FIBONACHI Bottom Up

```python
import time

def fibonachiBUP(n):
    if not isinstance(n, int):
        raise TypeError("n debe ser un entero >= 0")
    if n < 0:
        raise ValueError("n debe ser >= 0")

    if n == 0:
        return 0
    if n == 1:
        return 1

    f = [0, 1]  # f[0]=0, f[1]=1
    while len(f) <= n:
        f.append(f[-1] + f[-2])
    return f[n]

def main():
    Pruebas = [0, 1, 2, 5, 10, 20, 30]
    print("Bottom-up DEL FIBONACHI")
    for t in Pruebas:
        t0 = time.perf_counter()
        val = fibonachiBUP(t)
        t1 = time.perf_counter()
        print(f"F({t}) = {val}  (tiempo: {(t1-t0)*1000:.4f} ms)")

if __name__ == "__main__":
    main()
```

```
Bottom-up DEL FIBONACHI
F(0) = 0  (tiempo: 0.0020 ms)
F(1) = 1  (tiempo: 0.0035 ms)
F(2) = 1  (tiempo: 0.0035 ms)
F(5) = 5  (tiempo: 0.0023 ms)
F(10) = 55  (tiempo: 0.0018 ms)
F(20) = 6765  (tiempo: 0.0028 ms)
F(30) = 832040  (tiempo: 0.0036 ms)
```

Fibonachi Top Down

```python
import time
from functools import lru_cache

def fibonacci_top_down(n, memori=None):
    if not isinstance(n, int):
        raise TypeError("n debe ser un entero >= 0")
    if n < 0:
        raise ValueError("n debe ser >= 0")

    if memori is None:
        memori = {0: 0, 1: 1}

    if n in memori:
        return memori[n]

    res = fibonacci_top_down(n-1, memori) + fibonacci_top_down(n-2, memori)
    memori[n] = res
    return res

# Alternativa rápida con lru_cache
@lru_cache(maxsize=None)
def Fibonachi_fast(n):
    if n < 0:
        raise ValueError("n debe ser >= 0")
    if n == 0:
        return 0
    if n == 1:
        return 1
    return Fibonachi_fast(n-1) + Fibonachi_fast(n-2)

def main():
    pruebas = [0, 1, 2, 5, 10, 20, 30]
    print("Top-down Fibonacci (diccionario)")
    for act in pruebas:
        p0 = time.perf_counter()
        val = fibonacci_top_down(act)
        p1 = time.perf_counter()
        print(f"F({act}) = {val}  (tiempo: {(p1-p0)*1000:.4f} ms)")

    print("\nTop-down Fibonacci (lru_cache) - comparativa")
    for act in pruebas:
        p0 = time.perf_counter()
        val = Fibonachi_fast(act)
        p1 = time.perf_counter()
        print(f"F({act}) = {val}  (tiempo: {(p1-p0)*1000:.4f} ms)")

if __name__ == "__main__":
    main()
```

```
Top-down Fibonacci (diccionario)
F(0) = 0  (tiempo: 0.0040 ms)
F(1) = 1  (tiempo: 0.0024 ms)
F(2) = 1  (tiempo: 0.0025 ms)
F(5) = 5  (tiempo: 0.0033 ms)
F(10) = 55  (tiempo: 0.0039 ms)
F(20) = 6765  (tiempo: 0.0072 ms)
F(30) = 832040  (tiempo: 0.0103 ms)

Top-down Fibonacci (lru_cache) - comparativa
F(0) = 0  (tiempo: 0.0016 ms)
F(1) = 1  (tiempo: 0.0007 ms)
F(2) = 1  (tiempo: 0.0012 ms)
F(5) = 5  (tiempo: 0.0017 ms)
F(10) = 55  (tiempo: 0.0020 ms)
```

QuickSORT

```python
import random
import time

def quicksort(lista):
    if not isinstance(lista, list):
        raise TypeError("quicksort espera una lista")
    if len(lista) <= 1:
        return lista[:]
    arr = lista[:]

    def partir(inicio, fin):
        pivot = arr[inicio]
        lft = inicio + 1
        rig = fin

        while True:
            while lft <= rig and arr[lft] <= pivot:
                lft += 1
            while rig >= lft and arr[rig] >= pivot:
                rig -= 1

            if rig < lft:
                break
            else:
                arr[lft], arr[rig] = arr[rig], arr[lft]

        # colocar pivote en su posición correcta
        arr[inicio], arr[rig] = arr[rig], arr[inicio]
        return rig

    def quicksortau(inicio, fin):
        if inicio < fin:
            p = partir(inicio, fin)
            quicksortau(inicio, p - 1)
            quicksortau(p + 1, fin)

    quicksortau(0, len(arr)-1)
    return arr

def main():
    random.seed(0)
    tests = [
        [],
        [1],
        [2, 1],
        [5, 3, 8, 1, 2],
        [random.randint(0, 1000) for _ in range(1000)]
    ]
    print("Quicksort (Divide & Conquer)")
    for i, t in enumerate(tests):
        p0 = time.perf_counter()
        pram = quicksort(t)
        p1 = time.perf_counter()
        ok = pram == sorted(t)
        print(f"Test {i}: len={len(t)}  OK={ok}  tiempo={(p1-p0)*1000:.4f} ms")

if __name__ == "__main__":
    main()
```

```
Quicksort (Divide & Conquer)
Test 0: len=0  OK=True  tiempo=0.0040 ms
Test 1: len=1  OK=True  tiempo=0.0043 ms
Test 2: len=2  OK=True  tiempo=0.0056 ms
Test 3: len=5  OK=True  tiempo=0.0058 ms
Test 4: len=1000  OK=True  tiempo=1.1974 ms
```