

Assignment 1

Shubham Agrawal (22110249)

shubham.agrawal@iitgn.ac.in

ES-215

Computer Architecture And Organization

Q1.

Assumptions: Instead of running each program for 10 times, and taking average for best CPU time estimate, I have directly taken CPU time of 1 runtime of each program. Also, I have assumed that the fibonacci series starts with 0 as its first element.

a. Using recursion

```
main.cpp  [Icons] [Share] [Run]

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  long long fibonacciRecursion(int n) {
5      if (n <= 1)
6          return n;
7      return fibonacciRecursion(n - 1) + fibonacciRecursion(n - 2);
8  }
9
10 int main() {
11     struct timespec start, end;
12     clock_gettime(CLOCK_MONOTONIC, &start);
13
14     for (int i = 0; i < 50; i++) {
15         cout << fibonacciRecursion(i) << " ";
16     }
17     cout << endl;
18
19     clock_gettime(CLOCK_MONOTONIC, &end);
20     double time_taken = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start
        .tv_nsec) / 1e9;
21     cout << "Time taken by Recursion: " << time_taken << " seconds" << endl;
22
23     return 0;
24 }
25
```

Output

Clear

```
/tmp/OWPi8gpAw5.o
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 1346269 2178309 3524578 5702887
9227465 14930352 24157817 39088169 63245986 102334155 165580141 267914296
433494437 701408733 1134903170 1836311903 2971215073 4807526976 7778742049
Time taken by Recursion: 155.483 seconds

=== Code Execution Successful ===
```

CPU time : 155.483 seconds

b. Using loop

main.cpp

Share

Run

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 long long fibonacciLoop(int n) {
5     if (n <= 1)
6         return n;
7
8     long long a = 0, b = 1, c;
9     for (int i = 2; i <= n; i++) {
10         c = a + b;
11         a = b;
12         b = c;
13     }
14     return b;
15 }
```

```

16
17 ▾ int main() {
18     struct timespec start, end;
19     clock_gettime(CLOCK_MONOTONIC, &start);
20
21 ▾     for (int i = 0; i < 50; i++) {
22         cout << fibonacciLoop(i) << " ";
23     }
24     cout << endl;
25
26     clock_gettime(CLOCK_MONOTONIC, &end);
27     double time_taken = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start
        .tv_nsec) / 1e9;
28     cout << "Time taken by Loop: " << time_taken << " seconds" << endl;
29
30     return 0;
31 }
32

```

Output

Clear

/tmp/FDlP0gbDCU.o

```

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 1346269 2178309 3524578 5702887
  9227465 14930352 24157817 39088169 63245986 102334155 165580141 267914296
  433494437 701408733 1134903170 1836311903 2971215073 4807526976 7778742049





```

Time taken by Loop: 5.873e-05 seconds

=== Code Execution Successful ===

CPU time : 5.873×10^{-5} seconds

c. Using recursion with memoization

```
main.cpp    Share  Run
```

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  vector<long long> dp(50, -1);
5
6  long long fibonacciRecursionMemo(int n) {
7      if (n <= 1)
8          return n;
9      if (dp[n] != -1)
10         return dp[n];
11     return dp[n] = fibonacciRecursionMemo(n - 1) + fibonacciRecursionMemo(n - 2);
12 }
13
13
14 int main() {
15     struct timespec start, end;
16     clock_gettime(CLOCK_MONOTONIC, &start);
17
18     for (int i = 0; i < 50; i++) {
19         cout << fibonacciRecursionMemo(i) << " ";
20     }
21     cout << endl;
22
23     clock_gettime(CLOCK_MONOTONIC, &end);
24     double time_taken = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;
25     cout << "Time taken by Recursion with Memoization: " << time_taken << " seconds" << endl;
26
27     return 0;
28 }
29
```

Output

Clear

```
/tmp/owfQhkKqA9.o
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 1346269 2178309 3524578 5702887
9227465 14930352 24157817 39088169 63245986 102334155 165580141 267914296
433494437 701408733 1134903170 1836311903 2971215073 4807526976 7778742049
Time taken by Recursion with Memoization: 5.586e-05 seconds

=== Code Execution Successful ===
```

CPU time : 5.586×10^{-5} seconds

d. Using loop with memoization

main.cpp

Share

Run

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 vector<long long> dp(50, -1);
5
6 long long fibonacciLoopMemo(int n) {
7     if (n <= 1)
8         return n;
9     if (dp[n] != -1)
10        return dp[n];
11
12     dp[0] = 0;
13     dp[1] = 1;
14     for (int i = 2; i <= n; i++) {
15         dp[i] = dp[i - 1] + dp[i - 2];
16     }
17     return dp[n];
18 }
```

```

19
20 int main() {
21     struct timespec start, end;
22     clock_gettime(CLOCK_MONOTONIC, &start);
23
24     for (int i = 0; i < 50; i++) {
25         cout << fibonacciLoopMemo(i) << " ";
26     }
27     cout << endl;
28
29     clock_gettime(CLOCK_MONOTONIC, &end);
30     double time_taken = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start
        .tv_nsec) / 1e9;
31     cout << "Time taken by Loop with Memoization: " << time_taken << " seconds"
        << endl;
32
33     return 0;
34 }
35

```

Output

Clear

/tmp/areYodXfhH.o

```

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 1346269 2178309 3524578 5702887
9227465 14930352 24157817 39088169 63245986 102334155 165580141 267914296
433494437 701408733 1134903170 1836311903 2971215073 4807526976 7778742049
Time taken by Loop with Memoization: 4.873e-05 seconds

```

=== Code Execution Successful ===

CPU time : 4.873×10^{-5} seconds

Speed-up = Base program CPU time / Target program CPU time

Given, base program = Recursion program

Therefore,

1. Speedup for Loop program = 26,47,420.3984 [Loop program is 2647420.3984 times faster than Recursion program]
2. Speedup for Recursion using Memoization program = 27,83,440.7447 [Recursion Memoization program is 27,83,440.7447 times faster than Recursion program]
3. Speedup for Loop using Memoization program = 31,90,703.8785 [Loop Memoization program is 31,90,703.8785 times faster than Recursion program]

Q2.

Assumptions: Instead of running each program for 10 times, and taking average for best CPU time estimate, I have directly taken CPU time of 1 runtime of each program. Also, I have assumed that 'real time' given by the command prompt is the total execution time. The CPU time is calculated separately for all N and separately for int and double types and each bucket language. It is assumed that the meat portion time is time taken by the matrix multiplication only, ignoring time for initializing matrices, etc. I have initialized the two matrices to all 1's and all 2's respectively for calculation purposes to standardize the results.

i.e. Meat time = matrix_multiplication_time

Basket 1 - C++

INT type:

Code for N*N matrix multiplication

```
main.cpp
1  #include <bits/stdc++.h>
2  using namespace std;
3  using namespace chrono;
4
5  void MatrixMultiply(int N, vector<vector<int>> &A, vector<vector<int>> &B,
    vector<vector<int>> &result)
6  {
7      for (int i = 0; i < N; i++)
8      {
9          for (int j = 0; j < N; j++)
10         {
11             result[i][j] = 0;
12             for (int k = 0; k < N; ++k)
13             {
14                 result[i][j] += A[i][k] * B[k][j];
15             }
16         }
17     }
18 }
```

```

19 |
20 int main()
21 {
22     int N = 64 //128, 256, 512, 1024;
23     vector<vector<int>> A(N, vector<int>(N, 1));
24     vector<vector<int>> B(N, vector<int>(N, 2));
25     vector<vector<int>> result(N, vector<int>(N, 0));
26
27     auto start_meat = high_resolution_clock::now();
28     MatrixMultiply(N, A, B, result);
29     auto end_meat = high_resolution_clock::now();
30
31     auto duration_meat = duration_cast<nanoseconds>(end_meat - start_meat
32     ).count();
33     double matrix_multiplication_time = duration_meat * 1e-9;
34
35
36     cout << endl
37         << "Matrix multiplication time for size " << N << "x" << N << ": "
38         << matrix_multiplication_time << " seconds" << endl;
39
40     return 0;
41 }

```

Output:

1. 64x64

```

$ time ./shubham

Matrix multiplication time for size 64x64: 0.00214464 seconds

real    0m0.004s
user    0m0.004s
sys     0m0.000s

```

Proportion of meat part = Meat portion time / Total execution time

= 53.616%

2. 128x128

```
$ time ./shubham  
  
Matrix multiplication time for size 128x128: 0.0161415 seconds  
  
real    0m0.018s  
user    0m0.005s  
sys     0m0.006s
```

Proportion of meat part = Meat portion time / Total execution time

= 89.675%

3. 256x256

```
$ time ./shubham  
  
Matrix multiplication time for size 256x256: 0.146209 seconds  
  
real    0m0.149s  
user    0m0.132s  
sys     0m0.010s
```

Proportion of meat part = Meat portion time / Total execution time

= 98.127%

4. 512x512

```
$ time ./shubham  
  
Matrix multiplication time for size 512x512: 1.07639 seconds  
  
real    0m1.081s  
user    0m1.070s  
sys     0m0.011s
```

Proportion of meat part = Meat portion time / Total execution time

= 99.573%

5. 1024x1024

```
$ time ./shubham

Matrix multiplication time for size 1024x1024: 9.8471 seconds

real    0m9.855s
user    0m9.850s
sys     0m0.011s
```

Proportion of meat part = Meat portion time / Total execution time

= 99.919%

DOUBLE type:

Code for N*N matrix multiplication

```
main.cpp
1  #include <bits/stdc++.h>
2  using namespace std;
3  using namespace chrono;
4
5  void MatrixMultiply(int N, vector<vector<double>> &A, vector<vector<double>> &B
   , vector<vector<double>> &result)
6  {
7      for (int i = 0; i < N; i++)
8      {
9          for (int j = 0; j < N; j++)
10         {
11             result[i][j] = 0;
12             for (int k = 0; k < N; ++k)
13             {
14                 result[i][j] += A[i][k] * B[k][j];
15             }
16         }
17     }
18 }
19
```

```

19
20 int main()
21 {
22     int N = 64 //128, 256, 512, 1024;
23     vector<vector<double>> A(N, vector<double>(N, 1.0));
24     vector<vector<double>> B(N, vector<double>(N, 2.0));
25     vector<vector<double>> result(N, vector<double>(N, 0.0));
26
27     auto start_meat = high_resolution_clock::now();
28     MatrixMultiply(N, A, B, result);
29     auto end_meat = high_resolution_clock::now();
30
31     auto duration_meat = duration_cast<nanoseconds>(end_meat - start_meat
32         ).count();
33     double matrix_multiplication_time = duration_meat * 1e-9;
34
35
36     cout << endl
37         << "Matrix multiplication time for size " << N << "x" << N << ": "
38         << matrix_multiplication_time << " seconds" << endl;
39
40     return 0;
41 }

```

Output:

1. 64x64

```

$ time ./shubham

Matrix multiplication time for size 64x64: 0.00202636 seconds

real    0m0.023s
user    0m0.005s
sys     0m0.001s

```

Proportion of meat part = Meat portion time / Total execution time

= 88.102%

2. 128x128

```
$ time ./shubham

Matrix multiplication time for size 128x128: 0.0171143 seconds

real    0m0.019s
user    0m0.019s
sys     0m0.001s
```

Proportion of meat part = Meat portion time / Total execution time
= 90.075%

3. 256x256

```
$ time ./shubham

Matrix multiplication time for size 256x256: 0.152491 seconds

real    0m0.155s
user    0m0.145s
sys     0m0.010s
```

Proportion of meat part = Meat portion time / Total execution time
= 98.381%

4. 512x512

```
$ time ./shubham

Matrix multiplication time for size 512x512: 1.27971 seconds

real    0m1.304s
user    0m1.273s
sys     0m0.010s
```

Proportion of meat part = Meat portion time / Total execution time
= 98.137%

5. 1024x1024

```
$ time ./shubham

Matrix multiplication time for size 1024x1024: 10.8492 seconds

real    0m10.862s
user    0m10.853s
sys     0m0.010s
```

Proportion of meat part = Meat portion time / Total execution time
= 99.882%

Basket 2 - Python

INT type:

Code for N*N matrix multiplication

```
main.py  [Icons] [Share] [Run]

1  import time
2
3  def matrix_multiply_integer(N):
4      A = [[1 for _ in range(N)] for _ in range(N)]
5      B = [[2 for _ in range(N)] for _ in range(N)]
6      C = [[0 for _ in range(N)] for _ in range(N)]
7
8      start_meat_time = time.time()
9
10     for i in range(N):
11         for j in range(N):
12             for k in range(N):
13                 C[i][j] += A[i][k] * B[k][j]
14
15     end_meat_time = time.time()
16     matrix_multiplication_time = (end_meat_time - start_meat_time)
17
18     print(f"Matrix multiplication time for size {N}x{N}:
19           {matrix_multiplication_time} seconds")
20
21  N = 64 # 128, 256, 512, 1024
22  matrix_multiply_integer(N)
23  |
```

Output:

1. 64x64

```
$ time python3 shubham.py
Matrix multiplication time for size 64x64: 0.018562793731689453 seconds

real    0m0.035s
user    0m0.023s
sys     0m0.012s
```

Proportion of meat part = Meat portion time / Total execution time

= 53.036%

2. 128x128

```
$ time python3 shubham.py
Matrix multiplication time for size 128x128: 0.12119746208190918 seconds

real    0m0.133s
user    0m0.114s
sys     0m0.019s
```

Proportion of meat part = Meat portion time / Total execution time

= 91.126%

3. 256x256

```
$ time python3 shubham.py
Matrix multiplication time for size 256x256: 0.9826507568359375 seconds

real    0m0.998s
user    0m0.978s
sys     0m0.020s
```

Proportion of meat part = Meat portion time / Total execution time

= 98.462%

4. 512x512

```
$ time python3 shubham.py
Matrix multiplication time for size 512x512: 8.769368410110474 seconds

real    0m8.807s
user    0m8.776s
sys     0m0.031s
```

Proportion of meat part = Meat portion time / Total execution time

= 99.573%

5. 1024x1024

```
$ time python3 shubham.py
Matrix multiplication time for size 1024x1024: 70.26233696937561 seconds





real    1m10.381s
user    1m10.199s
sys     0m0.181s
```

Proportion of meat part = Meat portion time / Total execution time

= 99.831%

DOUBLE type:

Code for N*N matrix multiplication

```
main.py    Share  Run

1 import time
2
3 def matrix_multiply_integer(N):
4     A = [[1.0 for _ in range(N)] for _ in range(N)]
5     B = [[2.0 for _ in range(N)] for _ in range(N)]
6     C = [[0 for _ in range(N)] for _ in range(N)]
7
8     start_meat_time = time.time()
9
10    for i in range(N):
11        for j in range(N):
12            for k in range(N):
13                C[i][j] += A[i][k] * B[k][j]
14
15    end_meat_time = time.time()
16    matrix_multiplication_time = (end_meat_time - start_meat_time)
17
18    print(f"Matrix multiplication time for size {N}x{N}:
19          {matrix_multiplication_time} seconds")
20
21 N = 64 # 128, 256, 512, 1024
22 matrix_multiply_integer(N)
23 |
```

Output:

1. 64x64

```
$ time python3 shubham.py
Matrix multiplication time for size 64x64: 0.015418767929077148 seconds

real    0m0.053s
user    0m0.031s
sys     0m0.001s
```

Proportion of meat part = Meat portion time / Total execution time

= 29.092%

2. 128x128

```
$ time python3 shubham.py
Matrix multiplication time for size 128x128: 0.16347956657409668 seconds

real    0m0.187s
user    0m0.163s
sys     0m0.020s
```

Proportion of meat part = Meat portion time / Total execution time

$$= 87.422\%$$

3. 256x256

```
$ time python3 shubham.py
Matrix multiplication time for size 256x256: 1.312356948852539 seconds

real    0m1.335s
user    0m1.295s
sys     0m0.030s
```

Proportion of meat part = Meat portion time / Total execution time

$$= 98.304\%$$

4. 512x512

```
$ time python3 shubham.py
Matrix multiplication time for size 512x512: 11.754274368286133 seconds

real    0m11.951s
user    0m11.760s
sys     0m0.081s
```

Proportion of meat part = Meat portion time / Total execution time

$$= 98.354\%$$

5. 1024x1024

```
$ time python3 shubham.py
Matrix multiplication time for size 1024x1024: 119.53874039649963 seconds

real    1m59.748s
user    1m59.528s
sys      0m0.191s
```

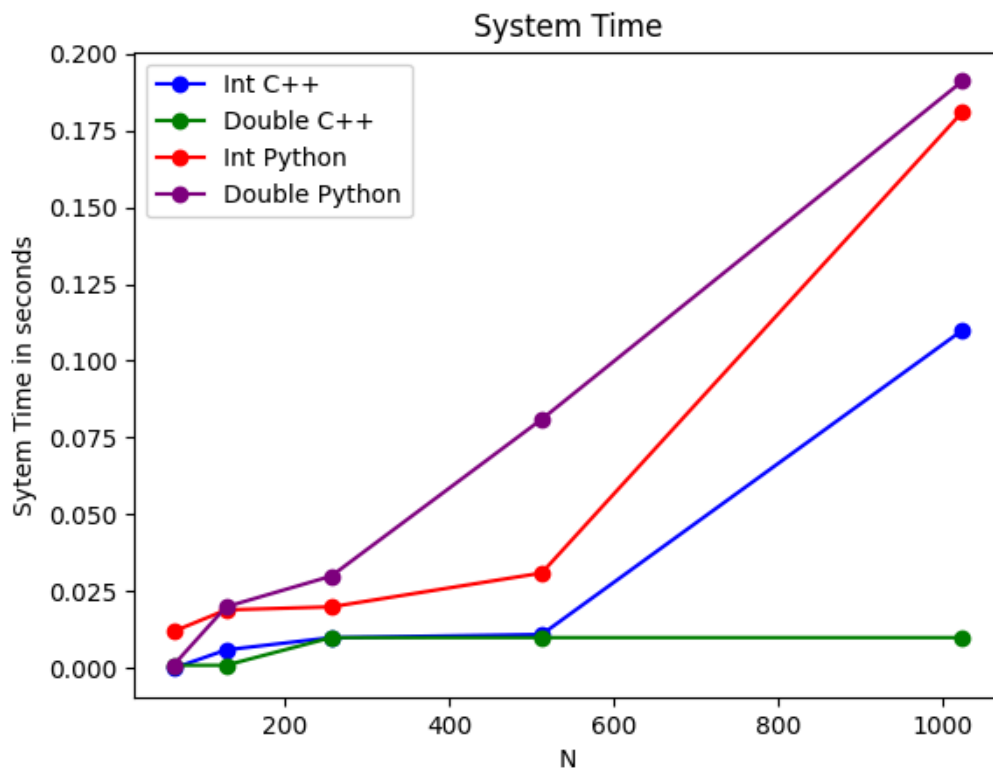
Proportion of meat part = Meat portion time / Total execution time

= 99.825%

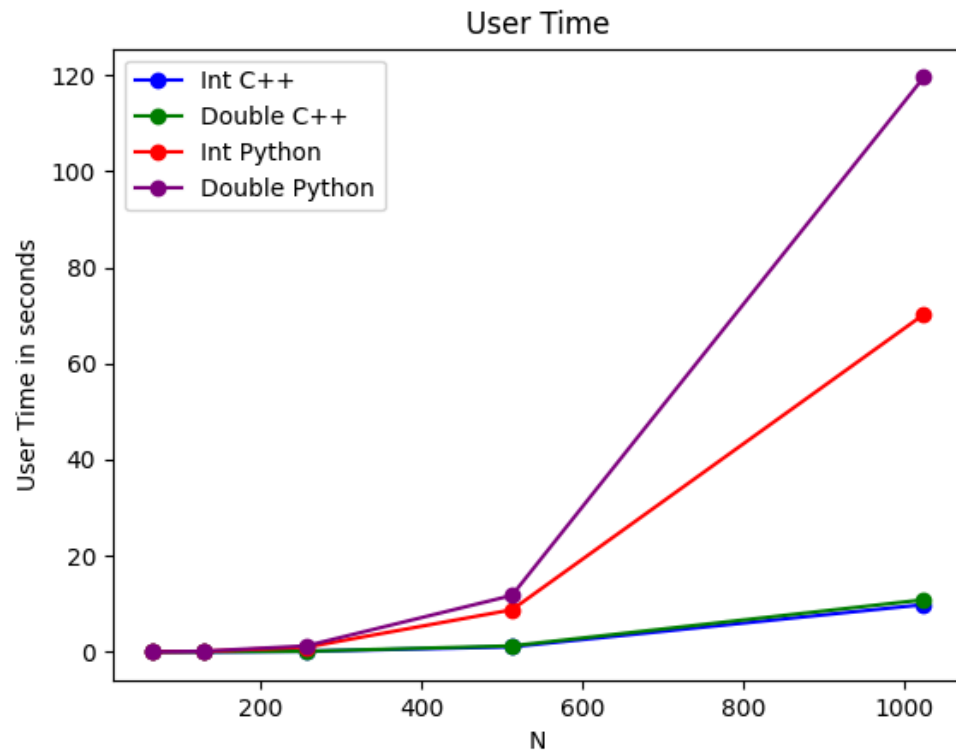
Part C:

Graphs

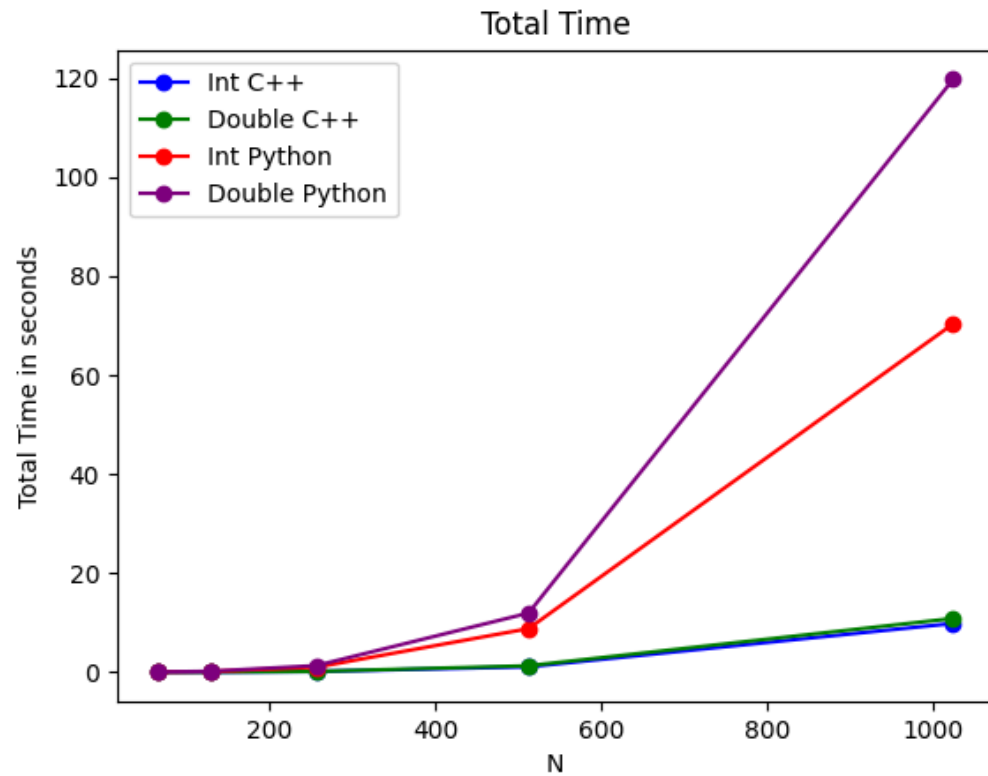
1. System time



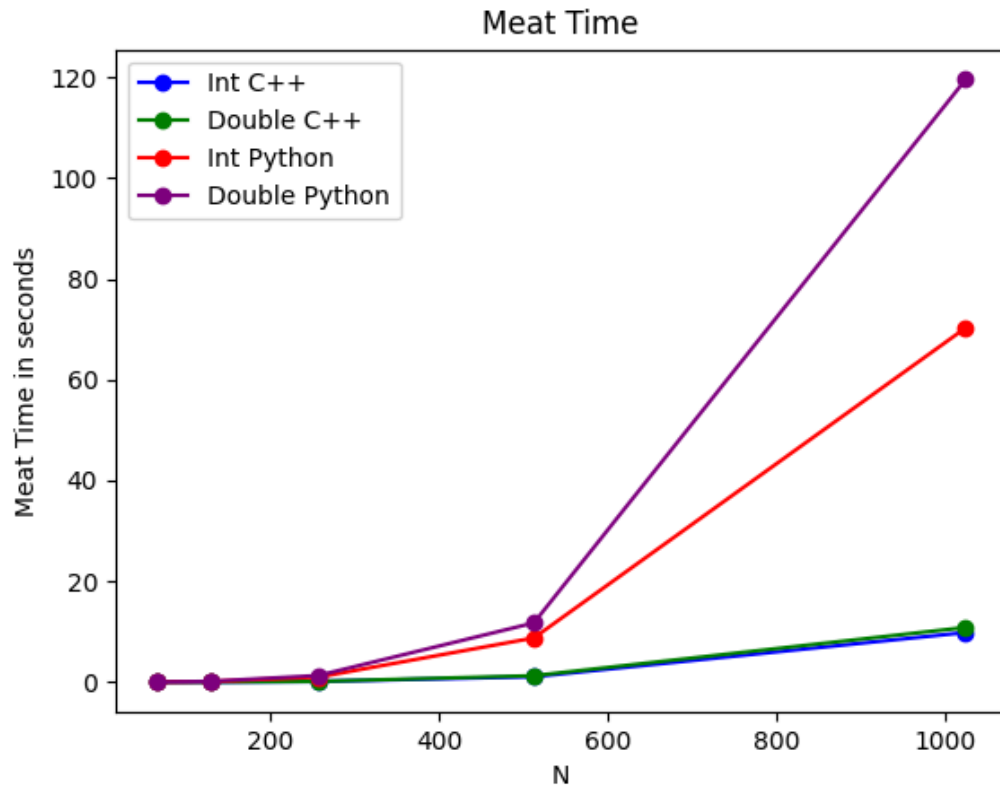
2. User time



3. Total execution(Real) time



4. Meat portion execution time



Observations:

- For each graph, the time increases with N because the number of computations expands exponentially as N grows.
- Double takes longer to compute than int in both C++ and Python, as double involves more complex calculations.
- Furthermore, Python takes more time compared to C++ since Python is an interpreted language, which tends to be slower than the compiled nature of C++.