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

LAB2 : Primality Test

Part 1 :

Q1 :

Algorithm Prime Number

```
Var
n, cmpt, i integer;
Begin
cmpt ← 0;
i ← 1;
print('Input Integer N>1 : ')
Read(n);
while i <= n do
    if ( n%i == 0 ) then
        cmpt ← cmpt + 1;
    end if
    i ← i + 1;
end while
if ( cmpt == 2 ) then
    print('Prime Number');
else
    print('Not A Prime Number');
end if
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 ← 0	$fr = 1$ (one affectation)
i ← 1	$fr = 1$ (one affectation)
print('Input Integer N>1 : ')	$fr = 1$ (one print)
Read(n)	$fr = 1$ (one read)

while i <= n **do** — $fr = n + 1$ (check the while condition n+1 times)
cmpt ← cmpt + 1 — $fr = 4$ (one affectation and one arithmetic operation (2) , repeated twice because N is Prime)
i ← i + 1 — $fr = 2n$ (one affectation and one arithmetic operation (2) , inside a while that loops n times (2n))
print('Prime Number') — $fr = 1$ (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 — $fr = 1$ (one affectation)
i ← 1 — $fr = 1$ (one affectation)
print('Input Integer N>1 : ') — $fr = 1$ (one print)
Read(n) — $fr = 1$ (one read)
while i <= n **do** — $fr = n + 1$ (check the while condition n+1 times)
cmpt ← cmpt + 1 — $fr = n$ (one affectation and one arithmetic operation (2) , repeated at worst $\frac{n}{2}$ time)
i ← i + 1 — $fr = 2n$ (one affectation and one arithmetic operation (2) , inside a while that loops n times (2n))
print('Not A Prime Number') — $fr = 1$ (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 :

```c
#include<stdio.h>
#include<time.h>

int main() {

int i = 1;
int cmpt = 0;
int n;


clock_t start_time = clock();

printf("Input Integer N>1 : ");
scanf("%d",&n);

while (i<=n) {
 if ( n%i==0 ) {++cmpt;}
 ++i;
}

if ( cmpt == 2 ) {printf("\nprime number");}

 else {printf("\nNot a prime number");}

clock_t end_time = clock();

double execution_time = (double) (end_time - start_time)/CLOCKS_PER_SEC;

printf("\nExecution Time %f seconds\n",execution_time);


return 0;

}
```

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

Theoritical

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) = 24.284×10^{-3} :

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

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

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

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

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

Theoritical 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

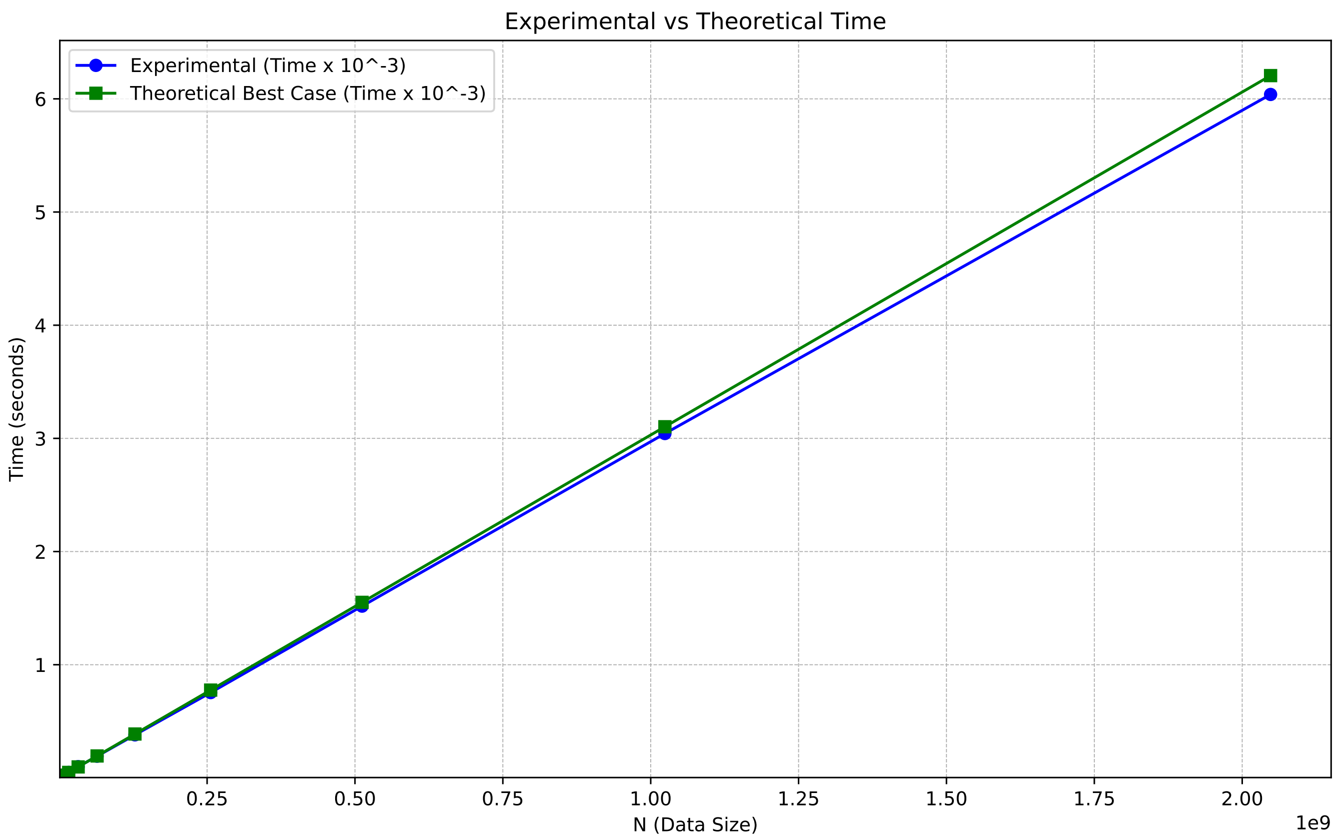
We notice that from the data and measurements obtained that the growth of the time complexity is linear $\sim O(n)$

Q3.d :

Experimental vs Theoritical

From the measurments obtained we notice that the they are pretty similar

Q3.e :



To Draw the plots i used the below python script :

```python
import matplotlib.pyplot as plt
import numpy as np

# Data from the tables
N = [1000003 , 2000003 , 4000037 , 8000009 , 16000057 , 32000011 , 64000031 , 128000003 , 256000001 , 512000009 , 1024000009 , 2048000011]
experimental_time = [3.643, 10.051, 12.12, 24.284, 48.268, 99.244, 191.605, 380.24, 753.837, 1517.66, 3042.77, 6038.826]
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]

# Convert times to consistent scales
experimental_time = np.array(experimental_time) * 1e-3
theoretical_time = np.array(theoretical_time) * 1e-3

# Plot
plt.figure(figsize=(12, 7))

# Plot experimental time with only the first 14 N values
plt.plot(N, experimental_time, 'o-', label='Experimental (Time x 10^-3)', color='blue')

# Plot theoretical time with all 16 N values
plt.plot(N, theoretical_time, 's-', label='Theoretical Best Case (Time x 10^-3)', color='green')

plt.xlim(left=min(N))  # Start x-axis at the minimum N value
plt.ylim(bottom=min(min(experimental_time), min(theoretical_time)))  # Start y-axis at the minimum time value


# Labels and title
plt.xlabel('N (Data Size)')
plt.ylabel('Time (seconds)')
plt.title('Experimental vs Theoretical Time')
plt.legend()
plt.grid(which="both", linestyle="--", linewidth=0.5)

# Save as PDF
plt.savefig('plot.pdf', format='pdf', bbox_inches='tight')

# Show the plot
plt.show()
```

Part 2 :

Q1 :

Algorithm Prime Number

```
Var
n, cmpt, i integer;
Begin
cmpt ← 0;
i ← 1;
print('Input Integer N>1 : ')
Read(n);
while i <= n/2 do
    if ( n%i == 0 ) then
        cmpt ← cmpt + 1;
    end if
    i ← i + 1;
end while
if ( cmpt == 1 ) then
    print('Prime Number');
else
    print('Not A Prime Number');
end if
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 ← 0	$fr = 1$ (one affectation)
i ← 1	$fr = 1$ (one affectation)
print('Input Integer N>1 : ')	$fr = 1$ (one print)
Read(n)	$fr = 1$ (one read)
while i $<=$ $\frac{n}{2}$ **do**	$fr = \frac{n}{2} + 1$ (check the while condition $\frac{n}{2} + 1$ times)
cmpt ← cmpt + 1	$fr = 2$ (one affectation and one arithmetic operation (2) , repeated once because N is Prime)
i ← 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 $T_3(n) = f_3(n) \times \Delta t$

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

<u>Worst Case (N is not prime)</u>

cmpt ← 0	$fr = 1$ (one affectation)
i ← 1	$fr = 1$ (one affectation)
print('Input Integer N>1 : ')	$fr = 1$ (one print)
Read(n)	$fr = 1$ (one read)
while i <= $\frac{n}{2}$ **do**	$fr = \frac{n}{2}+1$ (check the while condition $\frac{n}{2}$+1 times)
cmpt ← cmpt + 1	$fr = n$ (one affectation and one arithmetic operation (2) , repeated at worst $\frac{n}{2}$ times (n))
i ← 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 $T_4(n) = f_4(n) \times \Delta t$

$$
\begin{aligned}
T_4(n) &= f_4(n) \times \Delta t \\
&= (3n + 6) \times \Delta t \\
&= \underbrace{\frac{5\Delta t}{2}}_{a_4} n + \underbrace{\Delta t6}_{b_4} \\
&= \boxed{a_4 n + b_4}
\end{aligned}
$$

Conclusion

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

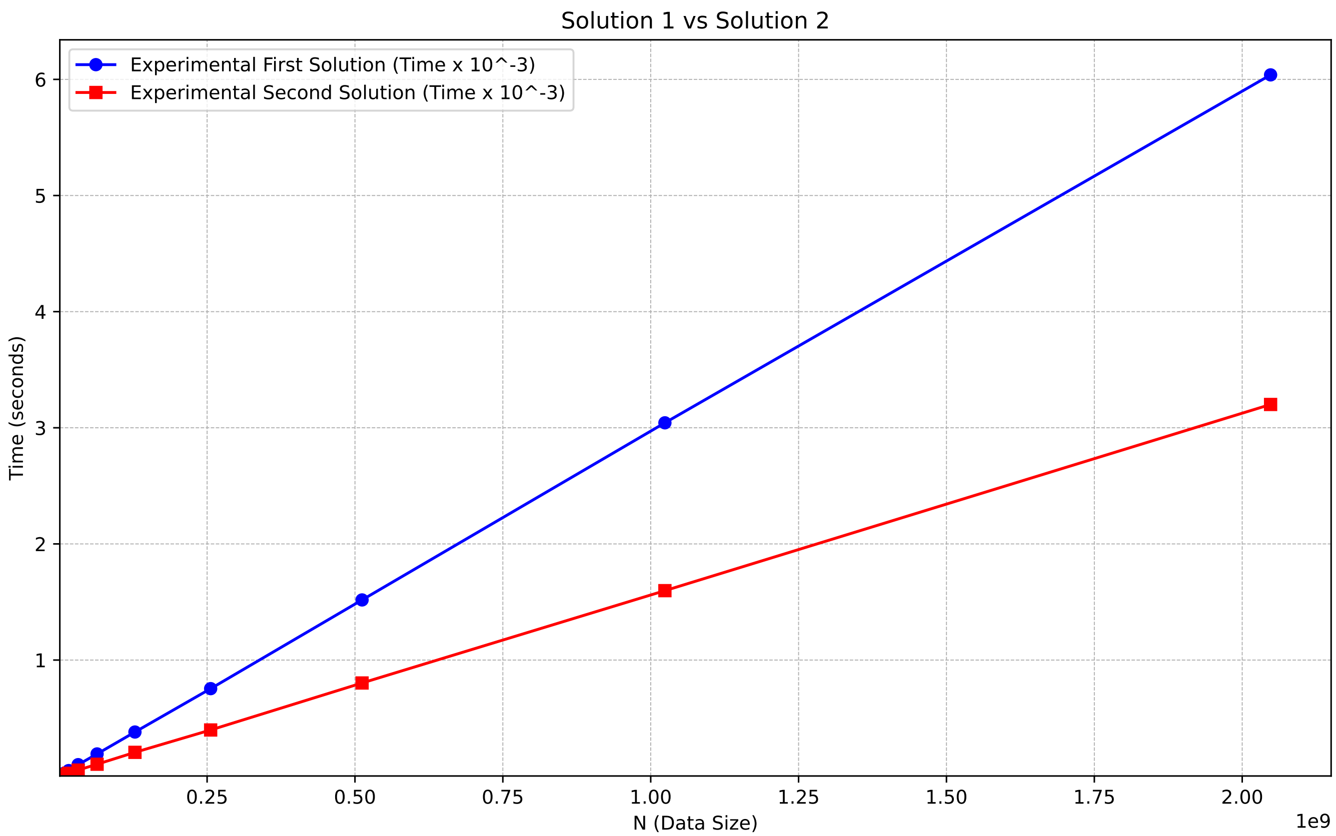
Q3 :

```c
#include<stdio.h>
#include<time.h>

int main() {

int i = 1;
int cmpt = 0;
int n;

clock_t start_time = clock();

printf("Input Integer N>1 : ");
scanf("%d",&n);

while (i<=(n/2)) {
 if (n%i==0) {
   ++cmpt;
 }

 ++i;

}

if ( cmpt == 1 ) {
printf("\nprime number");
}
else {
printf("\nNot a prime number");
}

clock_t end_time = clock();
double execution_time = (double) (end_time - start_time)/CLOCKS_PER_SEC;
printf("\nExecution Time %f seconds\n",execution_time);

return 0;
}
```

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 :

```python
import matplotlib.pyplot as plt
import numpy as np

# Data from the tables
N = [1000003 , 2000003 , 4000037 , 8000009 , 16000057 , 32000011 , 64000031 , 128000003 , 256000001 , 512000009 ,
    1024000009 , 2048000011]
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,
    6038.826]
experimental_time_2 = [1.676 , 3.247 , 6.842 , 13.009 , 25.376 , 52.545 , 102.985 , 205.319 , 397.85 , 802.53 ,
    1597.518 , 3200.743]

# Convert times to consistent scales
experimental_time_1 = np.array(experimental_time_1) * 1e-3
experimental_time_2 = np.array(experimental_time_2) * 1e-3

plt.figure(figsize=(12, 7))

plt.plot(N, experimental_time_1, 'o-', label='Experimental First Solution (Time x 10^-3)', color='blue')

plt.plot(N, experimental_time_2, 's-', label='Experimental Second Solution (Time x 10^-3)', color='red')

plt.xlim(left=min(N))  # Start x-axis at the minimum N value
plt.ylim(bottom=min(min(experimental_time_1), min(experimental_time_2)))  # Start y-axis at the minimum time value


# Labels and title
plt.xlabel('N (Data Size)')
plt.ylabel('Time (seconds)')
plt.title('Solution 1 vs Solution 2')
plt.legend()
plt.grid(which="both", linestyle="--", linewidth=0.5)

# Save as PDF
plt.savefig('plot.pdf', format='pdf', bbox_inches='tight')

# Show the plot
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 :

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 :

Algorithm Prime Number

```
Var
n, cmpt, i integer;
Begin
cmpt ← 0;
i ← 1;
print('Input Integer N>1 : ')
Read(n);
while i <= √n do
    if ( n%i == 0 ) then
        cmpt ← cmpt + 1;
    end if
    i ← i + 1;
end while
if ( cmpt == 1 ) then
    print('Prime Number');
else
    print('Not A Prime Number');
end if
End
```

Best Case (N is prime)

cmpt ← 0	$fr = 1$ (one affectation)
i ← 1	$fr = 1$ (one affectation)
print('Input Integer N>1 : ')	$fr = 1$ (one print)
Read(n)	$fr = 1$ (one read)
while i <= $\sqrt{n}$ **do**	$fr = \sqrt{n} + 1$ (check the while condition $\sqrt{n} + 1$ times)
cmpt ← cmpt + 1	$fr = 2$ (one affectation and one arithmetic operation (2) , repeated once because N is Prime)
i ← 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 ← 0	$fr = 1$ (one affectation)
i ← 1	$fr = 1$ (one affectation)
print('Input Integer N>1 : ')	$fr = 1$ (one print)
Read(n)	$fr = 1$ (one read)
while i <= $\sqrt{n}$ **do**	$fr = \sqrt{n} + 1$ (check the while condition $\sqrt{n} + 1$ times)
cmpt ← cmpt + 1	$fr = 2 \times \sqrt{n}$ (one affectation and one arithmetic operation (2) , repeated at worst $\sqrt{n}$)
i ← 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 $f_4(n)$ 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 $\sim O(\sqrt{n})$

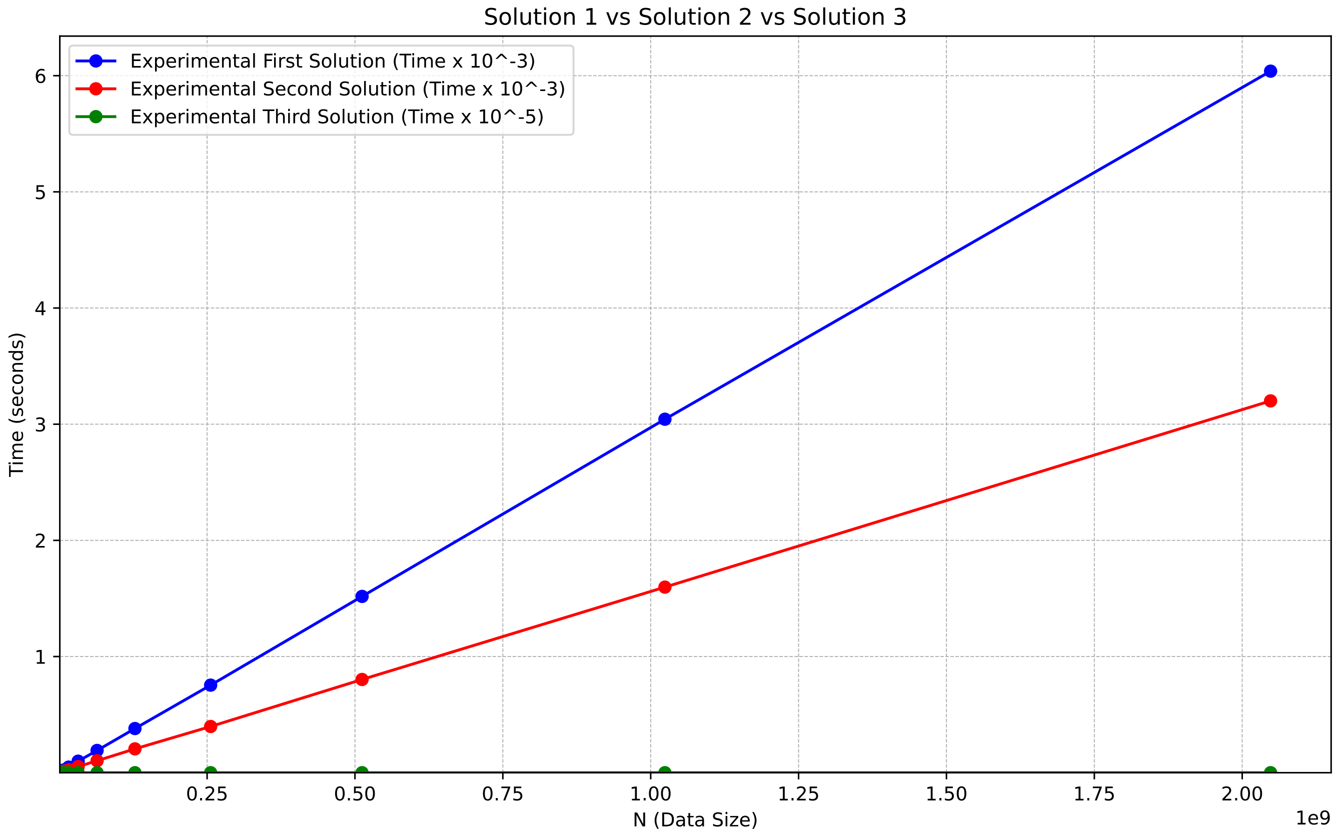
Q3 :

```c
#include<stdio.h>
#include<time.h>
#include<math.h>

int main() {

int i = 1;
int cmpt = 0;
int n;

clock_t start_time = clock();

printf("Input Integer N>1 : ");
scanf("%d",&n);

double limit = sqrt(n);

while (i<=limit) {
 if (n%i==0) {
   ++cmpt;
 }

 ++i;

}

if ( cmpt == 1 ) {
printf("\nprime number");
}
else {
printf("\nNot a prime number");
}

clock_t end_time = clock();
double execution_time = (double) (end_time - start_time)/CLOCKS_PER_SEC;
printf("\nExecution Time %f seconds\n",execution_time);

return 0;
}
```

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

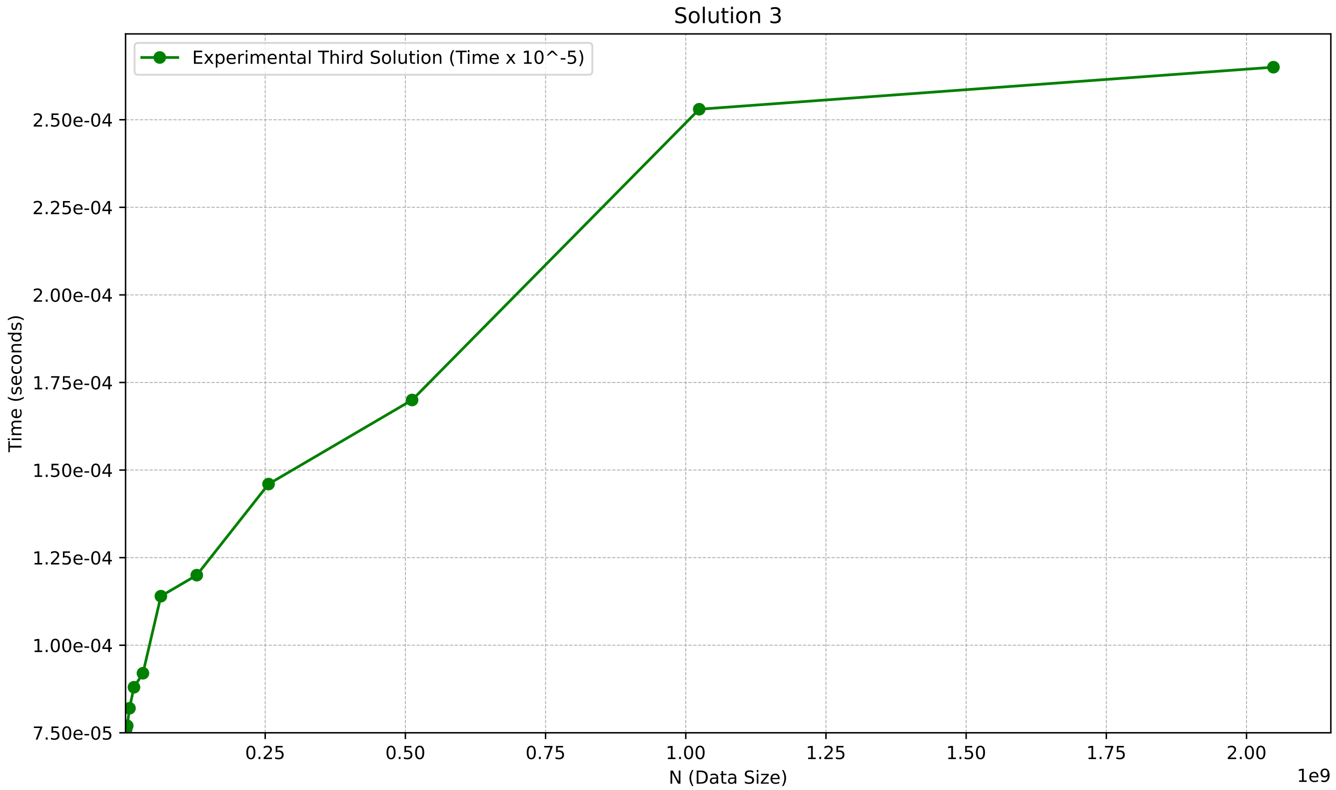

Solution 1 vs Solution 2 vs Solution 3
Experimental First Solution (Time x 10^-3)
Experimental Second Solution (Time x 10^-3)
Experimental Third Solution (Time x 10^-5)
Time (seconds)
N (Data Size)
1e9
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1
2
3
4
5
6

To Draw the plots i used the below python script :

```python
import matplotlib.pyplot as plt
import numpy as np

# Data from the tables
N = [1000003 , 2000003 , 4000037 , 8000009 , 16000057 , 32000011 , 64000031 , 128000003 , 256000001 , 512000009 ,
    1024000009 , 2048000011]
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,
    6038.826]
experimental_time_2 = [1.676 , 3.247 , 6.842 , 13.009 , 25.376 , 52.545 , 102.985 , 205.319 , 397.85 , 802.53 ,
    1597.518 , 3200.743]
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]

# Convert times to consistent scales
experimental_time_1 = np.array(experimental_time_1) * 1e-3
experimental_time_2 = np.array(experimental_time_2) * 1e-3
experimental_time_3 = np.array(experimental_time_3) * 1e-5

plt.figure(figsize=(12, 7))

plt.plot(N, experimental_time_1, 'o-', label='Experimental First Solution (Time x 10^-3)', color='blue')
plt.plot(N, experimental_time_2, 'o-', label='Experimental Second Solution (Time x 10^-3)', color='red')
plt.plot(N, experimental_time_3, 'o-', label='Experimental Third Solution (Time x 10^-5)', color='green')

plt.xlim(left=min(N))  # Start x-axis at the minimum N value
plt.ylim(bottom=min(min(experimental_time_1), min(experimental_time_2 ) , min(experimental_time_3)))  # Start y-axis
    at the minimum time value


# Labels and title
plt.xlabel('N (Data Size)')
plt.ylabel('Time (seconds)')
plt.title('Solution 1 vs Solution 2 vs Solution 3')
plt.legend()
plt.grid(which="both", linestyle="--", linewidth=0.5)

# Save as PDF
plt.savefig('plot.pdf', format='pdf', bbox_inches='tight')

# Show the plot
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


Solution 3
Experimental Third Solution (Time x 10^-5)
Time (seconds)
2.50e-04
2.25e-04
2.00e-04
1.75e-04
1.50e-04
1.25e-04
1.00e-04
7.50e-05
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e9
N (Data Size)

To Draw the plot i used the below python script :

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import FuncFormatter

# Data from the tables
N = [1000003 , 2000003 , 4000037 , 8000009 , 16000057 , 32000011 , 64000031 , 128000003 , 256000001 , 512000009 ,
    1024000009 , 2048000011]
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]

# Convert times to consistent scales
experimental_time_3 = np.array(experimental_time_3) * 1e-5

plt.figure(figsize=(12, 7))

plt.plot(N, experimental_time_3 , 'o-', label='Experimental Third Solution (Time x 10^-5)', color='green')

plt.xlim(left=min(N))  # Start x-axis at the minimum N value
plt.ylim(bottom= min(experimental_time_3))  # Start y-axis at the minimum time value

# Labels and title
plt.xlabel('N (Data Size)')
plt.ylabel('Time (seconds)')
plt.title('Solution 3')
plt.legend()
plt.grid(which="both", linestyle="--", linewidth=0.5)

# Use FuncFormatter to enforce scientific notation
formatter = FuncFormatter(lambda x, pos: '{:.2e}'.format(x))  # '{:.2e}' enforces scientific notation

# Apply the formatter to the y-axis
plt.gca().yaxis.set_major_formatter(formatter)

# Save as PDF
plt.savefig('sqrt.pdf', format='pdf', bbox_inches='tight')

# Show the plot
plt.show()
```

Part 4 :

Q1 :

Algorithm Prime Number

```
Var
n, cmpt, i ,step integer;
Begin
cmpt ← 0;
i ← 1;
step ← 1;
print('Input Integer N>1 : ')
Read(n);
if ( n%2 !=0 ) then
    step ← 2;
end if
while i <= √n do
    if (n%i == 0 ) then
        cmpt ← cmpt + 1;
    end if
    i ← i + step;
end while
if ( cmpt == 1 ) then
    print('Prime Number');
else
    print('Not A Prime Number');
end if
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 ← 0	$fr = 1$ (one affectation)
i ← 1	$fr = 1$ (one affectation)
step ← 1	$fr = 1$ (one affectation)
print('Input Integer N>1 : ')	$fr = 1$ (one print)
Read(n)	$fr = 1$ (one read)
step ← 2	$fr = 1$ (one affectation)
while i <= $\sqrt{n}$ **do**	$fr = \frac{\sqrt{n}}{2} + 1$ (check the while condition $\frac{\sqrt{n}}{2} + 1$ times because step $= 2$)
cmpt ← cmpt + 1	$fr = 2$ (one affectation and one arithmetic operation (2) , repeated once because N is Prime)
i ← 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)

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

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

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

Worst Case (N is not prime & even)

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

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

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 $\sim O(\sqrt{n})$

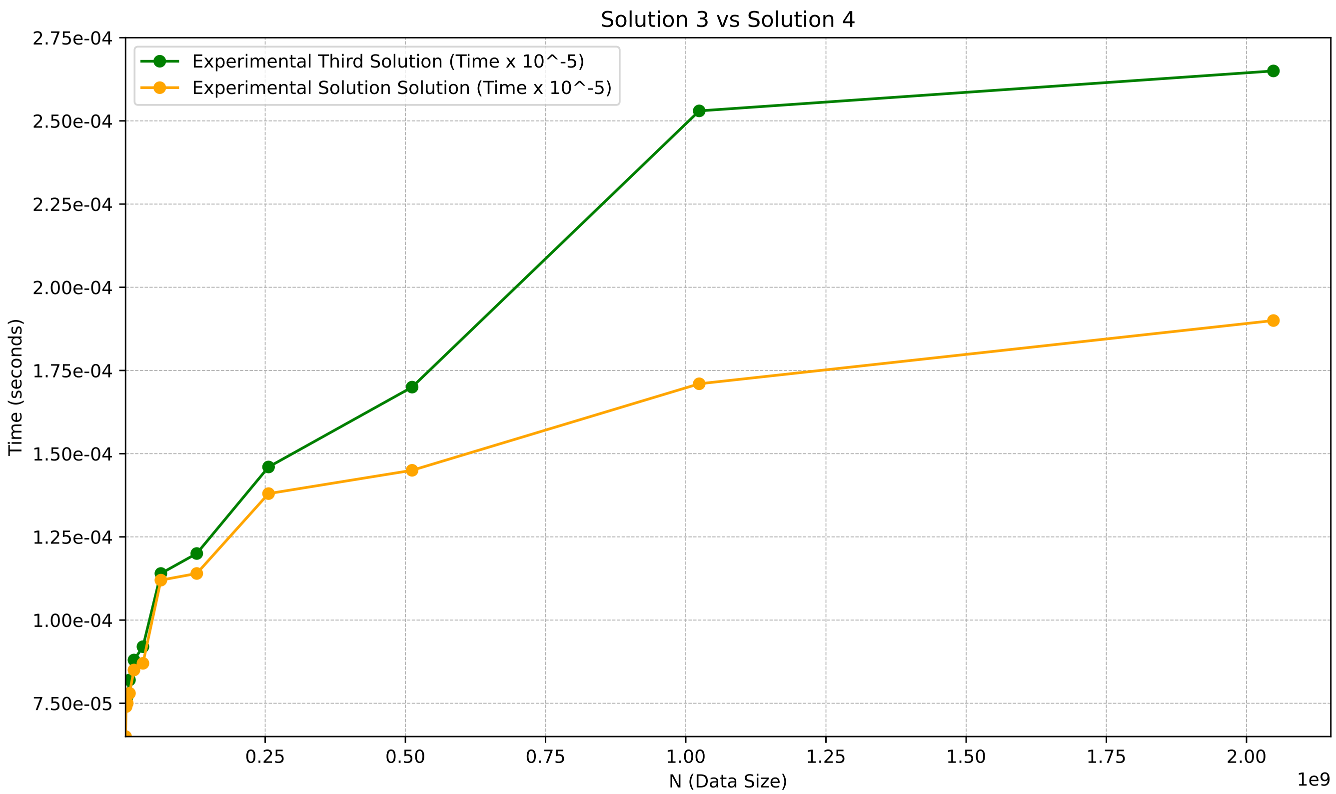
Q3 :

```c
#include<stdio.h>
#include<time.h>
#include<math.h>

int main() {

int i = 1;
int cmpt = 0;
int step = 1;
int n;

clock_t start_time = clock();

printf("Input Integer N>1 : ");
scanf("%d",&n);

double limit = sqrt(n);

if(n%2 != 0){
step = 2;
}

while (i<=limit) {
 if (n%i==0) {++cmpt;}
 i = i + step;
}

if ( cmpt == 1 ) {printf("\nprime number");}
else {printf("\nNot a prime number");}

clock_t end_time = clock();
double execution_time = (double) (end_time - start_time)/CLOCKS_PER_SEC;
printf("\nExecution Time %f seconds\n",execution_time);

return 0;
}
```

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 :

```python
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import FuncFormatter

# Data from the tables
N = [1000003 , 2000003 , 4000037 , 8000009 , 16000057 , 32000011 , 64000031 , 128000003 , 256000001 , 512000009 , 1024000009 , 2048000011]
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]
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]

# Convert times to consistent scales
experimental_time_3 = np.array(experimental_time_3) * 1e-5
experimental_time_4 = np.array(experimental_time_4) * 1e-5


plt.figure(figsize=(12, 7))

plt.plot(N, experimental_time_3 , 'o-', label='Experimental Third Solution (Time x 10^-5)', color='green')
plt.plot(N, experimental_time_4 , 'o-', label='Experimental Solution Solution (Time x 10^-5)', color='orange')

plt.xlim(left=min(N))  # Start x-axis at the minimum N value
plt.ylim(bottom= min(min(experimental_time_3), min(experimental_time_4)))  # Start y-axis at the minimum time value

# Labels and title
plt.xlabel('N (Data Size)')
plt.ylabel('Time (seconds)')
plt.title('Solution 3 vs Solution 4')
plt.legend()
plt.grid(which="both", linestyle="--", linewidth=0.5)

# Use FuncFormatter to enforce scientific notation
formatter = FuncFormatter(lambda x, pos: '{:.2e}'.format(x))  # '{:.2e}' enforces scientific notation

# Apply the formatter to the y-axis
plt.gca().yaxis.set_major_formatter(formatter)

# Save as PDF
plt.savefig('plot.pdf', format='pdf', bbox_inches='tight')

# Show the plot
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