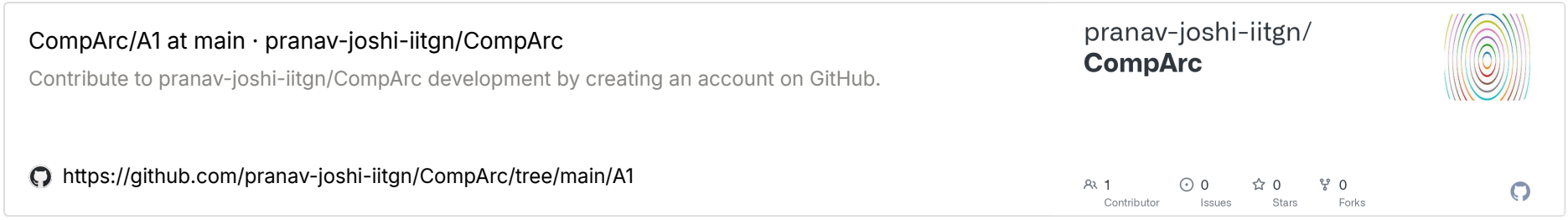
Assignment 1

Git-Hub Repository

This is where all of the code I wrote for this assignment is stored. All the file names and descriptions will be exactly the same as in this repository.



Question 1

1. Implement a program(s) to list the first 50 fibonacci numbers preferably in C/C++ in the following manner: (***Total: 50 points***)
    a. Using recursion (***10 points***)
    b. Using loop (***10 points***)
    c. Using recursion and memoization (***10 points***)
    d. Using loop and memoization (***10 points***)

    Find the speedup of all the programs on your machine by keeping program (1) as the baseline. (***10 points***).

    Tips: Measure the time taken by the program on the CPU using timespec.

Code

```
#include <stdio.h>
#include <time.h>
long int fibo_rec(int n){
    long int x;
    if(n==1){x = 1;return x;}
    else if(n==2){x = 2;return x;}
    else if(n>2){return fibo_rec(n-1) + fibo_rec(n-2);}
    else {x = 0 ;return x;}
}
void Print_fib_rec_memo(int n,long result[2]){
    long int x;
    if(n==1){
        result[0] = 0;
        result[1] = 1;
        printf("%ld\n",result[1]);
    } else if(n==2){
        result[0] = 1;
        result[1] = 1;
        printf("%ld\n",result[1]);
    } else {
        Print_fib_rec_memo(n-1,result);
        x = result[0] + result[1];
        result[0] = result[1];
        result[1] = x;
        printf("%ld\n",result[1]);
    }
}
void Print_fib_loop(){
```

```
    long int x,y,z;
    x = 0; // fib(0)
    y = 1; // fib(1)
    for(int i=1;i<=50;i++){
        printf("%ld\n",y);
        z = x + y;
        x = y;
        y = z;
    }
}
void Print_fibo_rec(){
    for(int i=1;i<=50;i++){
        printf("%ld\n",fibo_rec(i));
    }
}
void Print_fibo_loop_memo(){
    long F[51];
    F[0] = 0;
    F[1] = 1;
    printf("%ld",F[1]);
    for(int i = 2;i<=50;i++){
        F[i] = F[i-1] + F[i-2];
        printf("%ld\n",F[i]);
    }
}
void method(int x){
    switch(x){
        case 0:Print_fibo_rec();
        case 1:Print_fib_loop();
        case 2:
            long result[2];
            Print_fib_rec_memo(50,result);
        case 3:Print_fibo_loop_memo();
    }
}
int main(){
    struct timespec t;
    long long T[4];
    long long t0s,t1s,t0ns,t1ns;
    long long scales[4] = {1e9,1e6,1e6,1};
    int reps = 3;
    for(int q=0;q<4;q++){
        timespec_get(&t,TIME_UTC);
        t0ns = t.tv_nsec;
        t0s = t.tv_sec;
        for(int i = 0;i<reps;i++){method(q);}
        timespec_get(&t,TIME_UTC);
        t1ns = t.tv_nsec;
        t1s = t.tv_sec;
        T[q] = (t1s - t0s)*(1e9/scales[q]) + (t1ns - t0ns)/scales[q];
        T[q] = T[q] / reps;
    }
    printf("Recursion         : Time = %lld s\n",T[0]);
    printf("Loop              : Time = %lld ms\n",T[1]);
    printf("Recur. with memo. : Time = %lld ms\n",T[2]);
    printf("Loop with memo.   : Time = %lld ns\n",T[3]);
```

```
    return 0;
}
```

Output

```
Recursion         : Time = 218 s
Loop              : Time = 5 ms
Recur. with memo. : Time = 3 ms
Loop with memo.   : Time = 1342985 ns
```

Calculations

I assume by “program (1)” , we mean the recursion method.

Method	time (s)	speedup
Recursion	218	1.0
Loop	5 * 10^-3	43600.0
Recur. with memo.	3 * 10^-3	72666.67
Loop with memo.	1342985 * 10^-9	162324.97

Question 2

2. Write a simple Matrix Multiplication program for a given NxN matrix in any two of your preferred Languages from the following listed buckets, where N is iterated through the set of values 64, 128, 256, 512 and 1024. N can either be hardcoded or specified as input. Consider two cases (a) Elements of matrix are of data type **Integer** and (b) **Double** In each case, (*i.e.* Bucket 1 for (a) and (b) + Bucket 2 for (a) and(b)) (***Total: 100 points***)

   Bucket1: C, C++, Go
   Bucket2: Python, Java.

   a. Report the output of the ‘**time**’ describing the system and CPU times. (***25 points***)
   b. Using the ‘***language hooks***’ evaluate the execution time for the meat portions of the program and how much proportion is it w.r.t. total program execution time. (***25 points***)
   c. Plot the (a) and (b) execution times for each of the iterations. And compare the performance (System and Program execution times) of the program for given value of N for the languages in both the buckets. –Illustrate your observations. (***50 points***)

Bucket 1 (C)

Individual programs were created for each value of N and the data-type that we are dealing with. This is so that `time` can be used directly on each program to give the total execution time. Moreover this allows us to set the scale at which `cpu_time` is measured (using language hooks) to an optimal value for each program, thus reducing overflows and other issues.

For example, for $N = 64$ and Integer data-type, the program `Q2I2to6.c` was created :

```
#include <stdio.h>
#include <time.h>
long long MulInt(int a,int b, int c, int M1[a][b],int M2[b][c], int M3[a][c],int scale){
    long long T = 0;
    long long T0s,T1s,T0ns,T1ns,Ts,Tns;
    struct timespec t;
    timespec_get(&t,TIME_UTC);
    T0s = t.tv_sec;
    T0ns = t.tv_nsec;
    for(int i=0;i<a;i++){
        for(int j=0;j<c;j++){
```

```
            M3[i][j] = 0;
            for(int k=0;k<b;k++){
                M3[i][j] += M1[i][k] * M2[k][j];
            }
        }
    }
    timespec_get(&t,TIME_UTC);
    T1s = t.tv_sec;
    T1ns = t.tv_nsec;
    Ts = T1s - T0s;
    Tns = T1ns - T0ns;
    T = Ts * (1e9/scale) + Tns / scale;
    return T;
}
int N = 64;
int M1[64][64];
int M2[64][64];
int M3[64][64];
int main(){
    struct timespec t;
    long long cpu_time;
    cpu_time = MulInt(N,N,N,M1,M2,M3,1);
    printf("\nN = %d\ncpu\t%llde-9 s",N,cpu_time);
    return 0;
}
```

Similarly, all such programs with naming convention `Q2<data-type>2to<power of 2>.c` were created. The main output of any of these programs is the `cpu_time` which is the time required for running the meat portion of the program, i.e. the matrix multiplication.

Then, these C programs were compiled and run using a bash script `manage.sh` which stores the output of each program, as well as the output of the `time` command into a text file named `Q2O.txt` .

▼ `manage.sh`

```
#!/bin/bash
echo "Integers" > Q2O.txt
for (( n=6; n<=10; n++ ))
do
gcc "Q2I2to$n.c" -o executable;
{ time ./executable >> Q2O.txt ; } 2>> "Q2O.txt";
done
echo "" >> Q2O.txt ;
echo "Doubles" >> Q2O.txt ;
for (( n=6; n<=10; n++ ))
do
gcc "Q2D2to$n.c" -o executable;
{ time ./executable >> Q2O.txt ; } 2>> "Q2O.txt";
done
```

Then, the output was converted to tables of values in a file `extracted.md` using the python script `extract.py` .

Bucket 2 (Python)

Python program to time matrix multiplication

```
from sys import stdin
from time import time_ns
datatype_size = stdin.read()
```

```
datatype_size = datatype_size.split()
datatype,n = datatype_size
n = int(n)
N = 2**n

if datatype == "int":
    M1 = [list(range(N)) for i in range(N)]
    M2 = [list(range(N)) for i in range(N)]
    M3 = [list(range(N)) for i in range(N)]
elif datatype == "float":
    M1 = [[j + 0.5 for j in range(N)] for i in range(N)]
    M2 = [[j + 0.5 for j in range(N)] for i in range(N)]
    M3 = [[j + 0.5 for j in range(N)] for i in range(N)]
else: raise ValueError("datatype (first argument) can only be 'int' or 'float'")

t0 = time_ns()
for i in range(N):
    for j in range(N):
        M3[i][j] = 0
        for k in range(N):
            M3[i][j] += M1[i][k] * M2[i][j]
t1 = time_ns()

print(f"\nN = {N}")
print(f"cpu_time = {t1-t0} ns",end="")
```

This program takes the data type and the value of $n = \log_2(N)$ as input and multiplies 2 matrices of that size. Then it outputs the value of N and the time required for the matrix multiplication to the bash shell.

This output, along with the output of `time` is stored into the file `Q2O_py.txt` using the bash script `manage_py.sh` .

Then, this output is converted to tables of values in the file `extracted_py.md` using the python script `extract_py.py` .

Final Data set

Every value of dimensions of time is in seconds in this database.

▼ C `int`

N	cpu_time	exec_time	user_time	sys_time
64	1621021e-9	0.003	0.003	0.001
128	89e-4	0.010	0.010	0.000
256	77e-3	0.079	0.079	0.000
512	578e-3	0.579	0.574	0.004
1024	4909e-3	4.911	4.894	0.012

▼ C `double`

N	cpu_time	exec_time	user_time	sys_time
64	1126323e-9	0.003	0.001	0.002
128	11944e-6	0.014	0.011	0.003
256	81e-3	0.083	0.079	0.004
512	597e-3	0.598	0.594	0.004
1024	5562e-3	5.565	5.553	0.012

▼ Python `int`

N	cpu_time	exec_time	user_time	sys_time
64	120058371e-9	0.161	0.155	0.004

N	cpu_time	exec_time	user_time	sys_time
128	908691639e-9	0.949	0.93	0.013
256	7291797284e-9	7.337	7.279	0.024
512	53320719833e-9	53.412	53.153	0.073
1024	415752594392e-9	415.97	412.336	0.252

▼ Python `float`

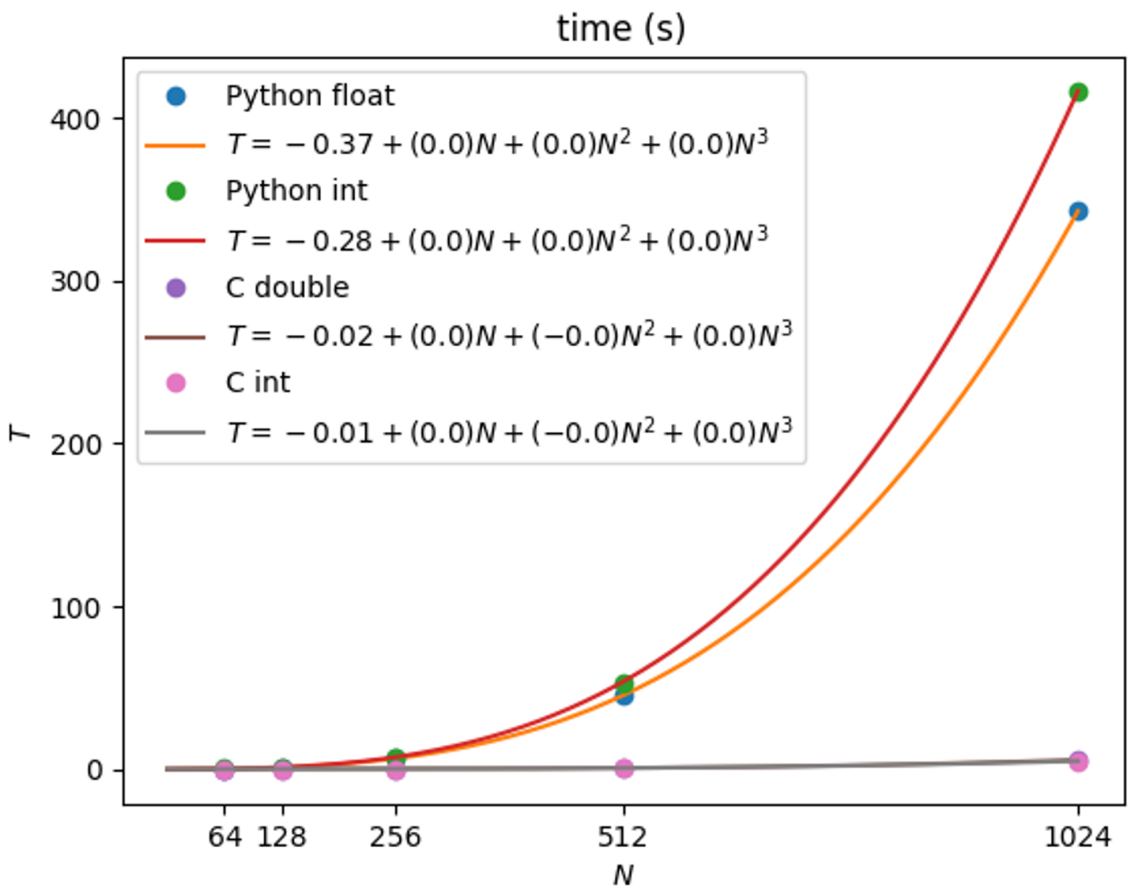
N	cpu_time	exec_time	user_time	sys_time
64	68849907e-9	0.095	0.091	0.004
128	770499453e-9	0.832	0.813	0.02
256	6401190748e-9	6.476	6.449	0.016
512	44882045281e-9	45.067	44.939	0.056
1024	341909757508e-9	342.339	341.442	0.172

These databases are stored as CSV files with names `Q2IntC.csv`, `Q2DoubleC.csv`, `Q2IntPy.csv`, `Q2FloatPy.csv` respectively.

Plots

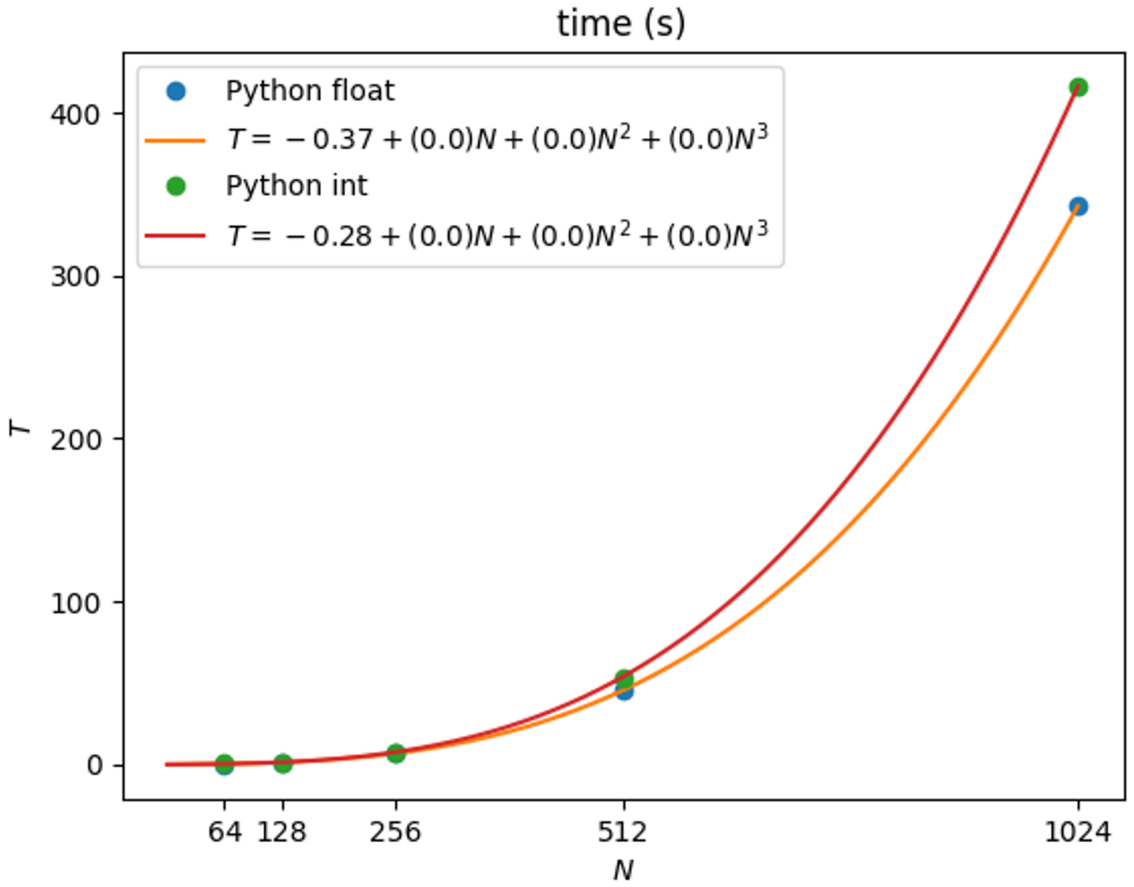
The data-set is used in the `Plot.ipynb` notebook to create these plots

Execution time

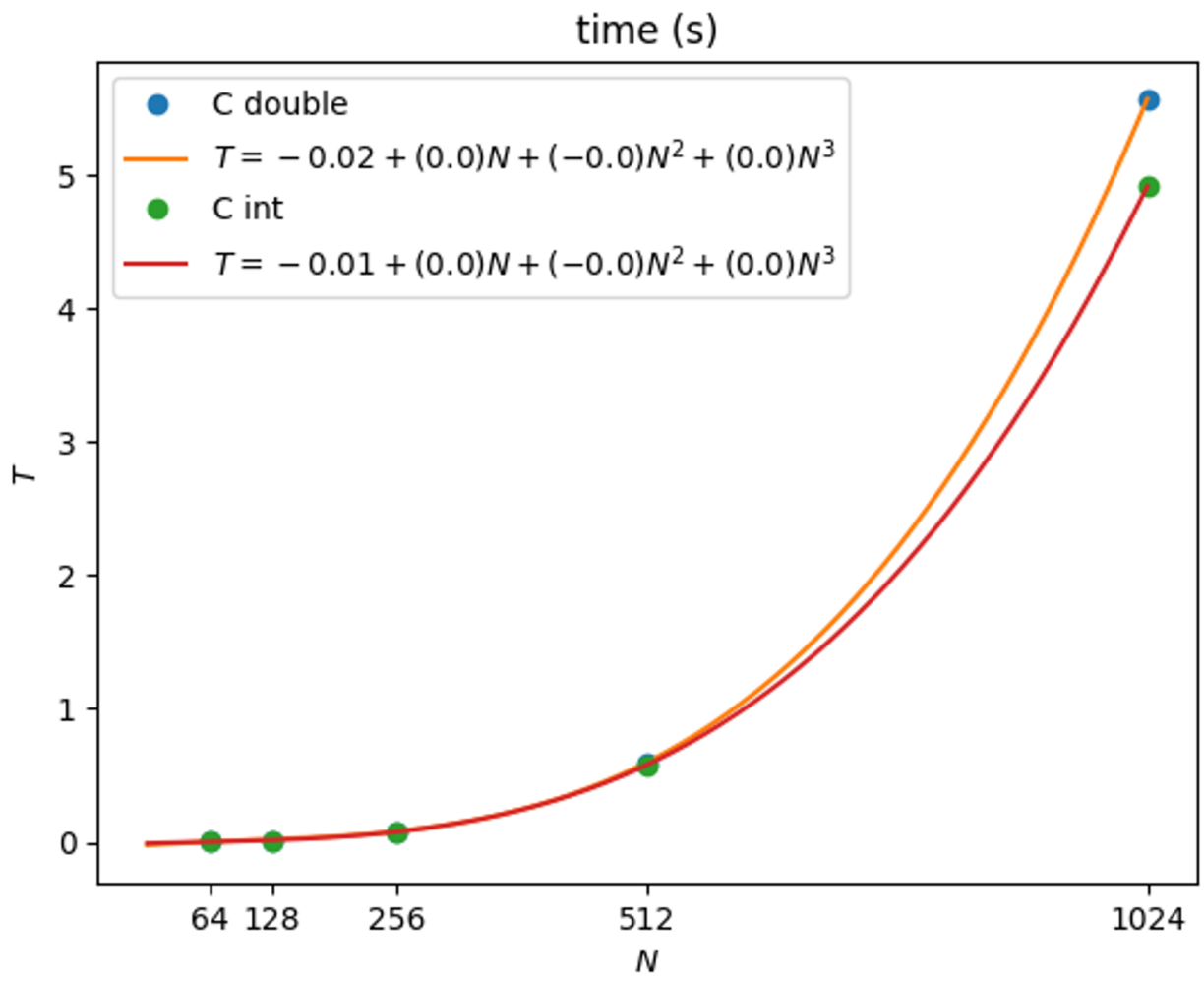


Clearly, C is much faster than Python. To compare between data types in the same language, these plots can be used :

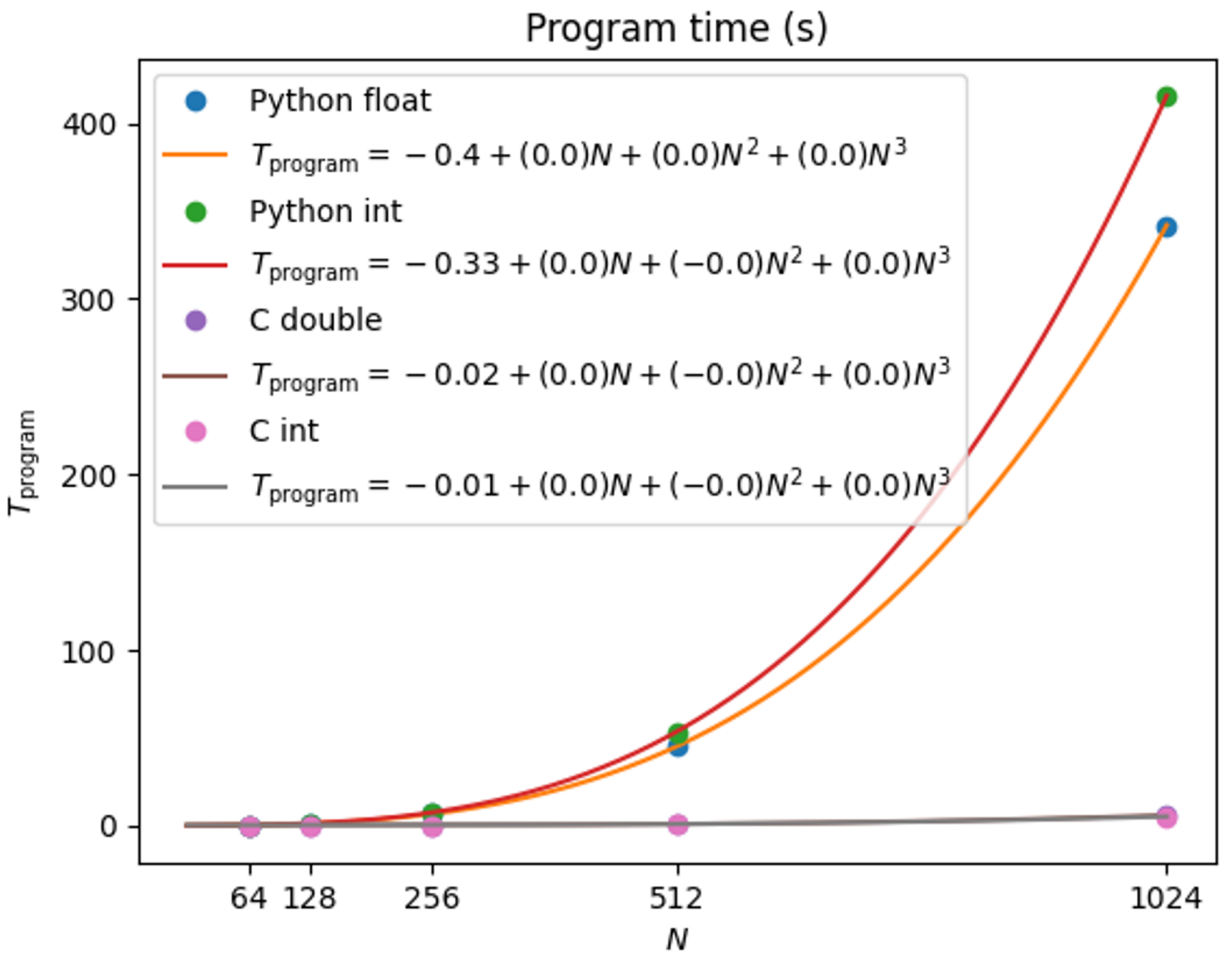
Python

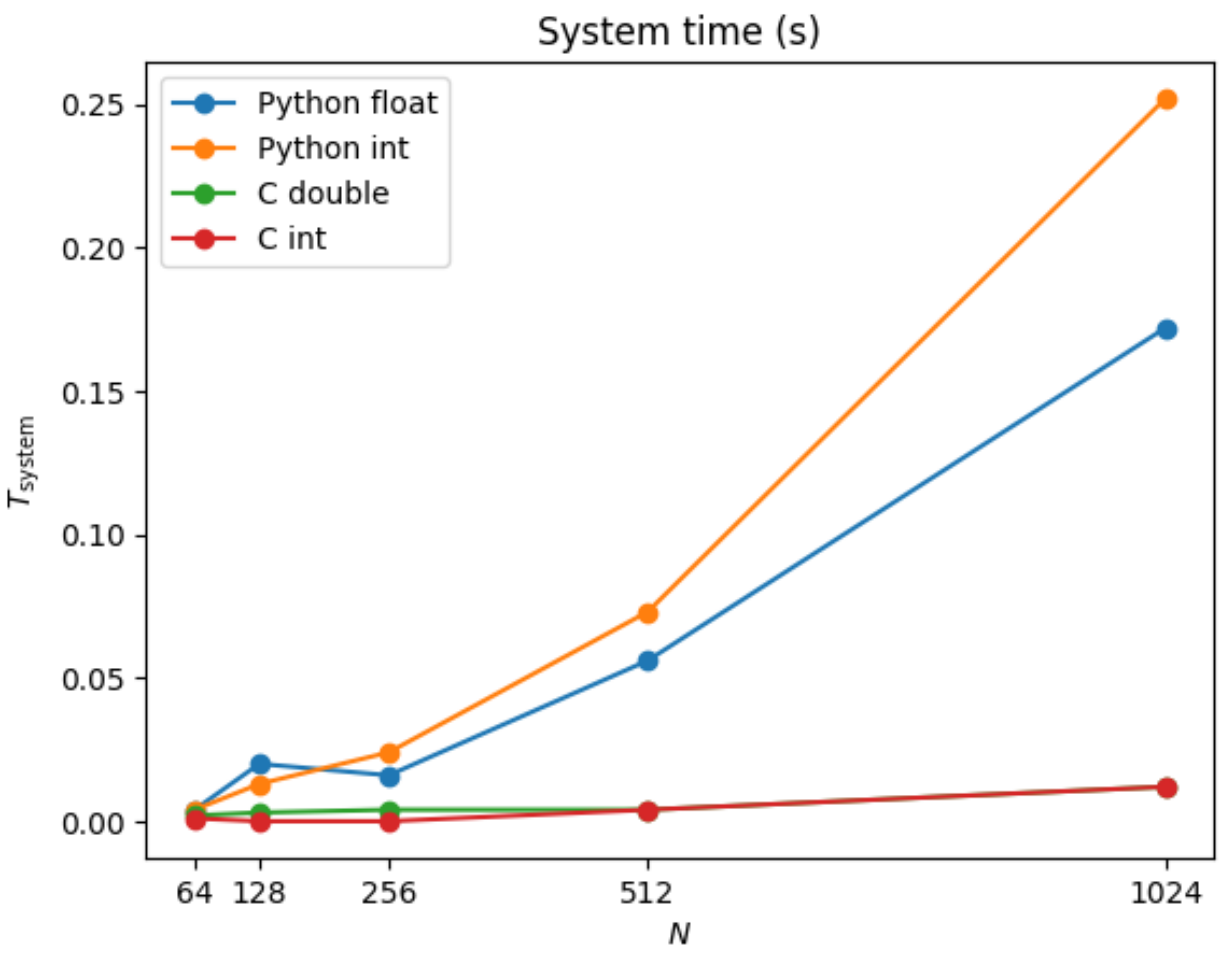


c

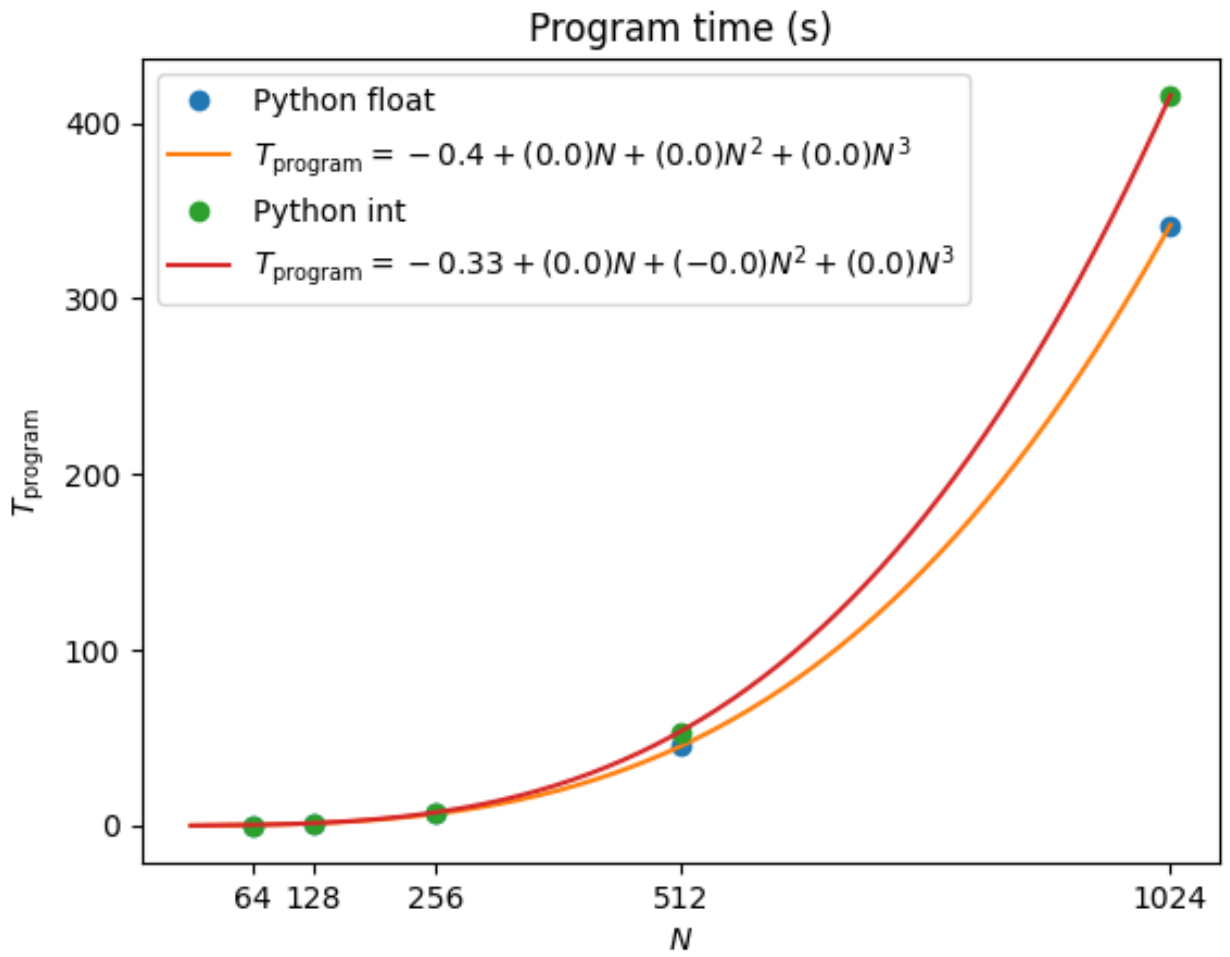


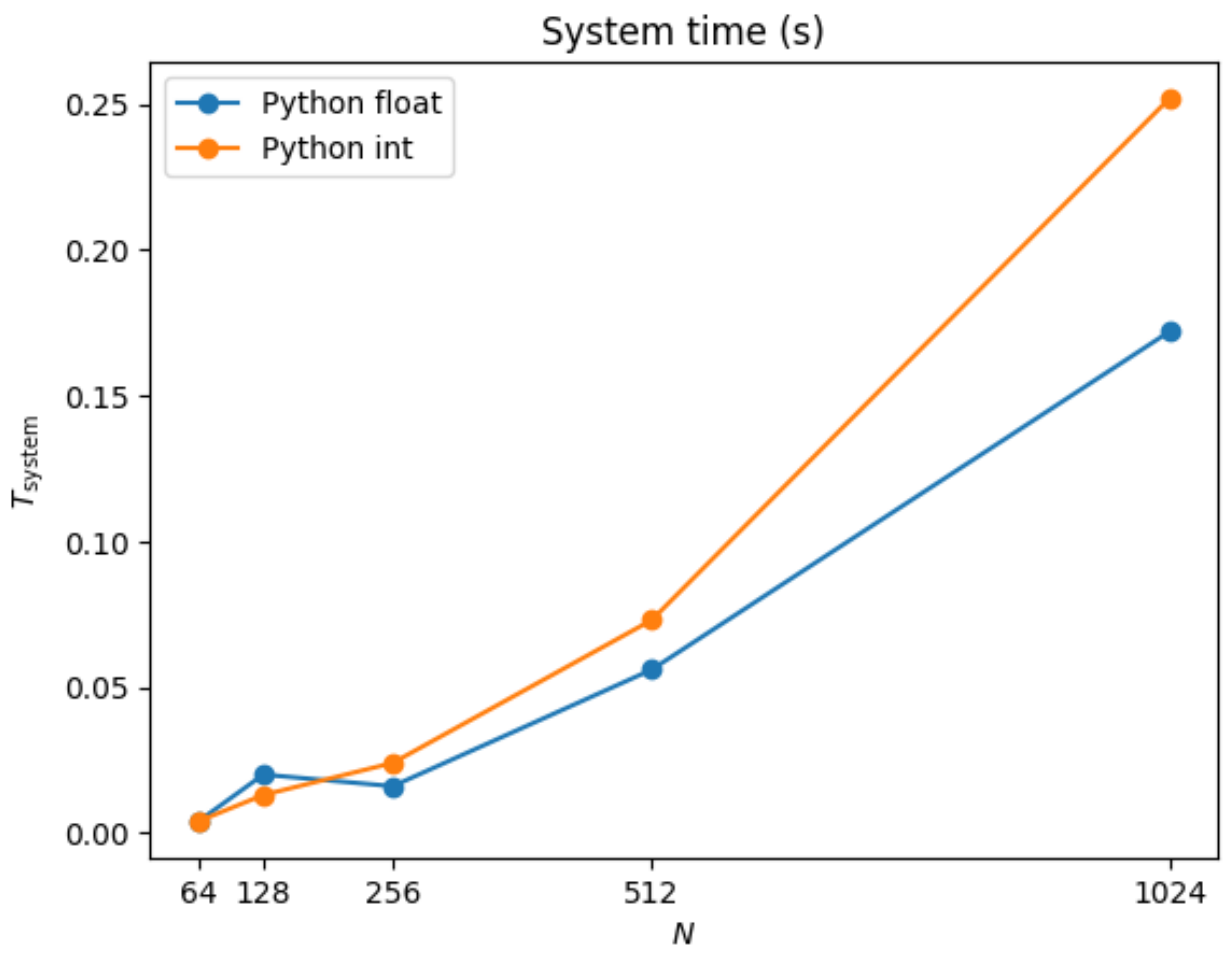
Program time and System time


Program time (s)
Python float
$T_{\text{program}} = -0.4 + (0.0)N + (0.0)N^2 + (0.0)N^3$
Python int
$T_{\text{program}} = -0.33 + (0.0)N + (-0.0)N^2 + (0.0)N^3$
C double
$T_{\text{program}} = -0.02 + (0.0)N + (-0.0)N^2 + (0.0)N^3$
C int
$T_{\text{program}} = -0.01 + (0.0)N + (-0.0)N^2 + (0.0)N^3$
T_{program}
0
100
200
300
400
64
128
256
512
1024
N

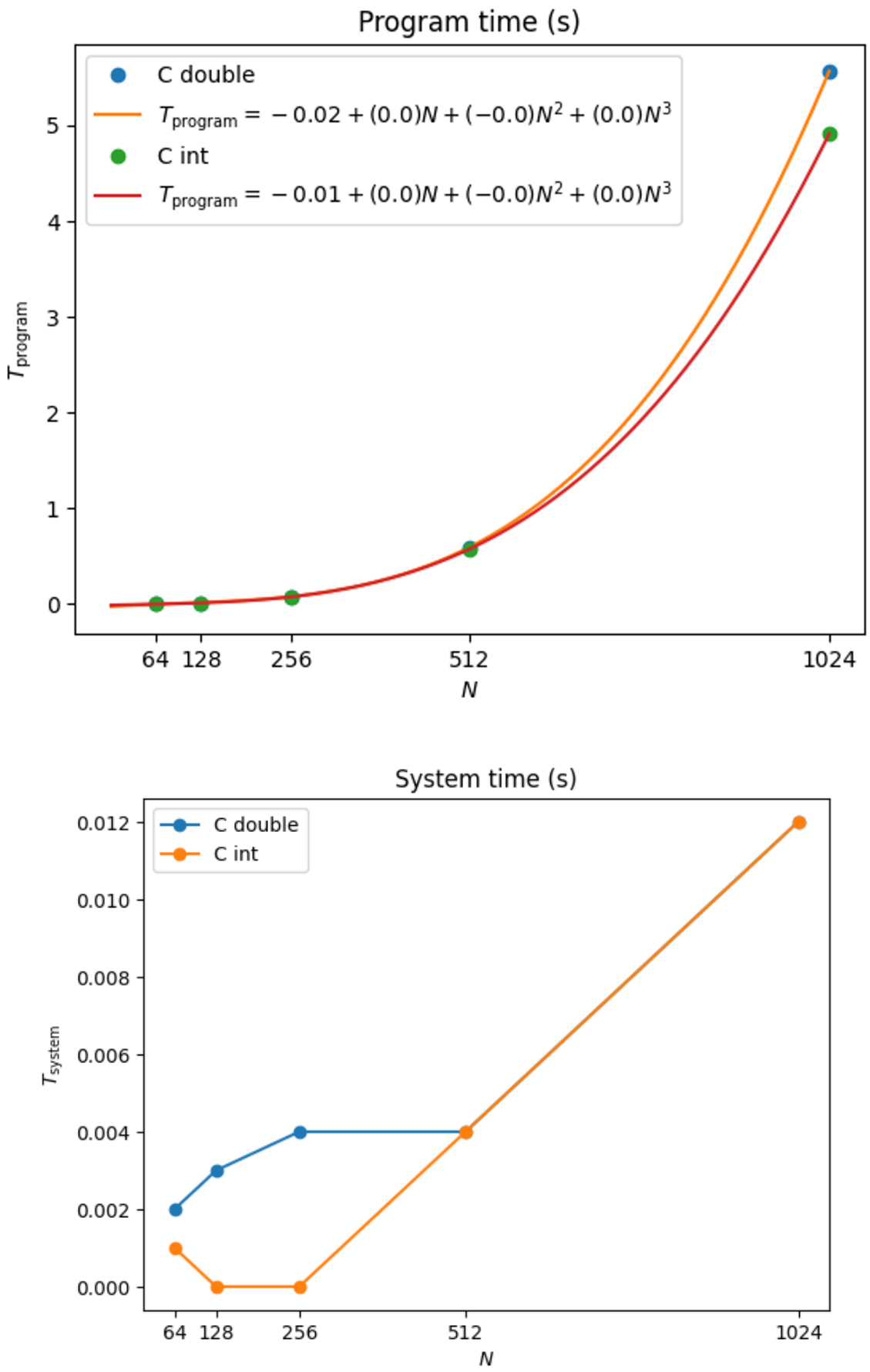

System time (s)
Python float
Python int
C double
C int
T_{system}
0.00
0.05
0.10
0.15
0.20
0.25
64
128
256
512
1024
N

Python


Program time (s)
Python float
$T_{\text{program}} = -0.4 + (0.0)N + (0.0)N^2 + (0.0)N^3$
Python int
$T_{\text{program}} = -0.33 + (0.0)N + (-0.0)N^2 + (0.0)N^3$
T_{program}
0
100
200
300
400
64
128
256
512
1024
N


System time (s)
Python float
Python int
T_{system}
0.00
0.05
0.10
0.15
0.20
0.25
64
128
256
512
1024
N

C


Program time (s)
C double
$T_{program} = -0.02 + (0.0)N + (-0.0)N^2 + (0.0)N^3$
C int
$T_{program} = -0.01 + (0.0)N + (-0.0)N^2 + (0.0)N^3$
$T_{program}$
0
1
2
3
4
5
64
128
256
512
1024
N
System time (s)
C double
C int
T_{system}
0.000
0.002
0.004
0.006
0.008
0.010
0.012
64
128
256
512
1024
N