Nicolle Ruiz Quintero (8974036) - Alejandro Fabregas (8977914)

Qué se obtiene al ejecutar algoritmo1(8)? Explique

Reemplazamos n con el 8, el bucle for se va a ejecutar si **i** es mayor a cero, entonces en la primera iteración **i** es 64 ya que 8*8 es 64 y **J** inicia como 1.

Lo primero que hace el bucle es calcular la suma de **i j** donde es igual a 65, esta imprime “suma 65” y además incrementa el valor de **j** en 1, para después dividirlo entre 2, por lo que 65 pasa a ser 32; en la segunda iteración del bucle **i** = 32 y **j** = 2, se calcula la suma dando 34 vuelve a imprimir lo anterior, pero con 34 y j aumenta su valor en 1, después se vuelve a 16 y así sucesivamente hasta que **i**=0 por lo que la condición en el ciclo for no se podría ejecutar

Qué se obtiene al ejecutar algoritmo2(8)? Explique.

Reemplazamos n por el 8, la variable res es 1 además incrementa de 2 en 2, comienza el primer bucle for, donde la variable i=1 y se incrementa en 4 en cada iteración; el bucle tiene como condición ejecutarse siempre y cuando **i** sea menor o igual a 2 x n en este caso 16.

Dentro del buce for comienza el segundo for, donde la variable **j** inicia en 1 y se incrementa en 1 en cada iteración, este bucle se ejecutara mientras que **j x j** sea igual o menor que n, teniendo en cuenta estas condiciones realizaremos la ejecución.

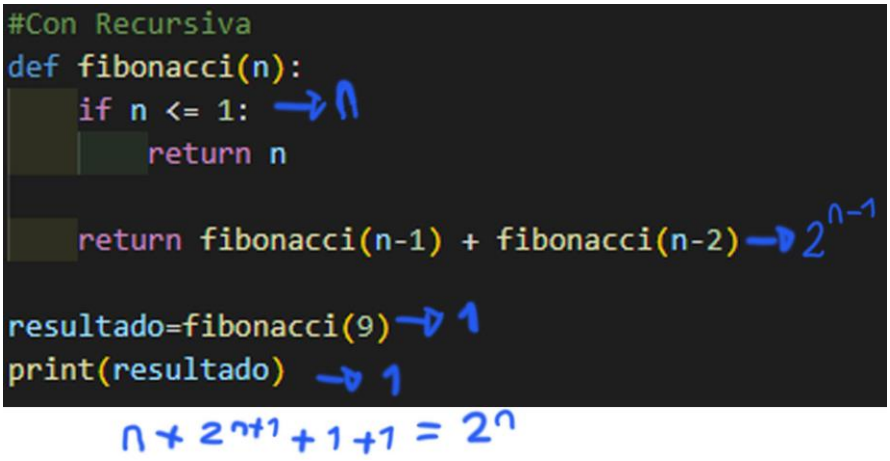
En la primera iteración el **i** es 1 por lo que es menor que 16 y **j** es 1 por lo que es menor que 8, res incrementa en 2 por lo que ahora res=3

En la segunda iteración **i** es 5 y **j** incrementa a 2 pero su multiplicación sigue siendo menor a n, y res incrementa en 2 por lo que ahora es 5...

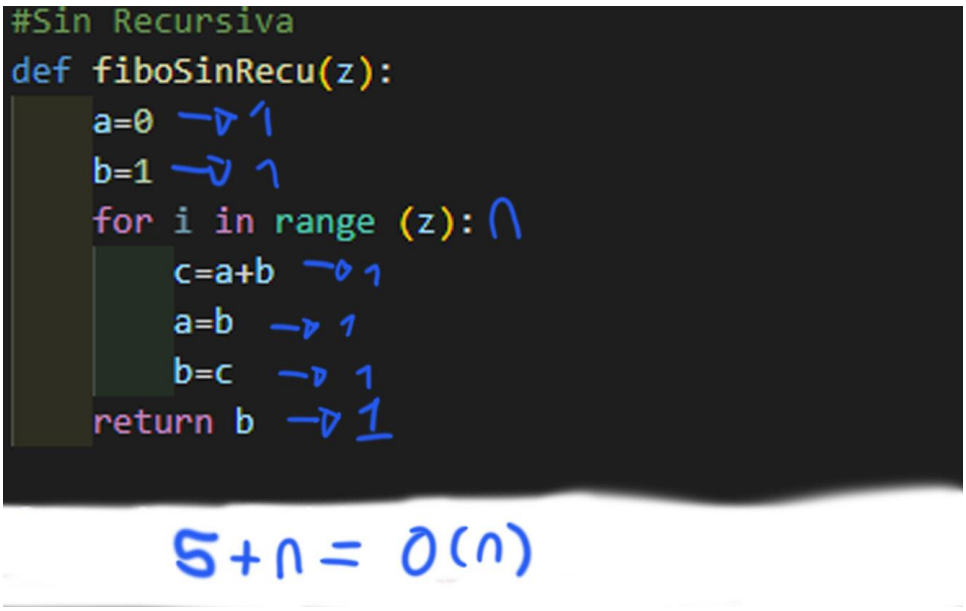
Y así hasta que los dos bucles rompan sus condiciones dándonos como resultado

res=49

PUNTO 6



PUNTO 7



8

A) Porque el iterativo es más rápido que el recursivo, esto se debe a medida que aumenta el valor de **n**, el tiempo que tarda en ejecutarse el algoritmo recursivo se vuelve cada vez más largo en comparación con el algoritmo iterativo, porque uno es lineal y el otro es exponencial.

B) Código del punto 4 del profesor

```python
def esPrimo(n):
    if n < 2: ans = False
    else:
        i, ans = 2, True
        while i * i <= n and ans:
            if n % i == 0: ans = False
            i += 1
```

$n + 1 + \sqrt{n} + n + n = \sqrt{n}$

Código del punto 4 mío:

```python
def esPrimo(num):
    contador=0
    for i in range(2,num):
        if num%i==0:
            contador+=1
    if contador==0:
        return True
```

$O(n)$

Esta es la Tarea 2 del curso *Estructuras de Datos*, 2023-1. La actividad se debe realizar en **parejas** o de forma **individual**. Sus soluciones deben ser entregadas a través de BrightSpace a más tardar el día **15 de Febrero a las 22:00**. En caso de dudas y aclaraciones puede escribir por el canal `#tareas` en el servidor de *Discord* del curso o comunicarse directamente con los profesores y/o el monitor.

Condiciones Generales

- Para la creación de su código y documentación del mismo use nombres en lo posible cortos y con un significado claro. La primera letra debe ser minúscula, si son más de 2 palabras se pone la primera letra de la primera palabra en minúscula y las iniciales de las demás palabras en mayúsculas. Además, para las operaciones, el nombre debe comenzar por un verbo en infinitivo. Esta notación se llama *lowerCamelCase*.

  Ejemplos de funciones: `quitarBoton`, `calcularCredito`, `sumarNumeros`

  Ejemplos de variables: `sumaGeneralSalario`, `promedio`, `nroHabitantes`

- Todos los puntos de la tarea deben ser realizados en un único archivo llamado `tarea2.pdf`.

Ejercicios de Complejidad Teórica

1. Indique el número de veces que se ejecuta cada línea y determine cuál es la complejidad del siguiente algoritmo:

```
void algoritmo1(int n){
    int i, j = 1;
    for(i = n * n; i > 0; i = i / 2){
        int suma = i + j;
        printf("Suma %d\n", suma);
        ++j;
    }
}
```

   Qué se obtiene al ejecutar `algoritmo1(8)`? Explique.

2. Indique el número de veces que se ejecuta cada línea y determine cuál es la complejidad del siguiente algoritmo:

```
int algoritmo2(int n){
    int res = 1, i, j;

    for(i = 1; i <= 2 * n; i += 4)
        for(j = 1; j * j <= n; j++)
            res += 2;
}
```

```
    return res;
}
```

Qué se obtiene al ejecutar `algoritmo2(8)`? Explique.

3. Indique el número de veces que se ejecuta cada línea y determine cuál es la complejidad del siguiente algoritmo:

```
void algoritmo3(int n){
    int i, j, k;
    for(i = n; i > 1; i--)
        for(j = 1; j <= n; j++)
            for(k = 1; k <= i; k++)
                printf("Vida cruel!!\n");
}
```

4. Indique el número de veces que se ejecuta cada línea y determine cuál es la complejidad del siguiente algoritmo:

```
int algoritmo4(int* valores, int n){
    int suma = 0, contador = 0;
    int i, j, h, flag;

    for(i = 0; i < n; i++){
      j = i + 1;
      flag = 0;
      while(j < n && flag == 0){
         if(valores[i] < valores[j]){
            for(h = j; h < n; h++){
              suma += valores[i];
            }
         }
         else{
            contador++;
            flag = 1;
         }
         ++j;
      }
    }
    return contador;
}
```

¿Qué calcula esta operación? Explique.

5. Indique el número de veces que se ejecuta cada línea y determine cuál es la complejidad del siguiente algoritmo:

```
void algoritmo5(int n){
  int i = 0; →1
  while(i <= n){ → n/((n/5)+1)
    printf("%d\n", i); →1
    i += n / 5; →n
  }
}
```

1+n((n/5)+1)+1+n = O(n)

Complejidad Teórico-Práctica

6. Escriba en Python una función que permita calcular el valor de la función de Fibonacci para un número n de acuerdo a su definición recursiva. Tenga en cuenta que la función de Fibonacci se define recursivamente como sigue:

$$Fibo(0) = 0$$
$$Fibo(1) = 1$$
$$Fibo(n) = Fibo(n-1) + Fibo(n-2)$$

Obtenga el valor del tiempo de ejecución para los siguientes valores (en caso de ser posible):

Tamaño Entrada	Tiempo	Tamaño Entrada	Tiempo
5	0m0.033s	35	0m13.209s
10	0m0.046s	40	0m2.613s
15	0m0.034s	45	0m2.715s
20	0m0.034s	50	0m2.642s
25	0m0.091s	60	0m2.645s
30	0m1,129s	100	0m2.648s

Cuál es el valor más alto para el cuál pudo obtener su tiempo de ejecución? Qué puede decir de los tiempos obtenidos? Cuál cree que es la complejidad del algoritmo?

7. Escriba en Python una función que permita calcular el valor de la función de Fibonacci utilizando ciclos y sin utilizar recursión. Halle su complejidad y obtenga el valor del tiempo de ejecución para los siguientes valores:

Tamaño Entrada	Tiempo	Tamaño Entrada	Tiempo
5	0m0.033s	45	0m0,36s
10	0m0,25s	50	0m0,38
15	0m0,31s	100	0m0,29s
20	0m0,23s	200	0m0,37s
25	0m0,26s	500	0m
30	0m0,34s	1000	0m
35	0m0,33s	5000	0m
40	0m0,27s	10000	m

8. Ejecute la operación `mostrarPrimos` que presentó en su solución al ejercicio 4 de la Tarea 1 y también la versión de la solución a este ejercicio que se subirá a la página del curso con los siguientes valores y mida el tiempo de ejecución:

Tamaño Entrada	Tiempo Solución Propia	Tiempo Solución Profesores
100	0m0.050s	0m0.032s
1000	0m0.126s	0m0.031s
5000	0m2.687s	0m0.050s
10000	0m10.441s	0m0.069s
50000	replit no lo calcula	0m1.023s
100000	replit no lo calcula	0m2.558s
200000	replit no lo calcula	0m5.923s

Responda las siguientes preguntas:

(a) ¿qué tan diferentes son los tiempos de ejecución y a qué cree que se deba dicha diferencia?

(b) ¿cuál es la complejidad de la operación o bloque de código en el que se determina si un número es primo en cada una de las soluciones?