ut2_pd2

Ejercicio 1

1. Desarrolla en seudocódigo, en forma recursiva, un algoritmo para calcular el factorial de un cierto número entero que se pasa como parámetro.
   - Identifica claramente el caso base y la sentencia que lo contempla.
   - ¿Puedes verificar que siempre el algoritmo progresará hacia el caso base?
2. Analiza el orden del tiempo de ejecución del algoritmo.
3. Implementa el algoritmo (en JAVA) y pruébalo:
   • ¿Qué sucede si el número es negativo?
   • Verifica que factorial(4), factorial(5), y factorial(0) produzcan los resultados

Solución

1. Seudocódigo:
   - El algoritmo siempre progresara al caso base porque en cada llamada recursiva n disminuye en uno. Entonces, eventualmente el valor de n será 0. Esto sucede unicamente si como precondición establecemos que n es un número mayor o igual a 0, sino el seudocódigo no funcionara. Entraría en un loop infinito restantando dos números negativos hasta dar error.

```
Algoritmo calcularFactorial(int n)
    si (n = 0)
        devolver 1; // Caso base
    sino
        devolver (n * factorial(n-1))
```

2. Calculo el orden de la siguiente manera

```
Algoritmo calcularFactorial(int n)
    si (n = 0) // O(1)
        devolver 1; // 0(1)
    sino
        devolver (n * factorial(n-1)) // O(1)
```

   - sabemos que hace n llamadas recursivas
   - Orden del método → n * (Orden del bloque) → n * (1+1+1) → n * 1 → Orden(n)

3. Si el numero es negativo, devuelve

```
at com.example.ut2_pd2.Factorial.calcularFactorial(Factorial.java:9)
```

Ejercicio 2

1. Desarrolla en seudocódigo, en forma recursiva, el algoritmo Algoritmo SumaLineal(A, n), que se describe en la ppt de clase sobre recursividad.
   - Identifica claramente el caso base y la sentencia que lo contempla
   - ¿Puedes verificar que siempre el algoritmo progresará hacia el caso base?
2. Analiza el orden del tiempo de ejecución del algoritmo
3. Implementa el algoritmo (en JAVA) y pruébalo:
   • ¿Qué sucede si el parámetro n es negativo?
   • ¿Qué sucede si el vector A está vacío?

Solución

1. Seudocódigo:
   - El algoritmo siempre progresara al caso base porque en cada llamada recursiva n disminuye en uno (`SumaLineal(A, n - 1)`). Entonces, eventualmente el valor de n será 1.

```
Algoritmo SumaLineal(A, n):
COM
SI n = 1 // Caso base
    devolver A[0]
SINO
devolver SumaLineal(A, n - 1) + A[n - 1])
FIN
```

2. Calculo el orden de la siguiente manera

```
Algoritmo SumaLineal(A, n):
COM
SI n = 1 // O(1)
    devolver A[0] // O(1)
SINO
devolver SumaLineal(A, n - 1) + A[n - 1]) // O(1)
FIN
```

- sabemos que hace n-1 llamadas recursivas porque el caso base es 1 y no 0
- Orden del método → (n-1) * (Orden del bloque) → (n-1) * (1+1+1) → (n-1) * 1 → Orden(n-1)

3. Si n es negativo devuelve:

   `delvuelve el mensaje "at com.example.SumaLineal.calcularSumaLineal(SumaLineal.java:8`

   Si A esta vacio devuelve:

   `int suma3 = SumaLineal.calcularSumaLineal(B, 0);`

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0
at com.example.SumaLineal.calcularSumaLineal(SumaLineal.java:8)
```

Ejercicio 3

1. Desarrolla en seudocódigo, en forma recursiva, un algoritmo para calcular la **potencia** de un número. El mismo ha de recibir como parámetros el número y el exponente (ver la ppt de clase sobre recursividad).
   - Identifica claramente el caso base y la sentencia que lo contempla.
   - ¿Puedes verificar que siempre el algoritmo progresará hacia el caso base?
2. Analiza el orden del tiempo de ejecución del algoritmo.
3. Implementa el algoritmo (en JAVA) y pruébalo
   - ¿Tu algoritmo soporta números reales o sólo enteros – para ambos parámetros
   - ¿qué sucede si uno, otro o ambos parámetros son negativos?

Solución

1. Seudocódigo:
   - El algoritmo siempre progresara al caso base porque en cada llamada recursiva n disminuye en uno (`p(x, n-1)`). Entonces, eventualmente el valor de n será 0.

```
CalcularPotencia(x, n)
si n = 0 // Caso base
    devolver 1
sino
    devolver x * p(x, n-1)
```

2. Calculo el orden de la siguiente manera

```
CalcularPotencia(x, n)
si n = 0 // O(1)
    devolver 1 O(1)
sino
    devolver x * p(x, n-1) // O(1)
```

   - sabemos que hace n llamadas recursivas porque el caso base es 1 y no 0
   - Orden del método → n * (Orden del bloque) → n * (1+1+1) → n * 1 → Orden(n)

3. Soporta solo numeros enteros para ambos parametros

   si x es negativo el resultado es positivo o negativo si n es par o impar respectivamente.
   si n es negativo tira la siguiente escepcion:

```
at com.example.ut2_pd2.Potencia.calcularPotencia(Potencia.java:8)
```

Ejercicio 4

1. Desarrolla en seudocódigo, en forma recursiva, un algoritmo para invertir los componentes de un vector pasado por parámetro, entre dos índices indicados también pasados como parámetros. (ver la ppt de clase sobre recursividad).
   • Identifica claramente el / los caso(s) base y la(s) sentencia(s) que lo contempla(n).
   • ¿Puedes verificar que siempre el algoritmo progresará hacia el caso base?
2. Analiza el orden del tiempo de ejecución del algoritmo.
3. Implementa el algoritmo (en JAVA) y pruébalo:
   - Crea un pequeño vector y prueba el algoritmo. Prueba situaciones de borde (extremos), parámetros fuera de rango, vector vacío, vector con sólo un elemento, etc.

Solución

1. Seudocódigo:
   - Siempre alcanzará el caso base porque i aumenta y j disminuye en cada llamada recursiva, por lo que en algún momento la condición i<j no se cumplirá, ya que i será mayor o igual a j, y se dejará de ejecutar la llamada recursiva.

```
InvertirArray(A, i, j)
  COM
    Si i<j then // Caso base es i >= j
      intercambiar A[i] con A[j]
      invertirArray(A,i+1,j-1)
    finsi
```

2. Calculo el orden de la siguiente manera

```
InvertirArray(A, i, j)
  COM
    Si i<j then  // Orden (1)
      intercambiar A[i] con A[j]  // Orden (1)
      invertirArray(A,i+1,j-1)    // Orden (1)
    finsi
```

   - sabemos que hace n/2 llamadas recursivas porque el caso base es i≥j
     - porque cuando haga n/2 llamadas → i= n/2 y j=n → caso base se cumple y no se ejecuta más recursivamente.
   - Orden del método → n/2 * (Orden del bloque) → n * (1+1+1) → n/2 * 1 → Orden(n/2)

3. **parametros fuera de rango:**

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3
at com.example.ut2_pd2.Vector.invertirArray(Vector.java:8)
at com.example.ut2_pd2.Main.main(Main.java:57)
```

vector vacío y c=0, j≠0:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0
```

vector vacío y c=0, j = 0:

```
cd /Users/jbevc/Desktop/ut4_ejDomiciliar
ios ; /usr/bin/env /Users/jbevc/Library/Application\ Support/Code/User/globalStorage/pleiades.ja
va-extension-pack-jdk/java/17/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/jbevc/
Desktop/ut4_ejDomiciliarios/demo/target/classes com.example.ut2_pd2.Main
```

vector con un solo elemento:

```
String[] v = {"papas"};
        Vector.invertirArray(v,0,0);
        for (String palabras : v) {
            System.out.println(palabras);
        }

// imprime papas
```

vector con un solo elemento y j>1:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 1 out of bounds for length 1
```