Actividad 2: Comparación de tiempos de ejecución entre algoritmos iterativos y recursivos para 3 problemas generales

Alberto Benavides

En esta actividad se desea comparar el desempeño en tiemmpo de ejecución de tres algoritmos en sus versiones recursivas e iterativas. Se trata de 1) un algoritmo para calcular el valor del elemento de la serie Fibonacci dada su posición, 2) el factorial de un número y 3) un algoritmo para determinar si una cadena de texto es palíndroma.

Fibonacci

La serie Fibonacci es una serie en que cada elemento es una suma de los dos anteriores, teniendo como elementos iniciales $f_0 = 0$ y $f_1 = 1$, de modo que para todos los demás elementos se sigue

$$f_n = f_{n-2} + f_{n-1}.$$

Ahora, se definirá este algoritmo en su vertiente recursiva

In [23]:
```
def FibRecursivo(n):
    if n < 2:
        return n
    else:
        return FibRecursivo(n-2) + FibRecursivo(n-1)
```

e iterativa

In [24]:
```
def FibIterativo(n):
    a = 0
    b = 1
    for x in range(n - 1):
        t = b
        b = b + a
        a = t
    return b
```

Para comprobar que estos algoritmos funcionan, se puede usar como variable de entrada un número $20 < r < 50$ obtenido de una distribución normal aleatoria de la librería random de Python.

In [125]:
```
from random import randint

r = randint(20, 30)
FibRecursivo(r) == FibIterativo(r)
```

Out[125]: True

Factorial

El factorial de un número n es el producto de todos los números enteros desde el 1 hasta dicho número. Es decir, el factorial de 5, denotado por $5!$ es

$$5! = 1 \times 2 \times 3 \times 4 \times 5.$$

Para calcular recursivamente el factorial de un número se puede ejecutar esta función

In [26]:
```
def FacRecursivo(n):
    if n > 1:
        return n * FacRecursivo(n - 1)
    else:
        return n
```

mientras que el cálculo iterativo se realiza como sigue

In [126]:
```
def FacIterativo(n):
    total = 1
    for x in range(n):
        total *= (n - x)
    return total
```

Para este algoritmo se procede de manera análoga a la comprobación hecha en Fibonacci

In [28]:
```
r = randint(20, 50)
FacRecursivo(r) == FacIterativo(r)
```

Out[28]: True

Palíndromo

Un palíndromo es una palabra que es idéntica escrita originalmente que al revés. En programación se llama cadena de texto un conjunto de caracteres agrupados uno tras otro. Esto es lo más parecido a una palabra en términos computacionales, por lo que se usará esta estructura para generar palíndromos. Para este fin, se desarrolla un algoritmo que generara cadenas palíndromas aleatorios de longiud $l \times 2$ a partir de la siguiente función:

In [127]:
```
from string import ascii_lowercase
from random import choice, shuffle

def PalAleatorio(l):
    letras = ascii_lowercase
    pal = ''.join(choice(letras) for i in range(l))
    # https://www.educative.io/edpresso/how-do-you-reverse-a-string-in-python
    pal += pal[::-1]
    # Desordenar cadenas
    '''
    if randint(0, 1) == 1:
        t = list(pal)
        shuffle(t)
        pal = ''.join(t)
    '''
    return pal
```

Con esto, ya se puede comprobar si el resultado es el mismo para el algoritmo recursivo

In [128]:
```
def PalRecursivo(s):
    if(len(s) == 0 or len(s) == 1):
        return True
    elif s[0] != s[-1]:
        return False
    else:
        return PalRecursivo(s[1: -1])
```

e iterativo

In [129]:
```
def PalIterativo(s):
    if(len(s) == 0 or len(s) == 1):
        return True
    for x in range(len(s) // 2):
        if s[x] != s[(x + 1) * -1]:
            return False
    return True
```

de esta manera

In [130]:
```
r = randint(20, 50)
s = PalAleatorio(r)

PalRecursivo(s) == PalIterativo(s)
```

Out[130]:
```
True
```

Experimentación

Ahora se medirán tiempos de ejecución comparando estos tres algoritmos de manera iterativa y recursiva. Para ello se utiliza la librería timeit que permite medir el tiempo de ejecución de determinado número de repeticiones de un algoritmo. Para esta serie de experimentos computacionales, se eligen 100 repeticiones. Se ha optado por utilizar esta librería puesto que la librería time requiere se especifiquen las repeticiones del experimento, lo que inhibe los procesos

Fibonacci

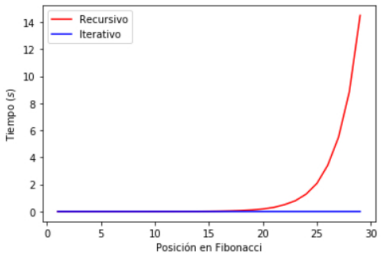
In [34]:
```
import timeit

nFib = []
recFib = []
iteFib = []

for x in range(1, 30):
    nFib.append(x)
    if __name__ == '__main__':
        recFib.append(timeit.timeit("FibRecursivo({})".format(x), setup="from __main__ import FibRecursivo", number=100))
        iteFib.append(timeit.timeit("FibIterativo({})".format(x), setup="from __main__ import FibIterativo", number=100))
```

In [35]:
```
import matplotlib.pyplot as plt

plt.plot(nFib, recFib, c="red", label="Recursivo")
plt.plot(nFib, iteFib, c="blue", label="Iterativo")
plt.xlabel("Posición en Fibonacci")
plt.ylabel("Tiempo ($s$)")
plt.legend()
plt.show()
```

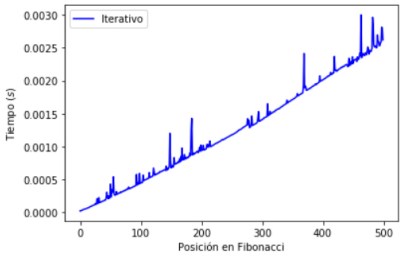


Se repetirá para el iterativo con muchas más repeticiones, para ver su comportamiento de manera individual.

In [36]:
```
nFib = []
iteFib = []

for x in range(500):
    nFib.append(x)
    if __name__ == '__main__':
        iteFib.append(timeit.timeit("FibIterativo({})".format(x), setup="from __main__ import FibIterativo", number=100))
```

In [37]:
```
plt.plot(nFib, iteFib, c="blue", label="Iterativo")
plt.xlabel("Posición en Fibonacci")
plt.ylabel("Tiempo ($s$)")
plt.legend()
plt.show()
```



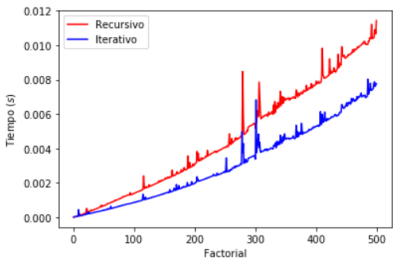
Factorial

In [94]:
```
nFac = []
```

```
recFac = []
iteFac = []

for x in range(1, 500):
    nFac.append(x)
    if __name__ == '__main__':
        recFac.append(timeit.timeit("FacRecursivo({})".format(x), setup="from __main__ import FacRecursivo", number=100))
        iteFac.append(timeit.timeit("FacIterativo({})".format(x), setup="from __main__ import FacIterativo", number=100))
```

In [95]:
```
plt.plot(nFac, recFac, c="red", label="Recursivo")
plt.plot(nFac, iteFac, c="blue", label="Iterativo")
plt.xlabel("Factorial")
plt.ylabel("Tiempo ($s$)")
plt.legend()
plt.show()
```

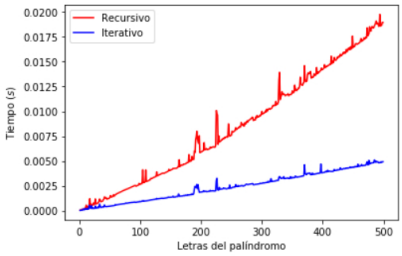


Palíndromos

In [123]:
```
nPal = []
recPal = []
itePal = []

for x in range(1, 500):
    nPal.append(x)
    s = PalAleatorio(x)
    recPal.append(timeit.timeit(stmt="PalRecursivo('{}')".format(s), setup="from __main__ import PalRecursivo", number=100))
    itePal.append(timeit.timeit(stmt="PalIterativo('{}')".format(s), setup="from __main__ import PalIterativo", number=100))
```

In [124]:
```
plt.plot(nPal, recPal, c="red", label="Recursivo")
plt.plot(nPal, itePal, c="blue", label="Iterativo")
plt.xlabel("Letras del palíndromo")
plt.ylabel("Tiempo ($s$)")
plt.legend()
plt.show()
```



Conclusiones

Todos los algoritmos han probado ser más rápidos en su versión iterativa respecto a su misma versión recursiva. Este comportamiento puede ser explicable debido a la pila de llamadas que se acumulan al llamarse una función a sí misma de manera recursiva. Es probable que estos algoritmos sean más rápidos si se trabajan con múltiples procesadores que permitas distribuir funciones en distintos hilos de procesamiento, lo cual suele ser irrealizable con funciones iterativas que dependen de secuencias anteriores para proceder.

Por otro lado, resalta el crecimiento exponencial del algoritmo recursivo de Fibonacci que puede explicarse debido a que conforme crece la posición del valor calculado, las llamadas crecen exponencialmente, ramificándose dos a la vez.

De esta manera, se puede concluir que es recomendable ejecutar algoritmos iterativos si no se dispone de varios núcleos de procesammiento, mientras que sería preferible optar por versiones iterativas de algoritmos en caso contrario.