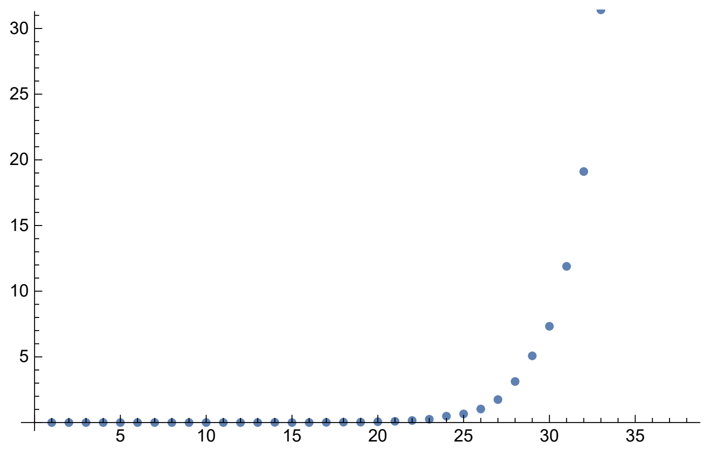
```
(* Fibonacci Numbers *)
(* Charlene Chu - September 17, 2017 *)

Remove["Global`*"]

f[1] = f[2] = 1; f[n_] := f[n - 1] + f[n - 2]

(* Question: Why is it so quick to compute f[6] but takes so long to compute f[40]? *)

(* Each time f[n] is called, it makes two recursive calls, f[n-1] and f[n-2]. So,
it makes at most 2^{n-2} calls since n≥3. Each time the function is called,
it makes one computation, the addition of f[n-1] and f[n-2]. So,
it will take at most c*2^{n-2} seconds to compute f[n],
where c is some constant. Conjecture: It will take exponential time to compute f[n]. *)

(* Let's look at the time it takes compute f[n] for n=3,...,40. *)

computationTimes = {};
For[i = 3, i < 41, i++,
  tempTime = Timing[f[i]][[1]]; AppendTo[computationTimes, tempTime];
 ];
computationTimes

{0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.015625, 0., 0., 0.015625, 0.03125,
 0.03125, 0.0625, 0.09375, 0.15625, 0.25, 0.484375, 0.65625, 1.03125, 1.75, 3.125,
 5.07813, 7.32813, 11.8906, 19.1094, 31.4219, 49.5781, 76.3594, 123.547, 211.219, 334.531}

ListPlot[computationTimes]
```



```
(* The graph of the time it takes to compute f[n] looks like an exponential
 function. We will try to find an exponential function that models this
 graph. We will first clean up the list of times to get rid of the zeros. The
 purpose of this is to avoid division by zero when finding the ratio. *)
```

```
list = {};
For[i = 1, i < Length[computationTimes] + 1, i++,
  If[computationTimes[[i]] ≠ 0, AppendTo[list, computationTimes[[i]]]]
 ];
list
```

```
{0.015625, 0.015625, 0.03125, 0.03125, 0.0625, 0.09375, 0.15625,
 0.25, 0.484375, 0.65625, 1.03125, 1.75, 3.125, 5.07813, 7.32813,
 11.8906, 19.1094, 31.4219, 49.5781, 76.3594, 123.547, 211.219, 334.531}
```

```
(* Using the cleaned up list, we will find the ratio of the time
 it takes to compute f[n+1] to the time it tkes to compute f[n]. *)
```

```
ratios = {};
For[i = 2, i < Length[list] + 1, i++,
  AppendTo[ratios, list[[i]] / list[[i - 1]]]
 ];
ratios
```

```
{1., 2., 1., 2., 1.5, 1.66667, 1.6, 1.9375, 1.35484, 1.57143, 1.69697, 1.78571, 1.625,
 1.44308, 1.6226, 1.6071, 1.64432, 1.57782, 1.54018, 1.61797, 1.70962, 1.58381}
```

```
(* Here, we calculate the common ratio, r, and the scale factor, a,
of the geometric progrssion that represents the time it takes to compute f[n]. *)
```

```
r = Mean[ratios]
```

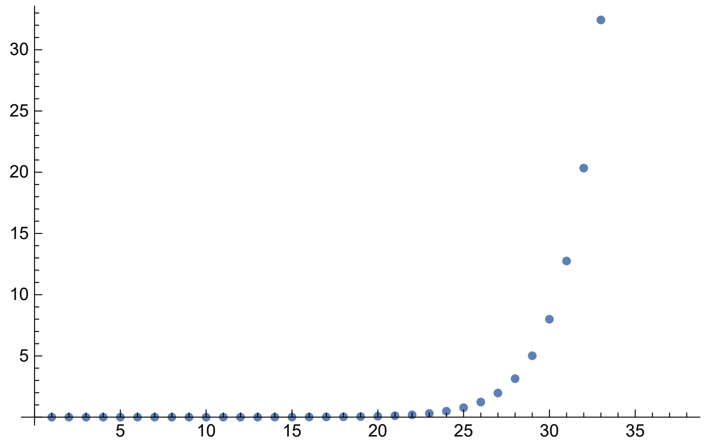
```
1.59476
```

```
a = computationTimes[[Length[computationTimes]]] / r^Length[computationTimes]
```

6.63837×10^{-6}

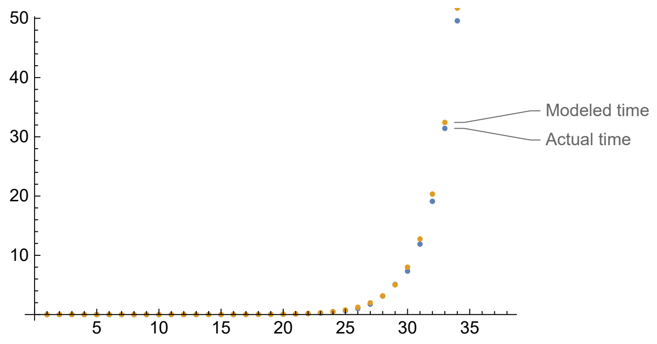
```
(* The time it takes to compute f[n] is modeled by the function t(n) = a*r^{n}. *)
```

```
modelTimes = Table[a * r^i, {i, Length[computationTimes]}];
ListPlot[modelTimes]
```



```
(* Notice that the graph of this model looks very similar to the graph of the
 actual time it takes to compute f[n]. Let's look at both graphs together. *)
```

```
ListPlot[{Labeled[computationTimes, "Actual time"], Labeled[modelTimes, "Modeled time"]}]
```



```
(* We can see that the the graphs are very close. Therefore,
the time takes to compute f[n] is closely modeled by the function t(n)=
 a*r^{n}. Conclusion: It takes exponential time to compute f[n]. *)

(* Answer: The reason why it takes so long to compute f[40] is because it takes
  exponential time to compute f[n]. Exponential time is not very efficient
  in algorithm run time complexity. Constant time is the most efficient,
followed by logarithmic time, linear time,
and polynomial time. Factorial time is the least efficient. *)
```