

Family Name : Chabane Chaouche
First Name : Rabah
ID : 222231485010
Soft ING 3rd

LAB2 : Primality Test

Part 1 :

Q1 :

Algorithm Prime Number

```

1: Var
2: n, cmpt, i integer;
3: Begin
4: cmpt  $\leftarrow$  0;
5: i  $\leftarrow$  1;
6: print('Input Integer N>1 : ')
7: Read(n);
8: while i  $\leq$  n do
9:   if ( n%i == 0 ) then
10:     cmpt  $\leftarrow$  cmpt + 1;
11:   end if
12:   i  $\leftarrow$  i + 1;
13: end while
14: if ( cmpt == 2 ) then
15:   print('Prime Number');
16: else
17:   print('Not A Prime Number');
18: end if
19: End

```

Q2 :

we have $f(n) = \sum fr$ since $f(n)$ is sum of frequency of execution (fr) we need to figure out the fr of each instruction and sum them :

Best Case (N is prime)

cmpt \leftarrow 0	$fr = 1$ (one affectation)
i \leftarrow 1	$fr = 1$ (one affectation)
print ('Input Integer N>1 : ')	$fr = 1$ (one print)
Read (n)	$fr = 1$ (one read)

```

while i <= n do
  cmpt ← cmpt + 1
  i ← i + 1
  print('Prime Number')

```

$fr = n + 1$ (check the while condition $n+1$ times)
 $fr = 4$ (one affectation and one arithmetic operation (2) , repeated twice because N is Prime)
 $fr = 2n$ (one affectation and one arithmetic operation (2) , inside a while that loops n times (2n))
 $fr = 1$ (one print)

$$\begin{aligned}
 f_1(n) &= \sum fr \\
 &= 1 + 1 + 1 + 1 + (n + 1) + 4 + 2n + 1 \\
 &= 3n + 10 \\
 &= \boxed{3n + 10}
 \end{aligned}$$

now that we have the complexity function $f_1(n)$ we need to find the $T_1(n)$ we have $T_1(n) = f_1(n) \times \Delta t$

$$\begin{aligned}
 T_1(n) &= f_1(n) \times \Delta t \\
 &= (3n + 10) \times \Delta t \\
 &= \underbrace{3\Delta t}_{a_1} n + \underbrace{\Delta t 10}_{b_1} \\
 &= \boxed{a_1 n + b_1}
 \end{aligned}$$

Worst Case (N is not prime)

```

cmpt ← 0
i ← 1
print('Input Integer N>1 : ')
Read(n)
while i <= n do
  cmpt ← cmpt + 1
  i ← i + 1
  print('Not A Prime Number')

```

$fr = 1$ (one affectation)
 $fr = 1$ (one affectation)
 $fr = 1$ (one print)
 $fr = 1$ (one read)
 $fr = n + 1$ (check the while condition $n+1$ times)
 $fr = n$ (one affectation and one arithmetic operation (2) , repeated at worst $\frac{n}{2}$ time)
 $fr = 2n$ (one affectation and one arithmetic operation (2) , inside a while that loops n times (2n))
 $fr = 1$ (one print)

$$\begin{aligned}
 f_2(n) &= \sum fr \\
 &= 1 + 1 + 1 + 1 + (n + 1) + n + 2n + 1 \\
 &= 4n + 6 \\
 &= \boxed{4n + 6}
 \end{aligned}$$

now that we have the complexity function $f_2(n)$ we need to find the $T_2(n)$ we have $T_2(n) = f_2(n) \times \Delta t$

$$\begin{aligned}
T_2(n) &= f_2(n) \times \Delta t \\
&= (3n + 10) \times \Delta t \\
&= \underbrace{3\Delta t}_{a_2} n + \underbrace{\Delta t 10}_{b_2} \\
&= \boxed{a_2 n + b_2}
\end{aligned}$$

Conclusion

Both $T_1(n)$ and $T_2(n)$ are linear complexity therefore they both $\sim O(n)$

Q3 :

```

1  #include<stdio.h>
2  #include<time.h>
3
4  int main() {
5
6  int i = 1;
7  int cmpt = 0;
8  int n;
9
10
11  clock_t start_time = clock();
12
13  printf("Input Integer N>1 : ");
14  scanf("%d",&n);
15
16  while (i<=n) {
17      if ( n%i==0 ) {++cmpt;}
18      ++i;
19  }
20
21  if ( cmpt == 2 ) {printf("\nprime number");}
22
23      else {printf("\nNot a prime number");}
24
25  clock_t end_time = clock();
26
27  double execution_time = (double) (end_time - start_time)/CLOCKS_PER_SEC;
28
29  printf("\nExecution Time %f seconds\n",execution_time);
30
31
32  return 0;
33
34  }

```

Q3.a :

All the given N numbers are prime number

Q3.b :**Experimental**

N	1000003	2000003	4000037	8000009	16000057	32000011	64000031	128000003	256000001	512000009
$T(n)$ 10^{-3}	3.643	10.051	12.12	24.284	48.268	99.244	191.605	380.24	753.837	1517.66

N	1024000009	2048000011
$T(n)$ 10^{-3}	3042.77	6038.826

Theoretical

We first need to find Δt , for that we will take one runtime value from the experimental study and solve a simple equation for $n = 8000009$ and execution time $T(n) = 24.284 \times 10^{-3}$:

$$f(n) \times \Delta t = T(n)$$

$$\Delta t = \frac{T(n)}{f(n)}$$

$$\Delta t = \frac{T(n)}{3n + 10}$$

$$\Delta t = \frac{24.284 \times 10^{-3}}{3 \times 8000009 + 10}$$

$$\Delta t = 1.01 \times 10^{-9}$$

Theoretical Best Case

N	1000003	2000003	4000037	8000009	16000057	32000011	64000031	128000003	256000001	512000009
$T(n)$ 10^{-3}	3.03	6.06	12.12	24.24	48.48	96.96	193.9	387.84	775.837	1551.13

N	1024000009	2048000011
$T(n)$ 10^{-3}	3102.72	6205.4

Q3.c :

Observation

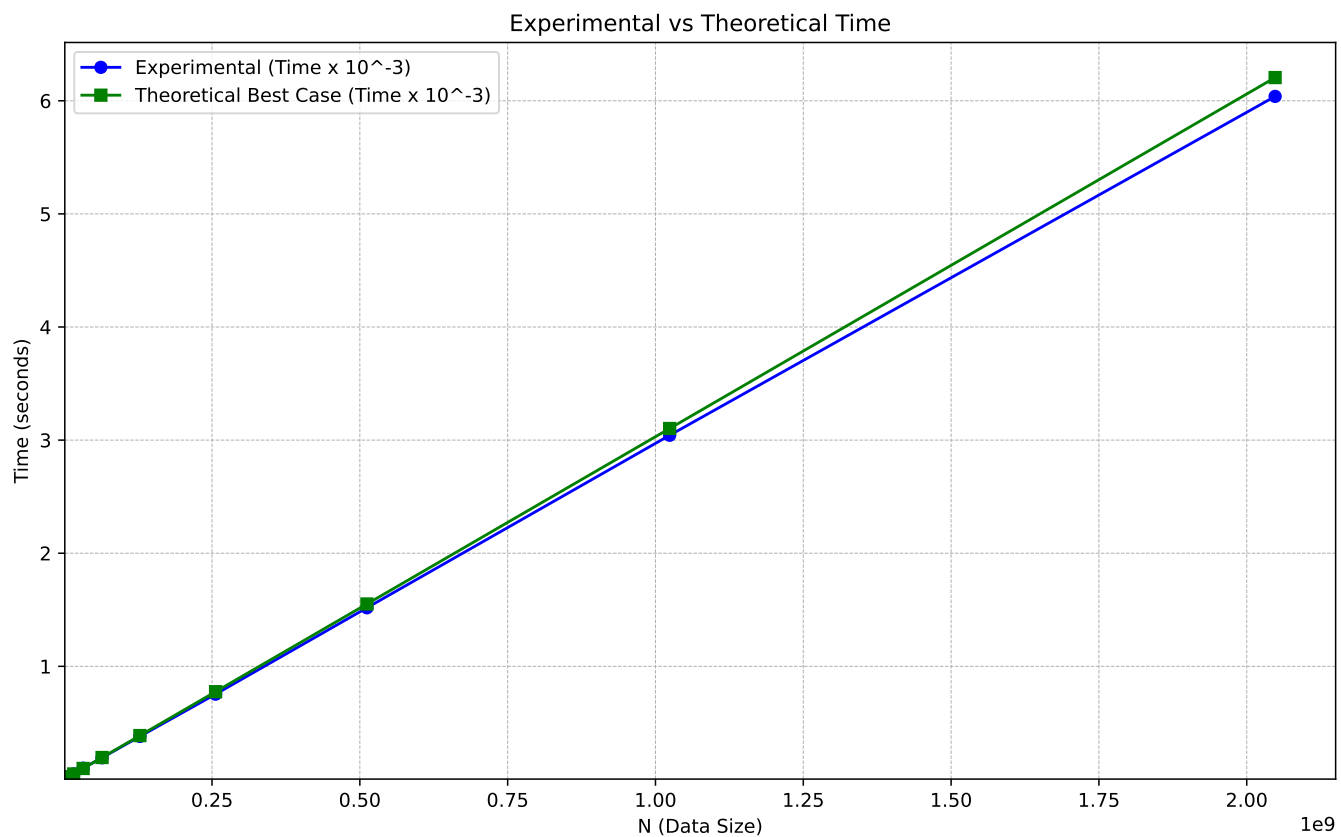
We notice that from the data and measurements obtained that the growth of the time complexity is linear $\sim O(n)$

Q3.d :

Experimental vs Theoretical

From the measurements obtained we notice that they are pretty similar

Q3.e :



To Draw the plots i used the below python script :

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Data from the tables
5 N = [1000003 , 2000003 , 4000037 , 8000009 , 16000057 , 32000011 , 64000031 , 128000003 , 256000001 , 512000009 ,
6      1024000009 , 2048000011]
7 experimental_time = [3.643, 10.051, 12.12, 24.284, 48.268, 99.244, 191.605, 380.24, 753.837, 1517.66, 3042.77,
8                      6038.826]
9 theoretical_time = [3.03, 6.06, 12.12, 24.24, 48.48, 96.96, 193.9, 387.84, 775.837, 1551.13, 3102.72, 6205.4]
10
11 # Convert times to consistent scales
12 experimental_time = np.array(experimental_time) * 1e-3
13 theoretical_time = np.array(theoretical_time) * 1e-3
14
15 # Plot
16 plt.figure(figsize=(12, 7))
17
18 # Plot experimental time with only the first 14 N values
19 plt.plot(N, experimental_time, 'o-', label='Experimental (Time x 10-3)', color='blue')
20
21 # Plot theoretical time with all 16 N values
22 plt.plot(N, theoretical_time, 's-', label='Theoretical Best Case (Time x 10-3)', color='green')
23
24 plt.xlim(left=min(N)) # Start x-axis at the minimum N value
25 plt.ylim(bottom=min(min(experimental_time), min(theoretical_time))) # Start y-axis at the minimum time value
26
27 # Labels and title
28 plt.xlabel('N (Data Size)')
29 plt.ylabel('Time (seconds)')
30 plt.title('Experimental vs Theoretical Time')
31 plt.legend()
32 plt.grid(which="both", linestyle="--", linewidth=0.5)
33
34 # Save as PDF
35 plt.savefig('plot.pdf', format='pdf', bbox_inches='tight')
36
37 # Show the plot
38 plt.show()
```

Part 2 :

Q1 :

Algorithm Prime Number

```
1: Var
2: n, cmpt, i integer;
3: Begin
4: cmpt  $\leftarrow$  0;
5: i  $\leftarrow$  1;
6: print('Input Integer N>1 : ')
7: Read(n);
8: while i  $\leq$   $\frac{n}{2}$  do
9:   if ( n%i == 0 ) then
10:    cmpt  $\leftarrow$  cmpt + 1;
11:   end if
12:   i  $\leftarrow$  i + 1;
13: end while
14: if ( cmpt == 1 ) then
15:   print('Prime Number');
16: else
17:   print('Not A Prime Number');
18: end if
19: End
```

Q2 :

we have $f(n) = \sum fr$ since $f(n)$ is sum of frequency of execution (fr) we need to figure out the fr of each instruction and sum them :

Best Case (N is prime)

cmpt \leftarrow 0	$fr = 1$ (one affectation)
i \leftarrow 1	$fr = 1$ (one affectation)
print ('Input Integer N>1 : ')	$fr = 1$ (one print)
Read (n)	$fr = 1$ (one read)
while i \leq $\frac{n}{2}$ do	$fr = \frac{n}{2} + 1$ (check the while condition $\frac{n}{2} + 1$ times)
cmpt \leftarrow cmpt + 1	$fr = 2$ (one affectation and one arithmetic operation (2) , repeated once because N is Prime)
i \leftarrow i + 1	$fr = n$ (one affectation and one arithmetic operation (2) , inside a while that loops $\frac{n}{2}$ times (n))
print ('Prime Number')	$fr = 1$ (one print)

$$\begin{aligned}
f_3(n) &= \sum fr \\
&= 1 + 1 + 1 + 1 + (\frac{n}{2} + 1) + 2 + n + 1 \\
&= \frac{n}{2} + n + 8 \\
&= \boxed{\frac{3n}{2} + 8}
\end{aligned}$$

now that we have the complexity function $f_3(n)$ we need to find the $T_3(n)$ we have $T_3(n) = f_3(n) \times \Delta t$

$$\begin{aligned}
T_3(n) &= f_3(n) \times \Delta t \\
&= (\frac{3n}{2} + 8) \times \Delta t \\
&= \underbrace{\frac{3\Delta t}{2}}_{a_3} n + \underbrace{\Delta t 8}_{b_3} \\
&= \boxed{a_3 n + b_3}
\end{aligned}$$

Worst Case (N is not prime)

cmpt \leftarrow 0	$fr = 1$ (one affectation)
i \leftarrow 1	$fr = 1$ (one affectation)
print ('Input Integer N>1 : ')	$fr = 1$ (one print)
Read (n)	$fr = 1$ (one read)
while i \leq $\frac{n}{2}$ do	$fr = \frac{n}{2} + 1$ (check the while condition $\frac{n}{2}+1$ times)
cmpt \leftarrow cmpt + 1	$fr = n$ (one affectation and one arithmetic operation (2) , repeated at worst $\frac{n}{2}$ times (n))
i \leftarrow i + 1	$fr = n$ (one affectation and one arithmetic operation (2) , inside a while that loops $\frac{n}{2}$ times (n))
print ('Not A Prime Number')	$fr = 1$ (one print)

$$\begin{aligned}
f_4(n) &= \sum fr \\
&= 1 + 1 + 1 + 1 + (\frac{n}{2} + 1) + n + n + 1 \\
&= 2n + \frac{n}{2} + 6 \\
&= \boxed{\frac{5n}{2} + 6}
\end{aligned}$$

now that we have the complexity function $f_4(n)$ we need to find the $T_4(n)$ we have $T_4(n) = f_4(n) \times \Delta t$

$$\begin{aligned}
T_4(n) &= f_4(n) \times \Delta t \\
&= (3n + 6) \times \Delta t \\
&= \underbrace{\frac{5\Delta t}{2}}_{a_4} n + \underbrace{\Delta t 6}_{b_4} \\
&= \boxed{a_4 n + b_4}
\end{aligned}$$

Conclusion

Both $T_3(n)$ and $T_4(n)$ are linear complexity therefore they both $\sim O(n)$

Q3 :

```

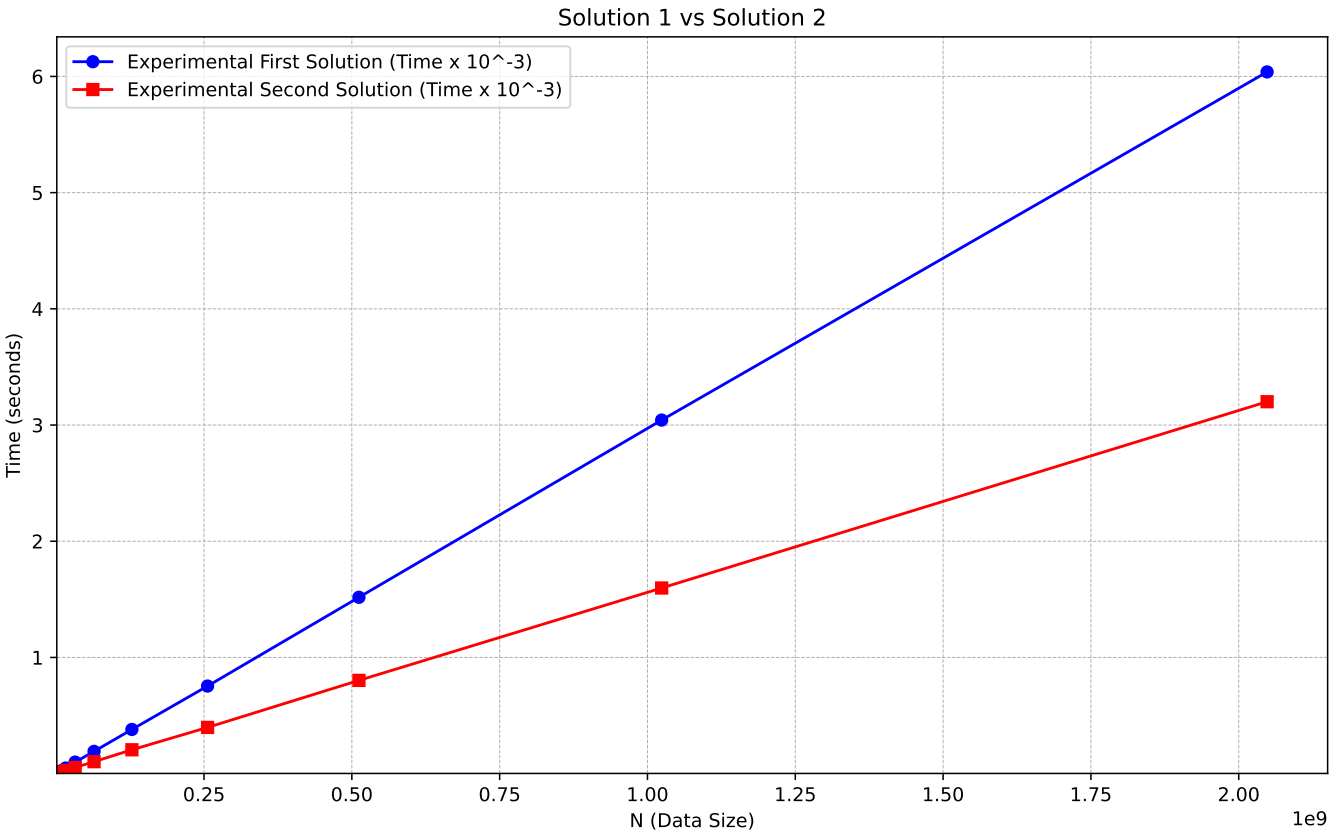
1  #include<stdio.h>
2  #include<time.h>
3
4  int main() {
5
6  int i = 1;
7  int cmpt = 0;
8  int n;
9
10 clock_t start_time = clock();
11
12 printf("Input Integer N>1 : ");
13 scanf("%d",&n);
14
15 while (i<=(n/2)) {
16     if (n%i==0) {
17         ++cmpt;
18     }
19
20     ++i;
21
22 }
23
24 if ( cmpt == 1 ) {
25     printf("\nprime number");
26 }
27 else {
28     printf("\nNot a prime number");
29 }
30
31 clock_t end_time = clock();
32 double execution_time = (double) (end_time - start_time)/CLOCKS_PER_SEC;
33 printf("\nExecution Time %f seconds\n",execution_time);
34
35 return 0;
36 }

```

Experimental

N	1000003	2000003	4000037	8000009	16000057	32000011	64000031	128000003	256000001	512000009
$T(n)$ 10^{-3}	1.676	3.247	6.842	13.009	25.376	52.545	102.985	205.319	397.85	802.53

N	1024000009	2048000011
$T(n)$ 10^{-3}	1597.518	3200.743



To Draw the plots i used the below python script :

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Data from the tables
5 N = [1000003 , 2000003 , 4000037 , 8000009 , 16000057 , 32000011 , 64000031 , 128000003 , 256000001 , 512000009 ,
6      1024000009 , 2048000011]
7 experimental_time_1 = [3.643, 10.051, 12.12, 24.284, 48.268, 99.244, 191.605, 380.24, 753.837, 1517.66, 3042.77,
8      6038.826]
9 experimental_time_2 = [1.676 , 3.247 , 6.842 , 13.009 , 25.376 , 52.545 , 102.985 , 205.319 , 397.85 , 802.53 ,
10     1597.518 , 3200.743]
11
12 # Convert times to consistent scales
13 experimental_time_1 = np.array(experimental_time_1) * 1e-3
14 experimental_time_2 = np.array(experimental_time_2) * 1e-3
15
16 plt.figure(figsize=(12, 7))
17
18 plt.plot(N, experimental_time_1, 'o--', label='Experimental First Solution (Time x 10-3)', color='blue')
19 plt.plot(N, experimental_time_2, 's--', label='Experimental Second Solution (Time x 10-3)', color='red')
20
21 plt.xlim(left=min(N)) # Start x-axis at the minimum N value
22 plt.ylim(bottom=min(min(experimental_time_1), min(experimental_time_2))) # Start y-axis at the minimum time value
23
24 # Labels and title
25 plt.xlabel('N (Data Size)')
26 plt.ylabel('Time (seconds)')
27 plt.title('Solution 1 vs Solution 2')
28 plt.legend()
29 plt.grid(which="both", linestyle="--", linewidth=0.5)
30
31 # Save as PDF
32 plt.savefig('plot.pdf', format='pdf', bbox_inches='tight')
33
34 # Show the plot
35 plt.show()
```

Observation

We notice that the second solution takes about half time of the first solution therefore the second solution is more efficient

Part 3 :

Q1 :

Algorithm Prime Number

```
1: Var
2: n, cmpt, i integer;
3: Begin
4: cmpt  $\leftarrow$  0;
5: i  $\leftarrow$  1;
6: print('Input Integer N>1 : ')
7: Read(n);
8: while i  $\leq$   $\sqrt{n}$  do
9:   if ( n%i == 0 ) then
10:    cmpt  $\leftarrow$  cmpt + 1;
11:   end if
12:   i  $\leftarrow$  i + 1;
13: end while
14: if ( cmpt == 1 ) then
15:   print('Prime Number');
16: else
17:   print('Not A Prime Number');
18: end if
19: End
```

Q2 :

we have $f(n) = \sum fr$ since $f(n)$ is sum of frequency of execution (fr) we need to figure out the fr of each instruction and sum them :

Best Case (N is prime)

cmpt \leftarrow 0	$fr = 1$ (one affectation)
i \leftarrow 1	$fr = 1$ (one affectation)
print ('Input Integer N>1 : ')	$fr = 1$ (one print)
Read (n)	$fr = 1$ (one read)
while i \leq \sqrt{n} do	$fr = \sqrt{n} + 1$ (check the while condition $\sqrt{n} + 1$ times)
cmpt \leftarrow cmpt + 1	$fr = 2$ (one affectation and one arithmetic operation (2) , repeated once because N is Prime)
i \leftarrow i + 1	$fr = 2 \times \sqrt{n}$ (one affectation and one arithmetic operation (2) , inside a while that loops \sqrt{n})
print ('Prime Number')	$fr = 1$ (one print)

$$\begin{aligned}
f_5(n) &= \sum fr \\
&= 1 + 1 + 1 + 1 + (\sqrt{n} + 1) + 2 + 2 \times \sqrt{n} + 1 \\
&= 3 \times \sqrt{n} + 8 \\
&= \boxed{3 \times \sqrt{n} + 8}
\end{aligned}$$

now that we have the complexity function $f_5(n)$ we need to find the $T_5(n)$ we have $T_5(n) = f_5(n) \times \Delta t$

$$\begin{aligned}
T_5(n) &= f_5(n) \times \Delta t \\
&= (3 \times \sqrt{n} + 8) \times \Delta t \\
&= \underbrace{3\Delta t}_{a_5} \sqrt{n} + \underbrace{\Delta t 8}_{b_5} \\
&= \boxed{a_5 \sqrt{n} + b_5}
\end{aligned}$$

Worst Case (N is not prime)

cmpt \leftarrow 0	$fr = 1$ (one affectation)
i \leftarrow 1	$fr = 1$ (one affectation)
print('Input Integer N>1 : ')	$fr = 1$ (one print)
Read(n)	$fr = 1$ (one read)
while i $\leq \sqrt{n}$ do	$fr = \sqrt{n} + 1$ (check the while condition $\sqrt{n} + 1$ times)
cmpt \leftarrow cmpt + 1	$fr = 2 \times \sqrt{n}$ (one affectation and one arithmetic operation (2) , repeated at worst \sqrt{n})
i \leftarrow i + 1	$fr = 2 \times \sqrt{n}$ (one affectation and one arithmetic operation (2) , inside a while that loops \sqrt{n})
print('Prime Number')	$fr = 1$ (one print)

$$\begin{aligned}
f_6(n) &= \sum fr \\
&= 1 + 1 + 1 + 1 + (\sqrt{n} + 1) + 2\sqrt{n} + 2\sqrt{n} + 1 \\
&= 5\sqrt{n} + 6 \\
&= \boxed{5\sqrt{n} + 6}
\end{aligned}$$

now that we have the complexity function $f_6(n)$ we need to find the $T_6(n)$ we have $T_6(n) = f_6(n) \times \Delta t$

$$\begin{aligned}
T_4(n) &= f_4(n) \times \Delta t \\
&= (5\sqrt{n} + 6) \times \Delta t \\
&= \underbrace{5\Delta t}_{a_6} \sqrt{n} + \underbrace{\Delta t 6}_{b_6} \\
&= \boxed{a_6\sqrt{n} + b_6}
\end{aligned}$$

Conclusion

Both $T_5(n)$ and $T_6(n)$ are square root complexity therefore they both $\sim O(\sqrt{n})$

Q3 :

```

1  #include<stdio.h>
2  #include<time.h>
3  #include<math.h>
4
5  int main() {
6
7  int i = 1;
8  int cmpt = 0;
9  int n;
10
11  clock_t start_time = clock();
12
13  printf("Input Integer N>1 : ");
14  scanf("%d",&n);
15
16  double limit = sqrt(n);
17
18  while (i<=limit) {
19      if (n%i==0) {
20          ++cmpt;
21      }
22
23      ++i;
24
25  }
26
27  if ( cmpt == 1 ) {
28      printf("\nprime number");
29  }
30  else {
31      printf("\nNot a prime number");
32  }
33
34  clock_t end_time = clock();
35  double execution_time = (double) (end_time - start_time)/CLOCKS_PER_SEC;
36  printf("\nExecution Time %f seconds\n",execution_time);

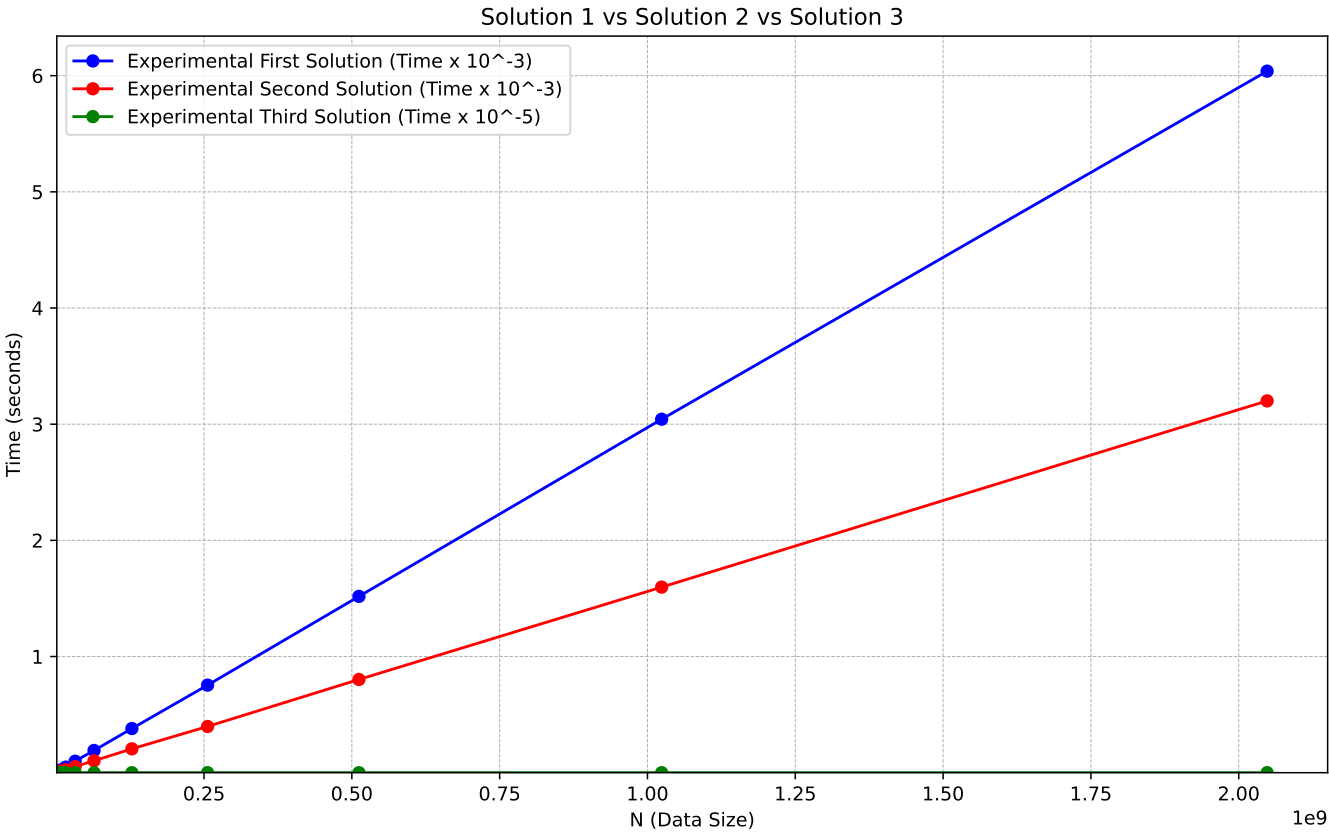
```

```
37
38 return 0;
39 }
```

Experimental

N	1000003	2000003	4000037	8000009	16000057	32000011	64000031	128000003	256000001	512000009
$T(n)$ 10^{-5}	7.5	7.6	7.7	8.2	8.8	9.2	11.4	12	14.6	17

N	1024000009	2048000011
$T(n)$ 10^{-5}	25.3	26.5

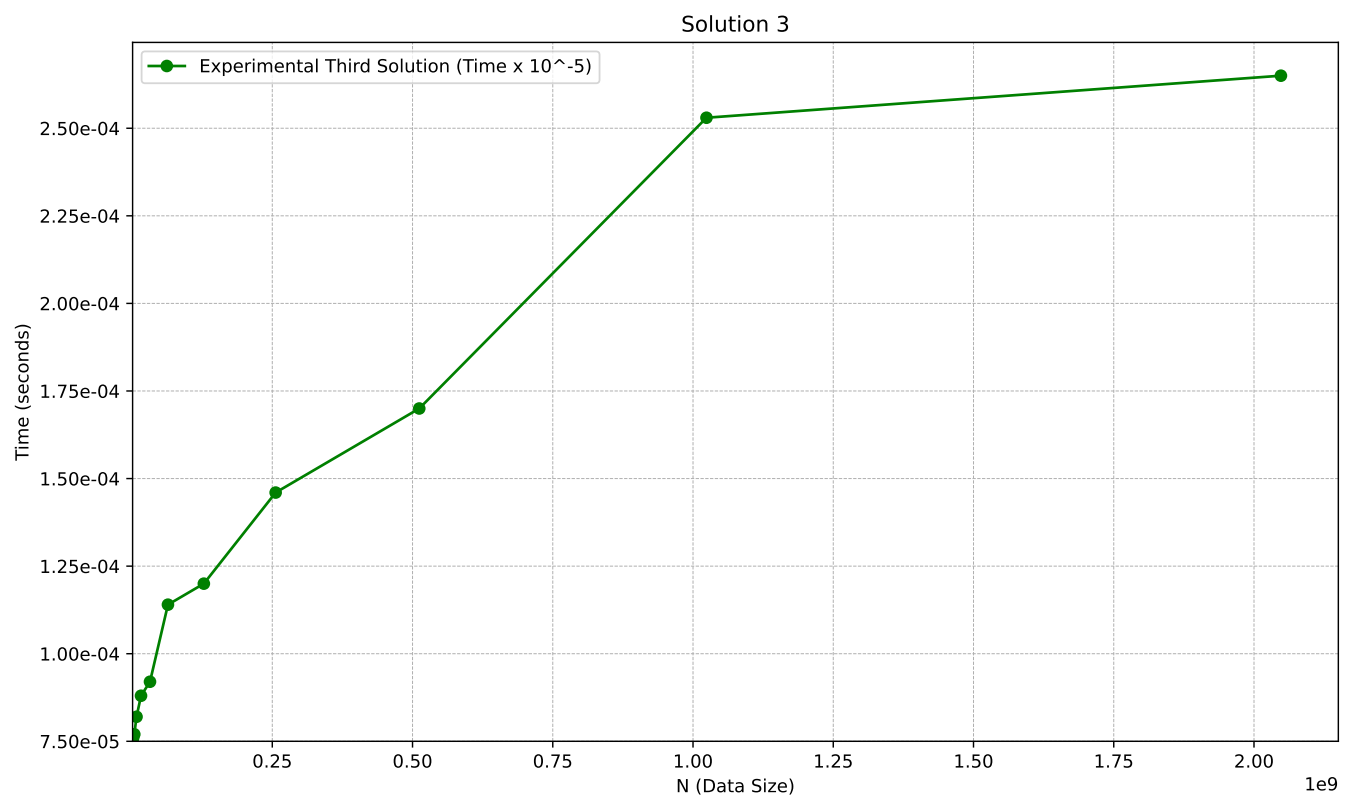


To Draw the plots i used the below python script :

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Data from the tables
5 N = [1000003 , 2000003 , 4000037 , 8000009 , 16000057 , 32000011 , 64000031 , 128000003 , 256000001 , 512000009 ,
6       1024000009 , 2048000011]
7 experimental_time_1 = [3.643, 10.051, 12.12, 24.284, 48.268, 99.244, 191.605, 380.24, 753.837, 1517.66, 3042.77,
8       6038.826]
9 experimental_time_2 = [1.676 , 3.247 , 6.842 , 13.009 , 25.376 , 52.545 , 102.985 , 205.319 , 397.85 , 802.53 ,
10      1597.518 , 3200.743]
11 experimental_time_3 = [ 7.5 , 7.6 , 7.7 , 8.2 , 8.8 , 9.2 , 11.4 , 12 , 14.6 , 17 , 25.3 , 26.5]
12
13 # Convert times to consistent scales
14 experimental_time_1 = np.array(experimental_time_1) * 1e-3
15 experimental_time_2 = np.array(experimental_time_2) * 1e-3
16 experimental_time_3 = np.array(experimental_time_3) * 1e-5
17
18 plt.figure(figsize=(12, 7))
19
20 plt.plot(N, experimental_time_1, 'o-', label='Experimental First Solution (Time x 10-3)', color='blue')
21 plt.plot(N, experimental_time_2, 'o-', label='Experimental Second Solution (Time x 10-3)', color='red')
22 plt.plot(N, experimental_time_3, 'o-', label='Experimental Third Solution (Time x 10-5)', color='green')
23
24 plt.xlim(left=min(N)) # Start x-axis at the minimum N value
25 plt.ylim(bottom=min(min(experimental_time_1), min(experimental_time_2) , min(experimental_time_3))) # Start y-axis
26 # at the minimum time value
27
28 # Labels and title
29 plt.xlabel('N (Data Size)')
30 plt.ylabel('Time (seconds)')
31 plt.title('Solution 1 vs Solution 2 vs Solution 3')
32 plt.legend()
33 plt.grid(which="both", linestyle="--", linewidth=0.5)
34
35 # Save as PDF
36 plt.savefig('plot.pdf', format='pdf', bbox_inches='tight')
37
38 # Show the plot
39 plt.show()
```

Observation

We notice in the plots that the green plot (the 3rd solution plot) is barely visible due to its y range of values being much smaller than the red and blue (solution 1 and 2) , so the third solution is the most efficient of them



To Draw the plot i used the below python script :

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 from matplotlib.ticker import FuncFormatter
4
5 # Data from the tables
6 N = [1000003 , 2000003 , 4000037 , 8000009 , 16000057 , 32000011 , 64000031 , 128000003 , 256000001 , 512000009 ,
7      1024000009 , 2048000011]
8 experimental_time_3 = [ 7.5 , 7.6 , 7.7 , 8.2 , 8.8 , 9.2 , 11.4 , 12 , 14.6 , 17 , 25.3 , 26.5]
9
10 # Convert times to consistent scales
11 experimental_time_3 = np.array(experimental_time_3) * 1e-5
12
13 plt.figure(figsize=(12, 7))
14
15 plt.plot(N, experimental_time_3, 'o--', label='Experimental Third Solution (Time x 10-5)', color='green')
16
17 plt.xlim(left=min(N)) # Start x-axis at the minimum N value
18 plt.ylim(bottom= min(experimental_time_3)) # Start y-axis at the minimum time value
19
20 # Labels and title
21 plt.xlabel('N (Data Size)')
22 plt.ylabel('Time (seconds)')
23 plt.title('Solution 3')
24 plt.legend()
25 plt.grid(which="both", linestyle="--", linewidth=0.5)
26
27 # Use FuncFormatter to enforce scientific notation
28 formatter = FuncFormatter(lambda x, pos: '{:.2e}'.format(x)) # '{:.2e}' enforces scientific notation
29
30 # Apply the formatter to the y-axis
31 plt.gca().yaxis.set_major_formatter(formatter)
32
33 # Save as PDF
34 plt.savefig('sqrt.pdf', format='pdf', bbox_inches='tight')
35
36 # Show the plot
37 plt.show()
```

Part 4 :

Q1 :

Algorithm Prime Number

```
1: Var
2: n, cmpt, i ,step integer;
3: Begin
4: cmpt  $\leftarrow$  0;
5: i  $\leftarrow$  1;
6: step  $\leftarrow$  1;

7: print('Input Integer N>1 : ')
8: Read(n);

9: if ( n%2 !=0 ) then
10:     step  $\leftarrow$  2;
11: end if
12: while i  $\leq$   $\sqrt{n}$  do
13:     if (n%i == 0 ) then
14:         cmpt  $\leftarrow$  cmpt + 1;
15:     end if
16:     i  $\leftarrow$  i + step;
17: end while

18: if ( cmpt == 1 ) then
19:     print('Prime Number');
20: else
21:     print('Not A Prime Number');
22: end if
23: End
```

Q2 :

we have $f(n) = \sum fr$ since $f(n)$ is sum of frequency of execution (fr) we need to figure out the fr of each instruction and sum them :

Best Case (N is prime & Odd)

cmpt \leftarrow 0	$fr = 1$ (one affectation)
i \leftarrow 1	$fr = 1$ (one affectation)
step \leftarrow 1	$fr = 1$ (one affectation)
print ('Input Integer N>1 : ')	$fr = 1$ (one print)
Read (n)	$fr = 1$ (one read)
step \leftarrow 2	$fr = 1$ (one affectation)
while i \leq \sqrt{n} do	$fr = \frac{\sqrt{n}}{2} + 1$ (check the while condition $\frac{\sqrt{n}}{2} + 1$ times because step = 2)
cmpt \leftarrow cmpt + 1	$fr = 2$ (one affectation and one arithmetic operation (2) , repeated once because N is Prime)
i \leftarrow i + step	$fr = \sqrt{n}$ (one affectation and one arithmetic operation (2) , inside a while that loops $\frac{\sqrt{n}}{2}$)
print ('Prime Number')	$fr = 1$ (one print)

$$\begin{aligned}
f_7(n) &= \sum fr \\
&= 1 + 1 + 1 + 1 + 1 + 1 + 1 + \left(\frac{\sqrt{n}}{2} + 1\right) + 2 + \sqrt{n} + 1 \\
&= \frac{\sqrt{n}}{2} + \sqrt{n} + 10 \\
&= \boxed{\frac{3\sqrt{n}}{2} + 10}
\end{aligned}$$

now that we have the complexity function $f_7(n)$ we need to find the $T_7(n)$ we have $T_7(n) = f_7(n) \times \Delta t$

$$\begin{aligned}
T_7(n) &= f_7(n) \times \Delta t \\
&= \left(\frac{3\sqrt{n}}{2} + 10\right) \times \Delta t \\
&= \underbrace{\frac{3\Delta t}{2}}_{a_7} \sqrt{n} + \underbrace{\Delta t 10}_{b_7} \\
&= \boxed{a_7 \sqrt{n} + b_7}
\end{aligned}$$

Worst Case (N is not prime & even)

cmpt \leftarrow 0	$fr = 1$ (one affectation)
i \leftarrow 1	$fr = 1$ (one affectation)
step \leftarrow 1	$fr = 1$ (one affectation)
<code>print('Input Integer N>1 : ')</code>	$fr = 1$ (one print)
<code>Read(n)</code>	$fr = 1$ (one read)
while i $\leq \sqrt{n}$ do	$fr = \sqrt{n} + 1$ (check the while condition $\sqrt{n} + 1$ times)
cmpt \leftarrow cmpt + 1	$fr = 2 \times \sqrt{n}$ (one affectation and one arithmetic operation (2) , repeated at worst \sqrt{n})
i \leftarrow i + step	$fr = 2\sqrt{n}$ (one affectation and one arithmetic operation (2) , inside a while that loops \sqrt{n})
<code>print('Prime Number')</code>	$fr = 1$ (one print)

$$\begin{aligned}
f_8(n) &= \sum fr \\
&= 1 + 1 + 1 + 1 + 1 + 1 + (\sqrt{n} + 1) + 2\sqrt{n} + 2\sqrt{n} + 1 \\
&= 5\sqrt{n} + 7 \\
&= \boxed{5\sqrt{n} + 7}
\end{aligned}$$

now that we have the complexity function $f_8(n)$ we need to find the $T_8(n)$ we have $T_8(n) = f_8(n) \times \Delta t$

$$\begin{aligned}
 T_8(n) &= f_8(n) \times \Delta t \\
 &= (5\sqrt{n} + 7) \times \Delta t \\
 &= \underbrace{5\Delta t}_{a_8} \sqrt{n} + \underbrace{\Delta t 7}_{b_8} \\
 &= \boxed{a_8 \sqrt{n} + b_8}
 \end{aligned}$$

Conclusion

Both $T_8(n)$ and $T_8(n)$ are square root complexity therefore they both $\sim O(\sqrt{n})$

Q3 :

```

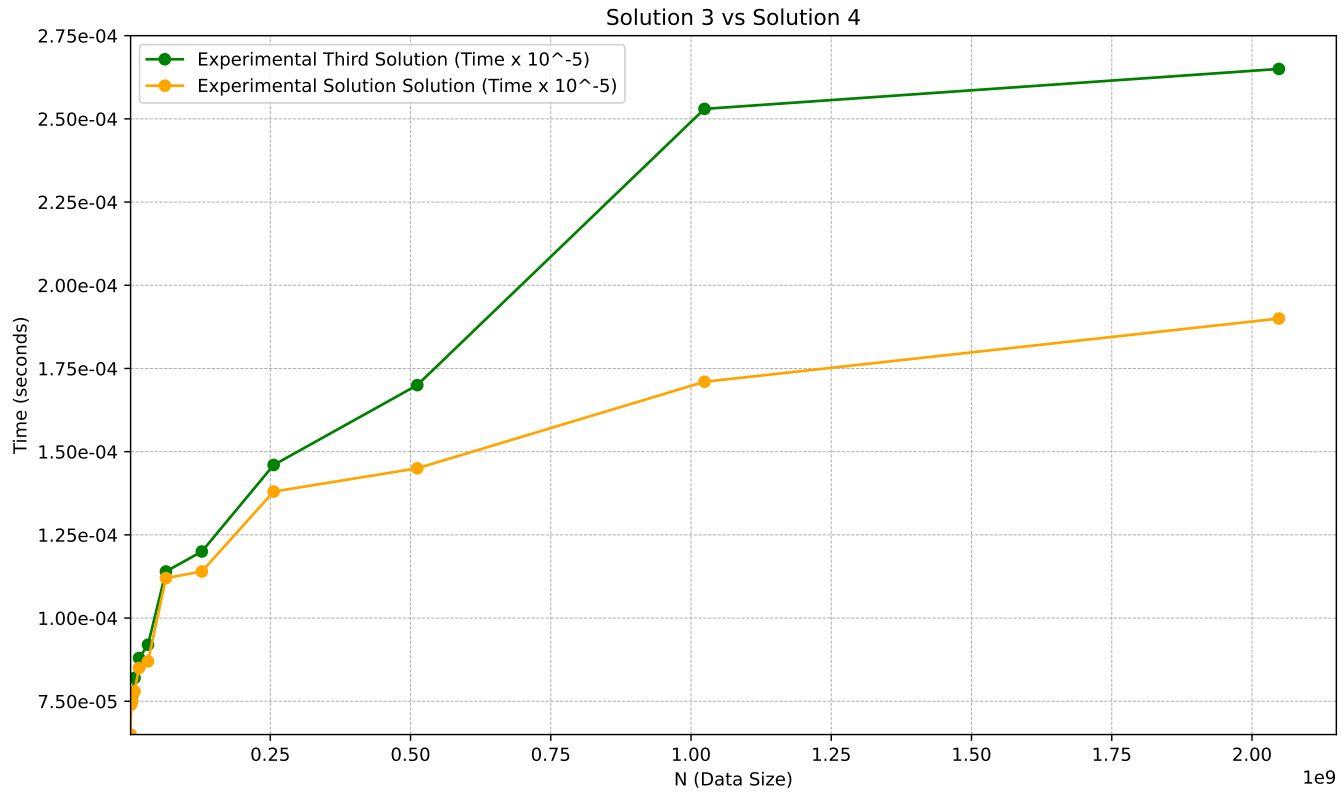
1  #include<stdio.h>
2  #include<time.h>
3  #include<math.h>
4
5  int main() {
6
7      int i = 1;
8      int cmpt = 0;
9      int step = 1;
10     int n;
11
12     clock_t start_time = clock();
13
14     printf("Input Integer N>1 : ");
15     scanf("%d",&n);
16
17     double limit = sqrt(n);
18
19     if(n%2 != 0){
20         step = 2;
21     }
22
23     while (i<=limit) {
24         if (n%i==0) {++cmpt;}
25         i = i + step;
26     }
27
28     if ( cmpt == 1 ) {printf("\nprime number");}
29     else {printf("\nNot a prime number");}
30
31     clock_t end_time = clock();
32     double execution_time = (double) (end_time - start_time)/CLOCKS_PER_SEC;
33     printf("\nExecution Time %f seconds\n",execution_time);
34
35     return 0;
36 }

```

Experimental

N	1000003	2000003	4000037	8000009	16000057	32000011	64000031	128000003	256000001	512000009
$T(n)$ 10^{-5}	6.5	7.4	7.5	7.8	8.5	8.7	11.2	11.4	13.8	14.5

N	1024000009	2048000011
$T(n)$ 10^{-5}	17.1	19



To Draw the plots i used the below python script :

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 from matplotlib.ticker import FuncFormatter
4
5 # Data from the tables
6 N = [1000003 , 2000003 , 4000037 , 8000009 , 16000057 , 32000011 , 64000031 , 128000003 , 256000001 , 512000009 ,
7      1024000009 , 2048000011]
8 experimental_time_3 = [ 7.5 , 7.6 , 7.7 , 8.2 , 8.8 , 9.2 , 11.4 , 12 , 14.6 , 17 , 25.3 , 26.5]
9 experimental_time_4 = [ 6.5 , 7.4 , 7.5 , 7.8 , 8.5 , 8.7 , 11.2 , 11.4 , 13.8 , 14.5 , 17.1 , 19]
10
11 # Convert times to consistent scales
12 experimental_time_3 = np.array(experimental_time_3) * 1e-5
13 experimental_time_4 = np.array(experimental_time_4) * 1e-5
14
15 plt.figure(figsize=(12, 7))
16
17 plt.plot(N, experimental_time_3, 'o-', label='Experimental Third Solution (Time x 10-5)', color='green')
18 plt.plot(N, experimental_time_4, 'o-', label='Experimental Solution Solution (Time x 10-5)', color='orange')
19
20 plt.xlim(left=min(N)) # Start x-axis at the minimum N value
21 plt.ylim(bottom= min(min(experimental_time_3), min(experimental_time_4))) # Start y-axis at the minimum time value
22
23 # Labels and title
24 plt.xlabel('N (Data Size)')
25 plt.ylabel('Time (seconds)')
26 plt.title('Solution 3 vs Solution 4')
27 plt.legend()
28 plt.grid(which="both", linestyle="--", linewidth=0.5)
29
30 # Use FuncFormatter to enforce scientific notation
31 formatter = FuncFormatter(lambda x, pos: '{:.2e}'.format(x)) # '{:.2e}' enforces scientific notation
32
33 # Apply the formatter to the y-axis
34 plt.gca().yaxis.set_major_formatter(formatter)
35
36 # Save as PDF
37 plt.savefig('plot.pdf', format='pdf', bbox_inches='tight')
38
39 # Show the plot
40 plt.show()
```

Observation

From the plots we notice that the 4th solution takes about the same time as 3rd solution but as n grows bigger the 4th solution takes about half time of the 3rd therefore the 4th solution is the most efficient out of all the solutions