MacBook-Air-8 Question-1 % cd "/Users/pavandeekshith/B-Tech/Btech 3rd Year/Computer Architecture/Assignment-1/Ques
tion-1/" && g++ loop_memo.cpp -o loop_memo && "/Users/pavandeekshith/B-Tech/Btech 3rd Year/Computer Architecture/Assignment-1/Question-1
/"loop_memo
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 317811 514229 832040 13
46269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986 102334155 165580141 267914296 433494437 701408733 1134903170 1
836311903 2971215073 4807526976 7778742049
Loop with Memoization Time: 3.1e-05 seconds
```

Speed up = 57.74 * 10^5 times faster than baseline

Question-2:

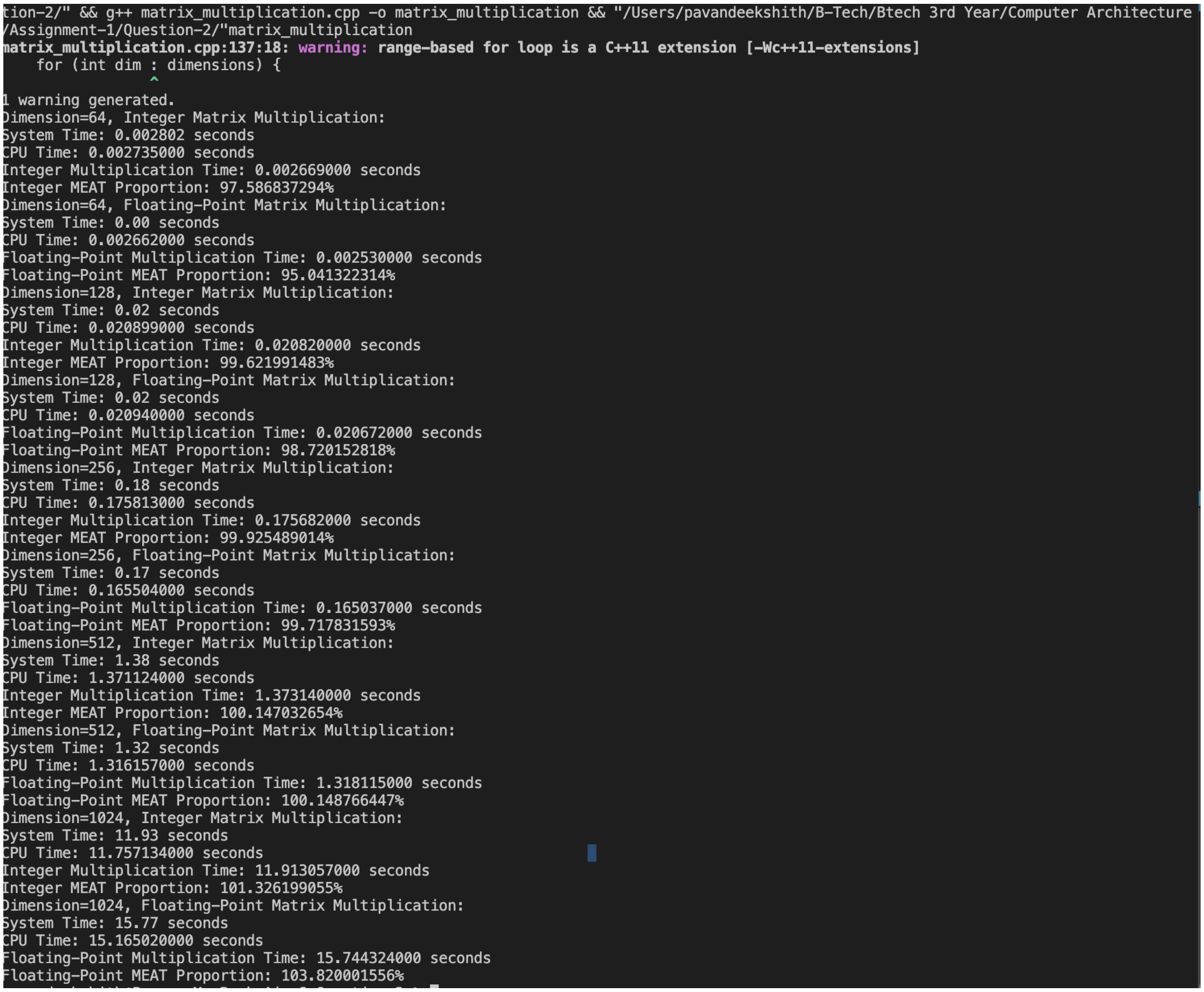
a) C++

Code:

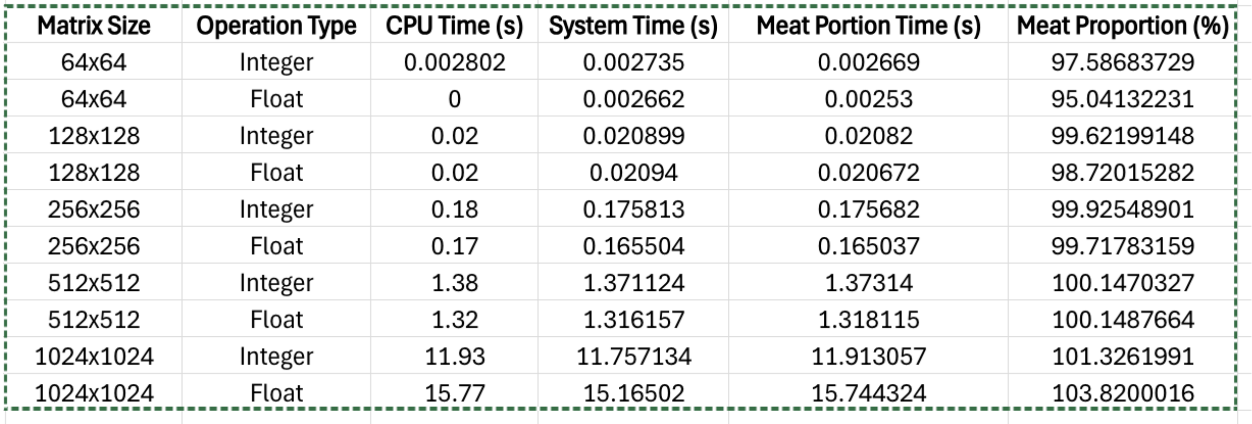
```
#include <iostream>
#include <vector>
#include <ctime>
#include <iomanip>

using namespace std;

void multiplyIntegerMatrices(const vector<vector<int> >& matrix1, const
vector<vector<int> >& matrix2, vector<vector<int> >& result, int dimension) {
    for (int i = 0; i < dimension; i++) {
        for (int j = 0; j < dimension; j++) {
            result[i][j] = 0;
            for (int k = 0; k < dimension; k++) {
                result[i][j] += matrix1[i][k] * matrix2[k][j];
            }
        }
    }
}

void multiplyFloatingPointMatrices(const vector<vector<double> >& matrix1, const
vector<vector<double> >& matrix2, vector<vector<double> >& result, int dimension) {
    for (int i = 0; i < dimension; i++) {
        for (int j = 0; j < dimension; j++) {
            result[i][j] = 0.0;
            for (int k = 0; k < dimension; k++) {
                result[i][j] += matrix1[i][k] * matrix2[k][j];
            }
        }
    }
}

void displayTime(const struct timespec& startTime, const struct timespec& endTime,
const string& message) {
    double elapsedTime = (endTime.tv_sec - startTime.tv_sec) * 1e9;
    elapsedTime = (elapsedTime + (endTime.tv_nsec - startTime.tv_nsec)) * 1e-9;
    cout << message << fixed << elapsedTime << setprecision(9) << " seconds" << endl;
}

int main() {
    int dimensions[] = {64, 128, 256, 512, 1024};
```

```
  for (int dim : dimensions) {
    struct timespec systemStartTime, systemEndTime, cpuStartTime, cpuEndTime;

    clock_gettime(CLOCK_MONOTONIC, &systemStartTime);
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &cpuStartTime);

    vector<vector<int> > matrixInt1(dim, vector<int>(dim, 1));
    vector<vector<int> > matrixInt2(dim, vector<int>(dim, 1));
    vector<vector<int> > matrixIntResult(dim, vector<int>(dim, 0));

    struct timespec multiplyStartTime, multiplyEndTime;
    clock_gettime(CLOCK_MONOTONIC, &multiplyStartTime);

    multiplyIntegerMatrices(matrixInt1, matrixInt2, matrixIntResult, dim);

    clock_gettime(CLOCK_MONOTONIC, &multiplyEndTime);

    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &cpuEndTime);
    clock_gettime(CLOCK_MONOTONIC, &systemEndTime);

    double totalSystemTime = (systemEndTime.tv_sec - systemStartTime.tv_sec) +
(systemEndTime.tv_nsec - systemStartTime.tv_nsec) * 1e-9;
    double totalCPUTime = (cpuEndTime.tv_sec - cpuStartTime.tv_sec) +
(cpuEndTime.tv_nsec - cpuStartTime.tv_nsec) * 1e-9;
    double multiplyTime = (multiplyEndTime.tv_sec - multiplyStartTime.tv_sec) +
(multiplyEndTime.tv_nsec - multiplyStartTime.tv_nsec) * 1e-9;

    double meatProportion = (multiplyTime / totalCPUTime) * 100;

    cout << "Dimension=" << dim << ", Integer Matrix Multiplication: " << endl;
    displayTime(systemStartTime, systemEndTime, "System Time: ");
    displayTime(cpuStartTime, cpuEndTime, "CPU Time: ");
    displayTime(multiplyStartTime, multiplyEndTime, "Integer Multiplication Time: ");
    cout << "Integer MEAT Proportion: " << fixed << meatProportion << setprecision(2)
<< "%" << endl;

    clock_gettime(CLOCK_MONOTONIC, &systemStartTime);
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &cpuStartTime);

    vector<vector<double> > matrixFloat1(dim, vector<double>(dim, 1.0));
    vector<vector<double> > matrixFloat2(dim, vector<double>(dim, 1.0));
    vector<vector<double> > matrixFloatResult(dim, vector<double>(dim, 0.0));
```

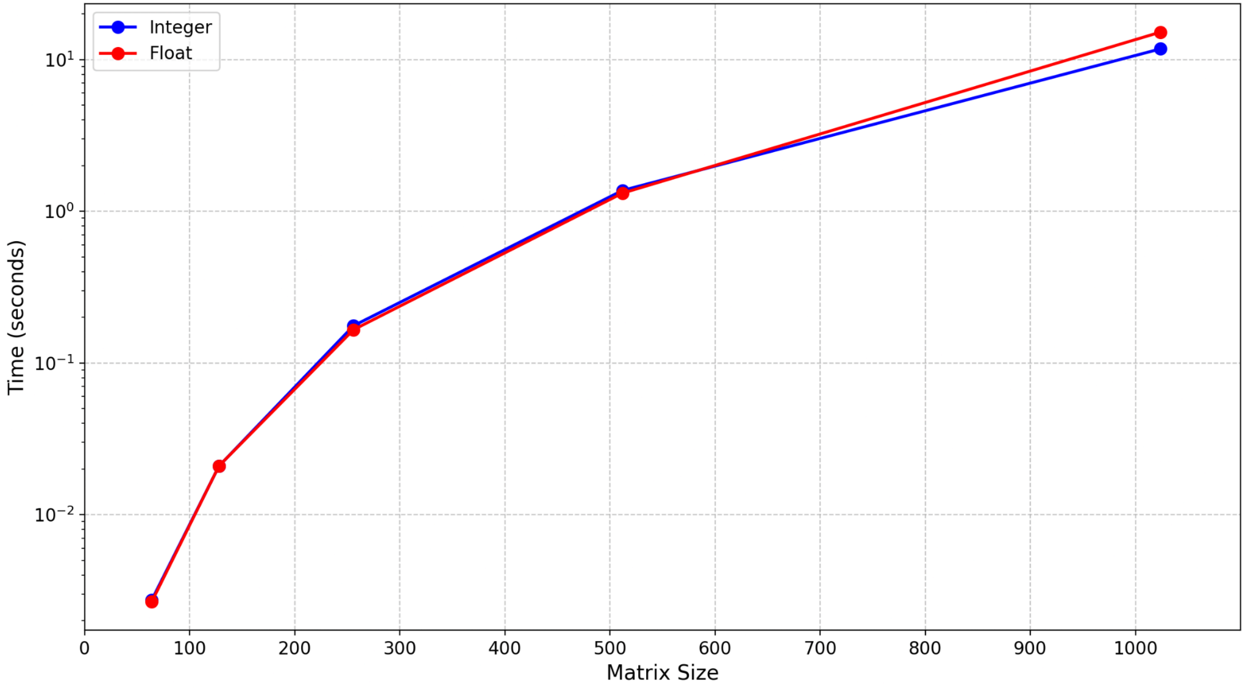
```
        clock_gettime(CLOCK_MONOTONIC, &multiplyStartTime);

        multiplyFloatingPointMatrices(matrixFloat1, matrixFloat2, matrixFloatResult, dim);

        clock_gettime(CLOCK_MONOTONIC, &multiplyEndTime);

        clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &cpuEndTime);
        clock_gettime(CLOCK_MONOTONIC, &systemEndTime);

        totalSystemTime = (systemEndTime.tv_sec - systemStartTime.tv_sec) +
(systemEndTime.tv_nsec - systemStartTime.tv_nsec) * 1e-9;
        totalCPUTime = (cpuEndTime.tv_sec - cpuStartTime.tv_sec) +
(cpuEndTime.tv_nsec - cpuStartTime.tv_nsec) * 1e-9;
        multiplyTime = (multiplyEndTime.tv_sec - multiplyStartTime.tv_sec) +
(multiplyEndTime.tv_nsec - multiplyStartTime.tv_nsec) * 1e-9;

        meatProportion = (multiplyTime / totalCPUTime) * 100;

        cout << "Dimension=" << dim << ", Floating-Point Matrix Multiplication: " << endl;
        displayTime(systemStartTime, systemEndTime, "System Time: ");
        displayTime(cpuStartTime, cpuEndTime, "CPU Time: ");
        displayTime(multiplyStartTime, multiplyEndTime, "Floating-Point Multiplication
Time: ");
        cout << "Floating-Point MEAT Proportion: " << fixed << meatProportion <<
setprecision(2) << "%" << endl;
    }

    return 0;
}
```

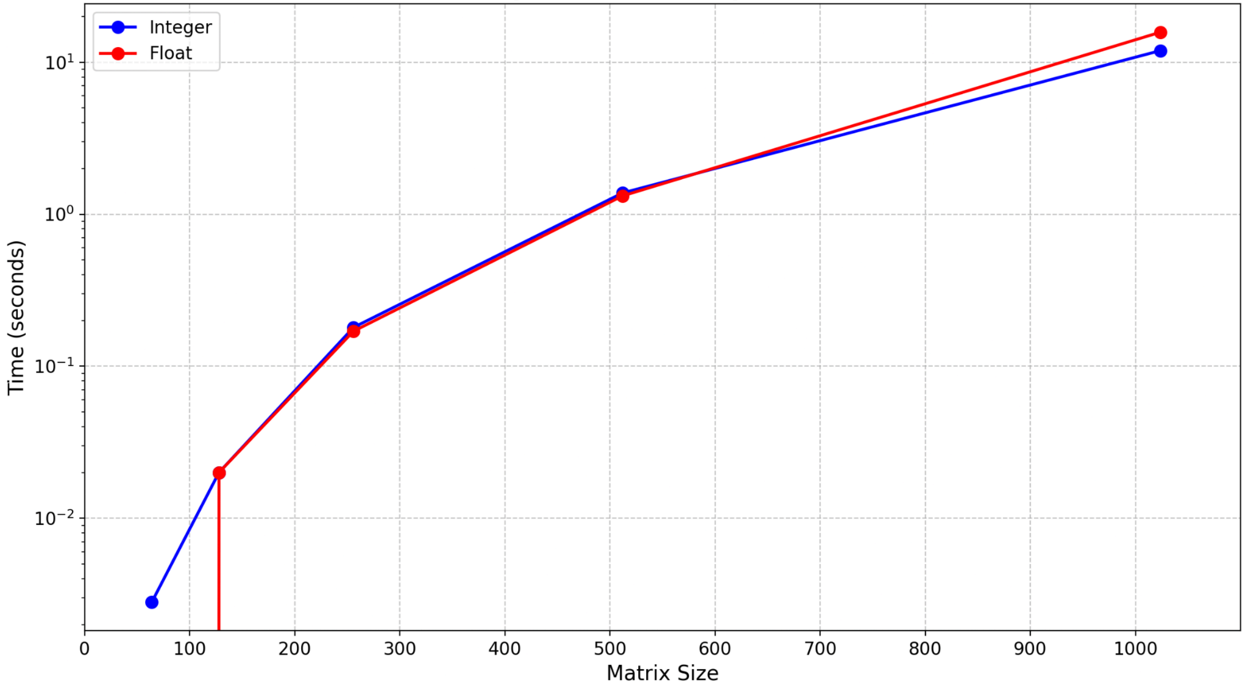


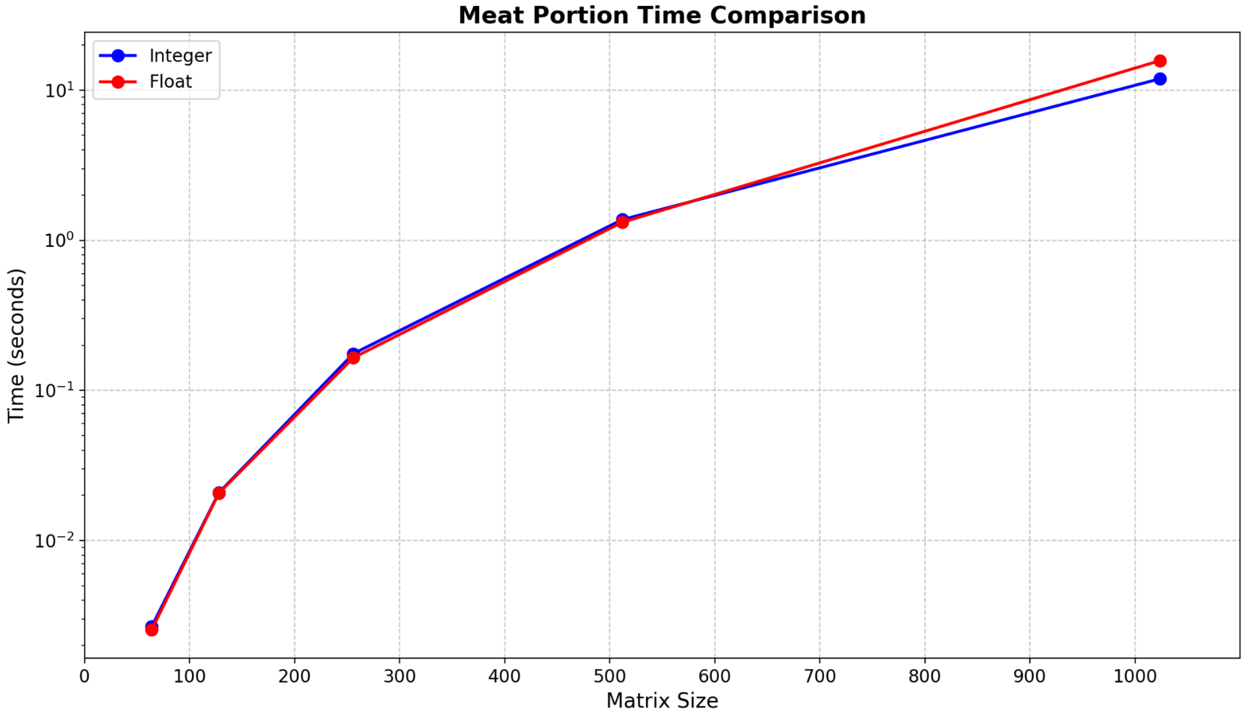
```
tion-2/" && g++ matrix_multiplication.cpp -o matrix_multiplication && "/Users/pavandeekshith/B-Tech/Btech 3rd Year/Computer Architecture
/Assignment-1/Question-2/"matrix_multiplication
matrix_multiplication.cpp:137:18: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
    for (int dim : dimensions) {
                 ^

1 warning generated.
Dimension=64, Integer Matrix Multiplication:
System Time: 0.002802 seconds
CPU Time: 0.002735000 seconds
Integer Multiplication Time: 0.002669000 seconds
Integer MEAT Proportion: 97.586837294%
Dimension=64, Floating-Point Matrix Multiplication:
System Time: 0.00 seconds
CPU Time: 0.002662000 seconds
Floating-Point Multiplication Time: 0.002530000 seconds
Floating-Point MEAT Proportion: 95.041322314%
Dimension=128, Integer Matrix Multiplication:
System Time: 0.02 seconds
CPU Time: 0.020899000 seconds
Integer Multiplication Time: 0.020820000 seconds
Integer MEAT Proportion: 99.621991483%
Dimension=128, Floating-Point Matrix Multiplication:
System Time: 0.02 seconds
CPU Time: 0.020940000 seconds
Floating-Point Multiplication Time: 0.020672000 seconds
Floating-Point MEAT Proportion: 98.720152818%
Dimension=256, Integer Matrix Multiplication:
System Time: 0.18 seconds
CPU Time: 0.175813000 seconds
Integer Multiplication Time: 0.175682000 seconds
Integer MEAT Proportion: 99.925489014%
Dimension=256, Floating-Point Matrix Multiplication:
System Time: 0.17 seconds
CPU Time: 0.165504000 seconds
Floating-Point Multiplication Time: 0.165037000 seconds
Floating-Point MEAT Proportion: 99.717831593%
Dimension=512, Integer Matrix Multiplication:
System Time: 1.38 seconds
CPU Time: 1.371124000 seconds
Integer Multiplication Time: 1.373140000 seconds
Integer MEAT Proportion: 100.147032654%
Dimension=512, Floating-Point Matrix Multiplication:
System Time: 1.32 seconds
CPU Time: 1.316157000 seconds
Floating-Point Multiplication Time: 1.318115000 seconds
Floating-Point MEAT Proportion: 100.148766447%
Dimension=1024, Integer Matrix Multiplication:
System Time: 11.93 seconds
CPU Time: 11.757134000 seconds
Integer Multiplication Time: 11.913057000 seconds
Integer MEAT Proportion: 101.326199055%
Dimension=1024, Floating-Point Matrix Multiplication:
System Time: 15.77 seconds
CPU Time: 15.165020000 seconds
Floating-Point Multiplication Time: 15.744324000 seconds
Floating-Point MEAT Proportion: 103.820001556%
```

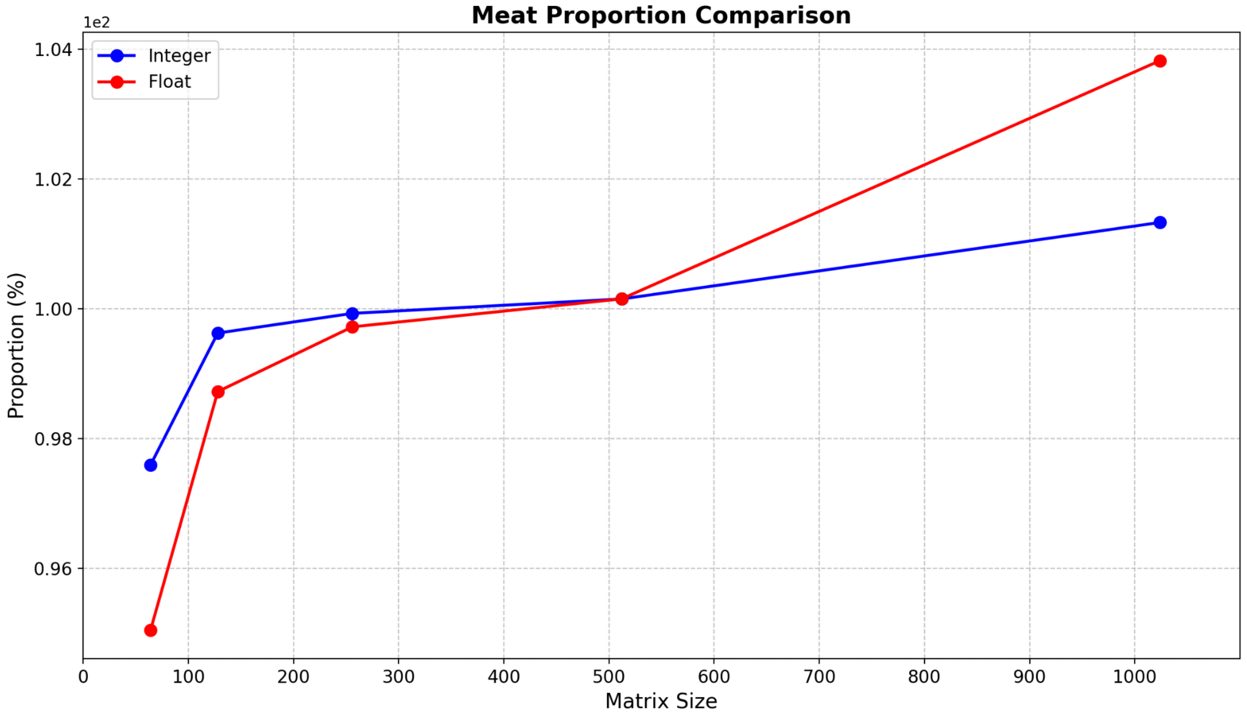


Matrix Size	Operation Type	CPU Time (s)	System Time (s)	Meat Portion Time (s)	Meat Proportion (%)
64x64	Integer	0.002802	0.002735	0.002669	97.58683729
64x64	Float	0	0.002662	0.00253	95.04132231
128x128	Integer	0.02	0.020899	0.02082	99.62199148
128x128	Float	0.02	0.02094	0.020672	98.72015282
256x256	Integer	0.18	0.175813	0.175682	99.92548901
256x256	Float	0.17	0.165504	0.165037	99.71783159
512x512	Integer	1.38	1.371124	1.37314	100.1470327
512x512	Float	1.32	1.316157	1.318115	100.1487664
1024x1024	Integer	11.93	11.757134	11.913057	101.3261991
1024x1024	Float	15.77	15.16502	15.744324	103.8200016


CPU Time Comparison


System Time Comparison

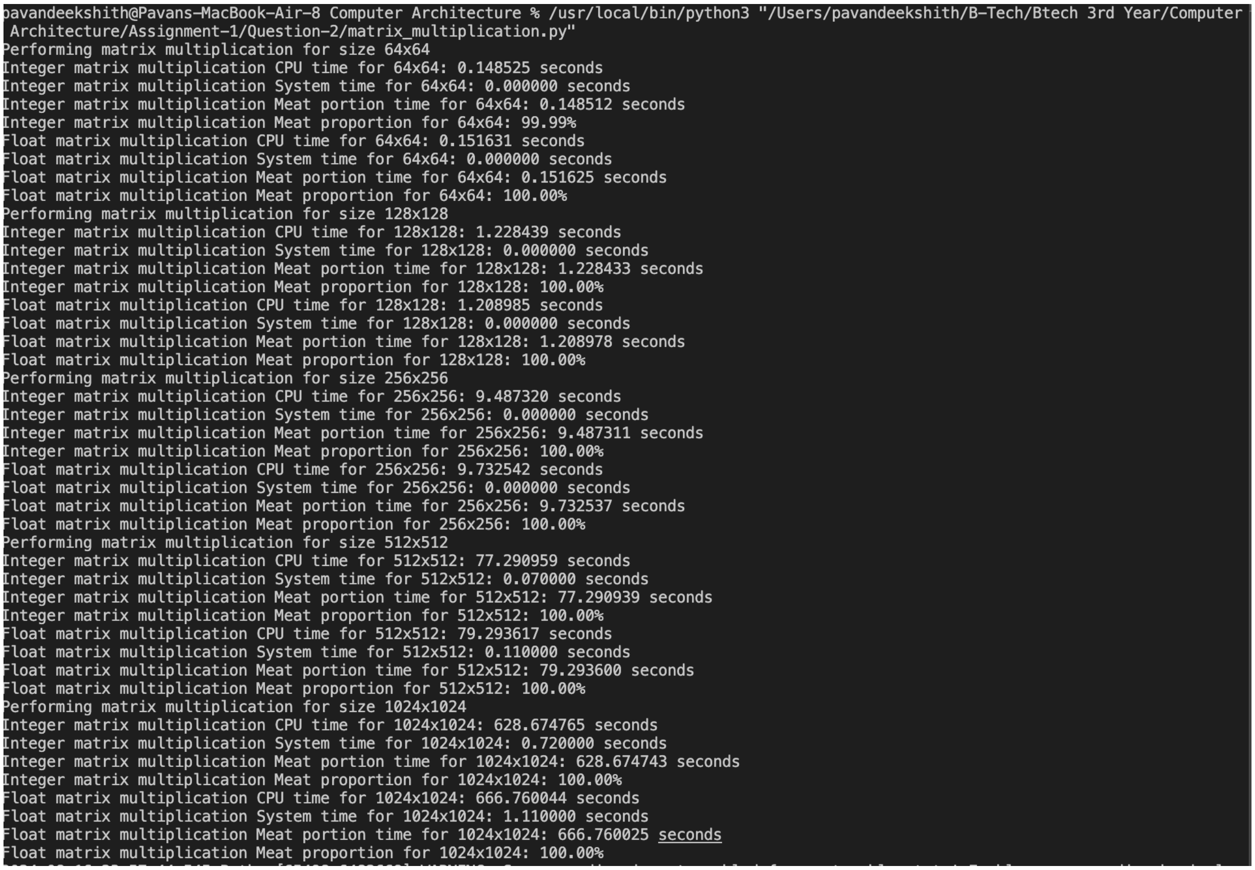

Meat Portion Time Comparison
Integer
Float
Time (seconds)
10^{1}
10^{0}
10^{-1}
10^{-2}
0
100
200
300
400
500
600
700
800
900
1000
Matrix Size


Meat Proportion Comparison
1e2
Integer
Float
Proportion (%)
1.04
1.02
1.00
0.98
0.96
0
100
200
300
400
500
600
700
800
900
1000
Matrix Size

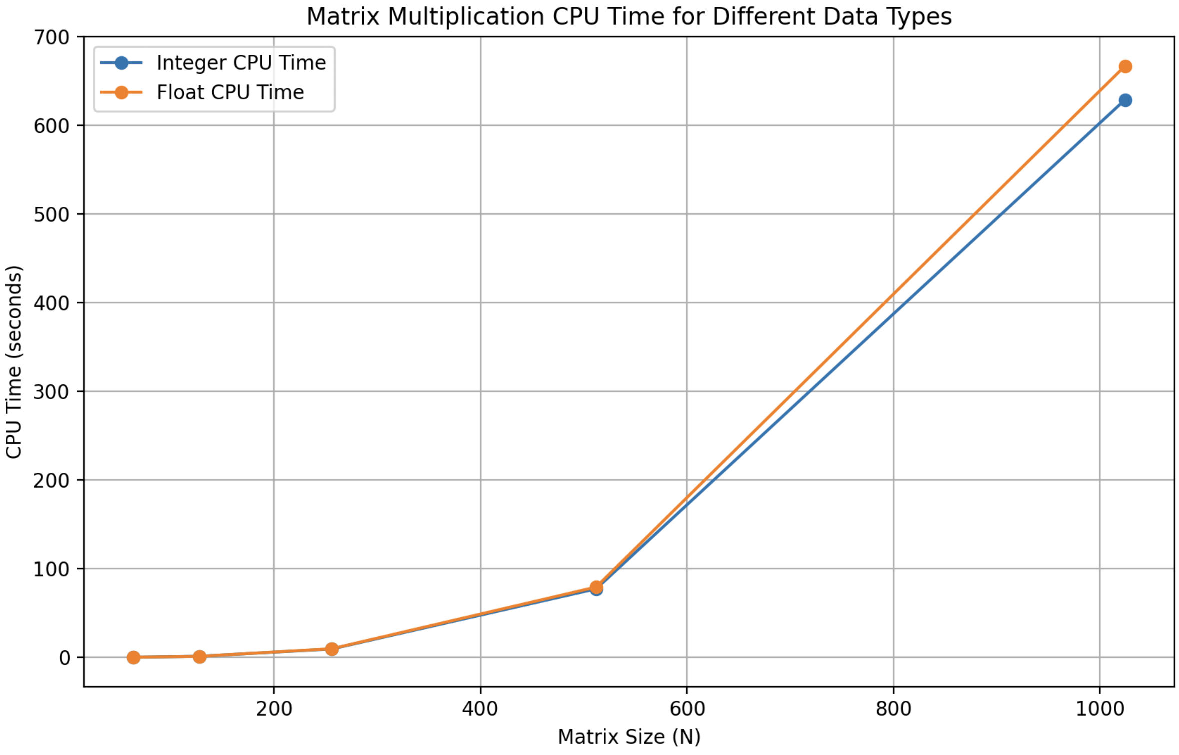
b) Python

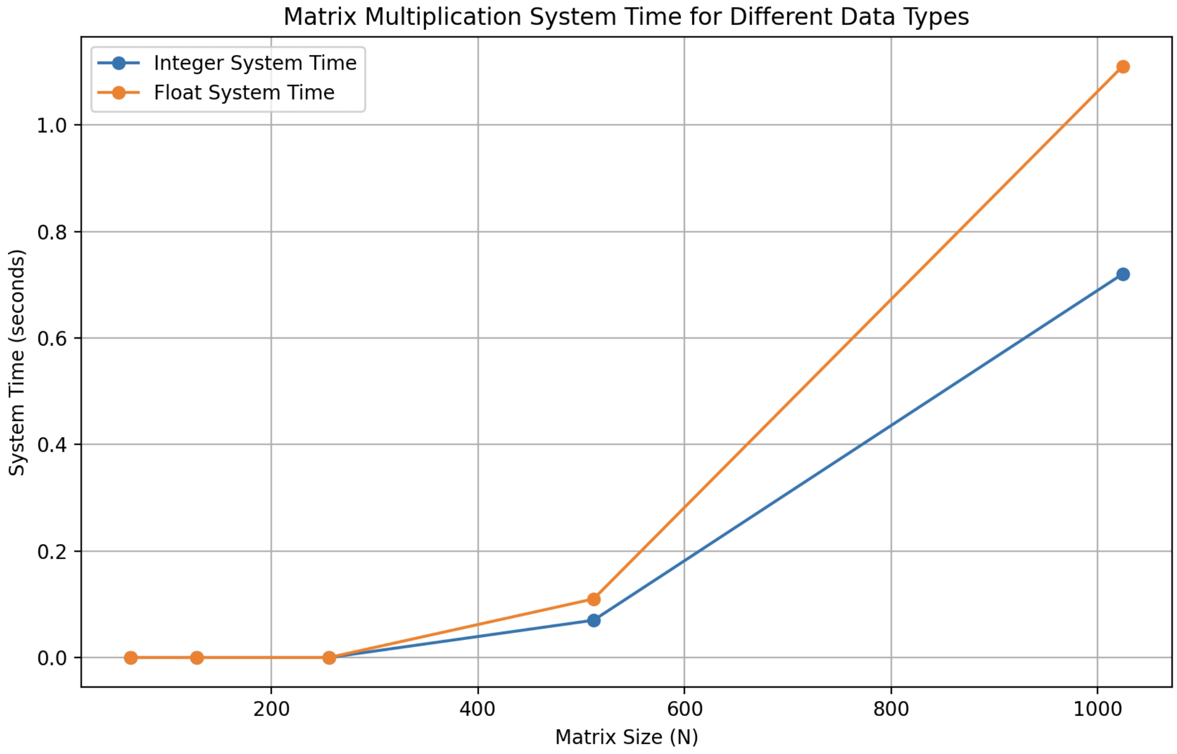
Code:

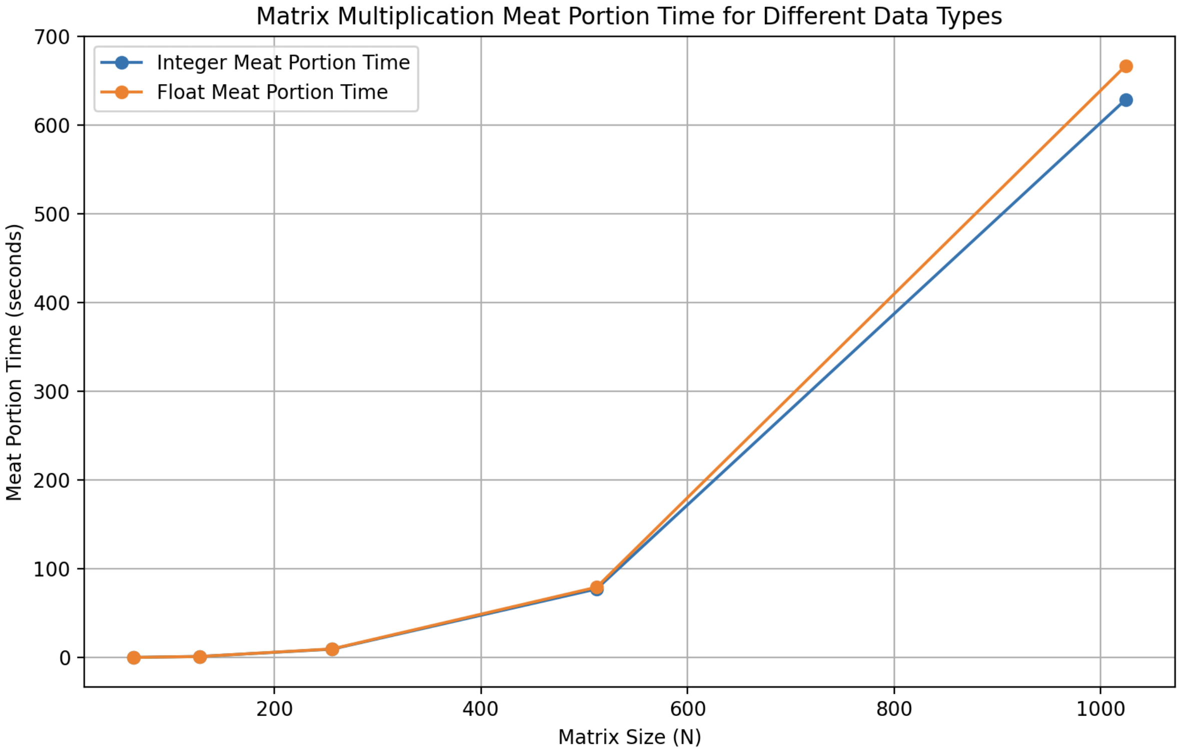
```
import numpy as np
import time
import os
import matplotlib.pyplot as plt

N_values = [64, 128, 256, 512, 1024]

int_cpu_times = []
int_system_times = []
float_cpu_times = []
float_system_times = []

int_meat_times = []
float_meat_times = []

int_meat_percentages = []
float_meat_percentages = []

def matrix_multiplication(N, dtype):
    np.random.seed(0)
    matrix = np.random.randint(0, 11, size=(N, N)).astype(dtype)
    result = np.zeros((N, N), dtype=dtype)

    start_process_time_total = time.process_time()
    start_os_times_total = os.times()

    start_meat_time = time.process_time()

    for i in range(N):
        for j in range(N):
            for k in range(N):
                result[i][j] += matrix[i][k] * matrix[k][j]

    end_meat_time = time.process_time()

    end_process_time_total = time.process_time()
    end_os_times_total = os.times()

    cpu_time_used_total = end_process_time_total - start_process_time_total
    user_time_used_total = end_os_times_total.user - start_os_times_total.user
    system_time_used_total = end_os_times_total.system - start_os_times_total.system
    meat_time_used = end_meat_time - start_meat_time

    meat_percentage = (meat_time_used / cpu_time_used_total * 100) if
cpu_time_used_total > 0 else 0

    return cpu_time_used_total, system_time_used_total, meat_time_used,
meat_percentage

for N in N_values:
    print(f"Performing matrix multiplication for size {N}x{N}")

    int_cpu_time, int_system_time, int_meat_time, int_meat_percentage =
matrix_multiplication(N, dtype=int)
    int_cpu_times.append(int_cpu_time)
    int_system_times.append(int_system_time)
    int_meat_times.append(int_meat_time)
    int_meat_percentages.append(int_meat_percentage)
    print(f"Integer matrix multiplication CPU time for {N}x{N}: {int_cpu_time:.6f} seconds")
    print(f"Integer matrix multiplication System time for {N}x{N}: {int_system_time:.6f}
seconds")
    print(f"Integer matrix multiplication Meat portion time for {N}x{N}: {int_meat_time:.6f}
seconds")
    print(f"Integer matrix multiplication Meat proportion for {N}x{N}:
{int_meat_percentage:.2f}%")

    float_cpu_time, float_system_time, float_meat_time, float_meat_percentage =
matrix_multiplication(N, dtype=float)
    float_cpu_times.append(float_cpu_time)
    float_system_times.append(float_system_time)
    float_meat_times.append(float_meat_time)
    float_meat_percentages.append(float_meat_percentage)
    print(f"Float matrix multiplication CPU time for {N}x{N}: {float_cpu_time:.6f} seconds")
    print(f"Float matrix multiplication System time for {N}x{N}: {float_system_time:.6f}
seconds")
    print(f"Float matrix multiplication Meat portion time for {N}x{N}: {float_meat_time:.6f}
seconds")
    print(f"Float matrix multiplication Meat proportion for {N}x{N}:
{float_meat_percentage:.2f}%")


plt.figure(figsize=(10, 6))
plt.plot(N_values, int_cpu_times, label="Integer CPU Time", marker='o')
plt.plot(N_values, float_cpu_times, label="Float CPU Time", marker='o')
plt.xlabel('Matrix Size (N)')
plt.ylabel('CPU Time (seconds)')
plt.title('Matrix Multiplication CPU Time for Different Data Types')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(N_values, int_system_times, label="Integer System Time", marker='o')
plt.plot(N_values, float_system_times, label="Float System Time", marker='o')
plt.xlabel('Matrix Size (N)')
plt.ylabel('System Time (seconds)')
plt.title('Matrix Multiplication System Time for Different Data Types')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(N_values, int_meat_times, label="Integer Meat Portion Time", marker='o')
plt.plot(N_values, float_meat_times, label="Float Meat Portion Time", marker='o')
plt.xlabel('Matrix Size (N)')
plt.ylabel('Meat Portion Time (seconds)')
plt.title('Matrix Multiplication Meat Portion Time for Different Data Types')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(N_values, int_meat_percentages, label="Integer Meat Portion Percentage",
marker='o')
plt.plot(N_values, float_meat_percentages, label="Float Meat Portion Percentage",
marker='o')
plt.xlabel('Matrix Size (N)')
plt.ylabel('Meat Portion Percentage (%)')
plt.title('Matrix Multiplication Meat Portion Percentage for Different Data Types')
plt.legend()
plt.grid(True)
plt.show()
```

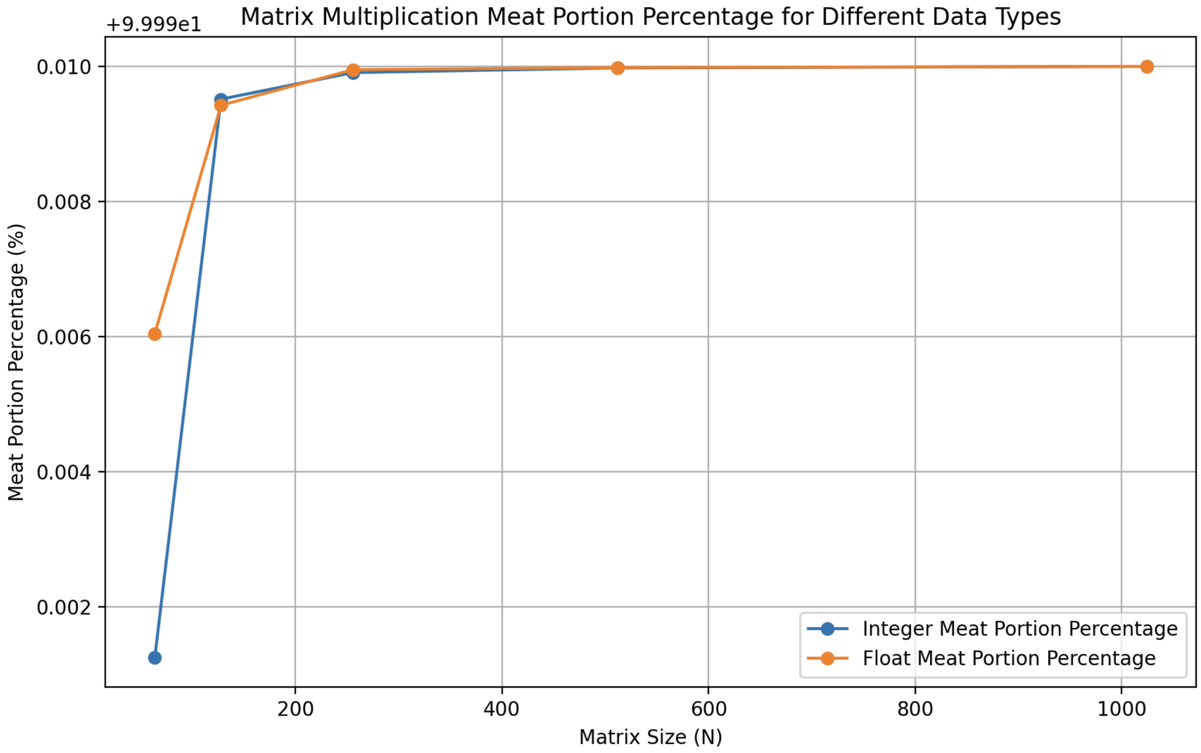


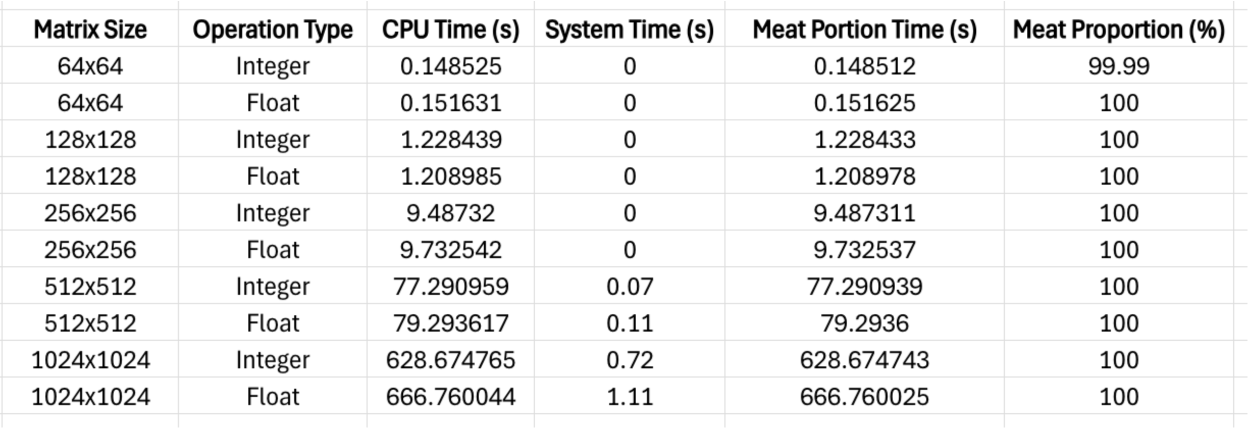
```
pavandeekshith@Pavans-MacBook-Air-8 Computer Architecture % /usr/local/bin/python3 "/Users/pavandeekshith/B-Tech/Btech 3rd Year/Computer
 Architecture/Assignment-1/Question-2/matrix_multiplication.py"
Performing matrix multiplication for size 64x64
Integer matrix multiplication CPU time for 64x64: 0.148525 seconds
Integer matrix multiplication System time for 64x64: 0.000000 seconds
Integer matrix multiplication Meat portion time for 64x64: 0.148512 seconds
Integer matrix multiplication Meat proportion for 64x64: 99.99%
Float matrix multiplication CPU time for 64x64: 0.151631 seconds
Float matrix multiplication System time for 64x64: 0.000000 seconds
Float matrix multiplication Meat portion time for 64x64: 0.151625 seconds
Float matrix multiplication Meat proportion for 64x64: 100.00%
Performing matrix multiplication for size 128x128
Integer matrix multiplication CPU time for 128x128: 1.228439 seconds
Integer matrix multiplication System time for 128x128: 0.000000 seconds
Integer matrix multiplication Meat portion time for 128x128: 1.228433 seconds
Integer matrix multiplication Meat proportion for 128x128: 100.00%
Float matrix multiplication CPU time for 128x128: 1.208985 seconds
Float matrix multiplication System time for 128x128: 0.000000 seconds
Float matrix multiplication Meat portion time for 128x128: 1.208978 seconds
Float matrix multiplication Meat proportion for 128x128: 100.00%
Performing matrix multiplication for size 256x256
Integer matrix multiplication CPU time for 256x256: 9.487320 seconds
Integer matrix multiplication System time for 256x256: 0.000000 seconds
Integer matrix multiplication Meat portion time for 256x256: 9.487311 seconds
Integer matrix multiplication Meat proportion for 256x256: 100.00%
Float matrix multiplication CPU time for 256x256: 9.732542 seconds
Float matrix multiplication System time for 256x256: 0.000000 seconds
Float matrix multiplication Meat portion time for 256x256: 9.732537 seconds
Float matrix multiplication Meat proportion for 256x256: 100.00%
Performing matrix multiplication for size 512x512
Integer matrix multiplication CPU time for 512x512: 77.290959 seconds
Integer matrix multiplication System time for 512x512: 0.070000 seconds
Integer matrix multiplication Meat portion time for 512x512: 77.290939 seconds
Integer matrix multiplication Meat proportion for 512x512: 100.00%
Float matrix multiplication CPU time for 512x512: 79.293617 seconds
Float matrix multiplication System time for 512x512: 0.110000 seconds
Float matrix multiplication Meat portion time for 512x512: 79.293600 seconds
Float matrix multiplication Meat proportion for 512x512: 100.00%
Performing matrix multiplication for size 1024x1024
Integer matrix multiplication CPU time for 1024x1024: 628.674765 seconds
Integer matrix multiplication System time for 1024x1024: 0.720000 seconds
Integer matrix multiplication Meat portion time for 1024x1024: 628.674743 seconds
Integer matrix multiplication Meat proportion for 1024x1024: 100.00%
Float matrix multiplication CPU time for 1024x1024: 666.760044 seconds
Float matrix multiplication System time for 1024x1024: 1.110000 seconds
Float matrix multiplication Meat portion time for 1024x1024: 666.760025 seconds
Float matrix multiplication Meat proportion for 1024x1024: 100.00%
```




Matrix Multiplication System Time for Different Data Types
Integer System Time
Float System Time
System Time (seconds)
0.0
0.2
0.4
0.6
0.8
1.0
200
400
600
800
1000
Matrix Size (N)


Matrix Multiplication Meat Portion Time for Different Data Types
Integer Meat Portion Time
Float Meat Portion Time
Meat Portion Time (seconds)
0
100
200
300
400
500
600
700
200
400
600
800
1000
Matrix Size (N)





Matrix Size	Operation Type	CPU Time (s)	System Time (s)	Meat Portion Time (s)	Meat Proportion (%)
64x64	Integer	0.148525	0	0.148512	99.99
64x64	Float	0.151631	0	0.151625	100
128x128	Integer	1.228439	0	1.228433	100
128x128	Float	1.208985	0	1.208978	100
256x256	Integer	9.48732	0	9.487311	100
256x256	Float	9.732542	0	9.732537	100
512x512	Integer	77.290959	0.07	77.290939	100
512x512	Float	79.293617	0.11	79.2936	100
1024x1024	Integer	628.674765	0.72	628.674743	100
1024x1024	Float	666.760044	1.11	666.760025	100