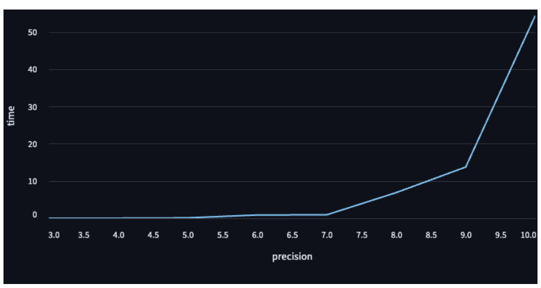
[Final] Approximation of e

Three Approximation

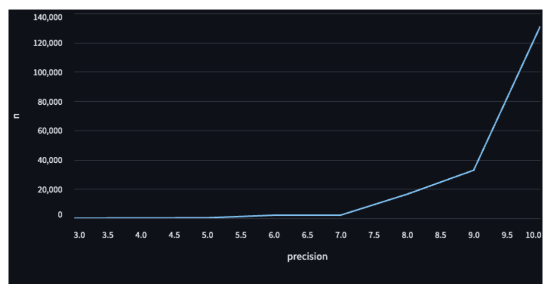
Approximation 1

$$ln\, e = \int_1^e \frac{1}{t} dt = 1$$

▼ Time



▼ n value



▼ Code solution

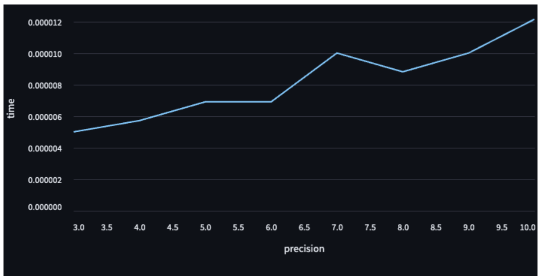
```
def approximation1(n:int,prec:int = 100) -> Decimal:
    n = int(n)
    getcontext().prec = prec

    def riemann_sum(e:Decimal) -> Decimal:
        delta_x = Decimal(e - 1) / Decimal(n)
```

```
        d_dot_5 = Decimal(0.5)

        s = Decimal(0)
        for i in range(1,n + 1):
            s +=  Decimal(1)/Decimal(1+Decimal(i-d_dot_5)*del
        return s * delta_x

    lower = Decimal("2")
    upper = Decimal("3")

    step = 0

    old_s = 0

    while (step < n):

        middle = Decimal((lower+upper)/2)
        s = riemann_sum(middle)

        if old_s == s:
            break
        old_s = s

        if s > 1:
            upper = middle
        else:
            lower = middle

        step += 1
        print(f"step {step:4} : e value {middle}")
    return Decimal((lower+upper)/2)
```

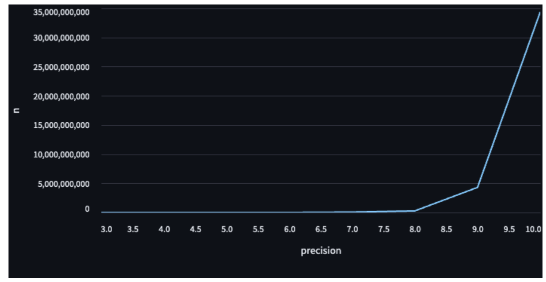
Approximation 2

$$e = \lim_{\delta \to 0}(1+\delta)^{\frac{1}{\delta}} = \lim_{n \to \infty}(1+\frac{1}{n})^n$$

▼ Time



▼ n value



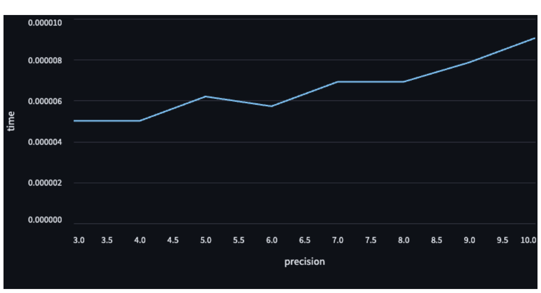
▼ Code solution

```
def approximation2(n:int,prec:int = 100) -> Decimal:
    getcontext().prec = prec

    s = Decimal(1) + Decimal(1) / Decimal(n)
    s = s**Decimal(n)

    return s
```

Approximation 3

$$e = \Sigma_{n=0}^{\infty}\frac{1}{n!}$$

▼ Time



▼ n value

▼ Code solution

```
def approximation3(n:int,prec:int = 100) -> Decimal:

    getcontext().prec = prec


    tmp = Decimal(1)
    s = Decimal(1)

    old_s = 0

    for i in range(1,n + 1):
        s += tmp / Decimal(i)
        tmp /= Decimal(i)

        if old_s == s:
            break
        old_s = s
    return s
```

Testing

▼ Code to test time and using n value

```
def test(tail:int=10,prec:int=15) -> list[dict]:

    getcontext().prec = prec
    real_e = "2.71828182845904523536028747135266249775724709

    data = []

    for precision in range(3,tail+1):
```

```
        n = 1
        start = time.time()
        test_e = solve.approximation1(n,prec=prec)
        time_delta = time.time() - start
        while str(test_e)[:precision+2] != real_e[:precision-

            start = time.time()
            n *= 2
            test_e = solve.approximation1(n,prec=prec)
            time_delta = time.time()-start
        data.append({
            "precision" : precision,
            "Approximation" : "Approximation1",
            "time": time_delta,
            "e_value" : str(test_e),
            "n" : n
        })

        n = 1
        start = time.time()
        test_e = solve.approximation2(n,prec=prec)
        time_delta = time.time() - start
        while str(test_e)[:precision+2] != real_e[:precision-

            start = time.time()
            n *= 2
            test_e = solve.approximation2(n,prec=prec)
            time_delta = time.time()-start
        data.append({
            "precision" : precision,
            "Approximation" : "Approximation2",
            "time": time_delta,
            "e_value" : str(test_e),
            "n" : n
        })
```

```
        n = 1
        start = time.time()
        test_e = solve.approximation3(n,prec=prec)
        time_delta = time.time() - start
        while str(test_e)[:precision+2] != real_e[:precision+

            start = time.time()
            n += 1
            test_e = solve.approximation3(n,prec=prec)
            time_delta = time.time()-start
        data.append({
            "precision" : precision,
            "Approximation" : "Approximation3",
            "time": time_delta,
            "e_value" : str(test_e),
            "n" : n
        })


    return data
```

Compare

Plot the image to compare the difference. (Using streamlit)

`data.json` is the time and n data from last step.

The output images are put at the approximation part.

▼ Code the plot the image

```
import streamlit as st
import pandas as pd
import json
```

```
def get_data(approximation:str,filename="data.json") -> pd.Da

    with open(filename,"r") as f:
        raw_data = json.load(f)

    data = filter(lambda x : x["Approximation"]==approximatio

    df = pd.json_normalize(data,
                    meta=["precision",
                        "Approximation",
                        "e_value",
                        "n",
                        "time"])

    return df

st.title("Approximations of e")
st.write("## Approximation 1")
st.latex(r"ln \, e = \int_1^{e}\frac{1}{t} dt= 1")
st.subheader("Time chart")
st.line_chart(get_data("Approximation1"), x="precision", y="t

st.subheader("n value chart")
st.line_chart(get_data("Approximation1"), x="precision", y="n

st.write("## Approximation 2")
st.latex(r"e = \lim_{\delta \to 0}(1+\delta)^\frac{1}{\delta}

st.subheader("Time chart")
st.line_chart(get_data("Approximation2"), x="precision", y="t

st.subheader("n value chart")
st.line_chart(get_data("Approximation2"), x="precision", y="n
```

```
st.write("## Approximation 3")
st.latex(r"e = \Sigma_{n=0}^{\infty} \frac{1}{n!}")

st.subheader("Time chart")
st.line_chart(get_data("Approximation3"), x="precision", y="t

st.subheader("n value chart")
st.line_chart(get_data("Approximation3"), x="precision", y="r
```

Conclusion

Approximation1 的逼近方法較為特殊，使用 $ln\,e = \int_1^e \frac{1}{t}dt = 1$ 的定義，因此我透過黎曼和計算積分值，並使用二分逼近法來逐步逼近 e，時間複雜度為 $O(n^2)$ ，執行效率最差，精度與執行時間及 n 值呈指數關係，所解如上。

假設次方運算的時間複雜度為：$O(n)$，此三個 Approximation 的時間複雜度分別為：

1. $O(n^2)$
2. $O(n)$
3. $O(n)$

而其中因為 Approximation3 的收斂速度較快，因此如需達到同樣的精度，在 Approximation2 中需要使用較大的 n 值，可以看到在圖表中，n 值的增長在 Approximation2 中呈指數增長，而在 Approximation3 中呈線性增長。

就結論來說，若要比較效率：

$$eff_3 > eff_2 > eff_1$$

但，由於我使用 Python 來實作，在 Approximation2 內使用的次方運算有經過底層優化，因此在此實現中，如果需要達到同樣精度，所需的時間比較大約如下：

$$T_3 \approx T_2 < T_1$$