Práctica 2 Algorítmica:

Divide y Vencerás

Por:

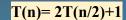
José A. Carmona Molina Nuria Manzano Mata Antonio Rodríguez Rodríguez

Algoritmo 1: MultiplicadorDyV

```
int multiplicador(int i, int j) {
    int suma = 0;
    if (j == 0 || i == 0) return 0;
    if (i == 1) return j;
    if (j == 1) return i;
    for (int n = 0; n < i; n++) {
        suma += j;
    }
    return suma;
}</pre>
```

```
int multiplicadorDyV(int i, int j) {
    if (j == 0 || i == 0) return 0;
    if (i == 1) return j;
    if (j == 1) return i;
    int aux, resultado=0, izquierda, derecha;
    int mitad= j / 2;
    if (i\%2 == 0) aux = 0;
    else aux = 1;
    izquierda = multiplicadorDyV(i, mitad);
    derecha = multiplicadorDyV(i, mitad + aux);
    resultado = izquierda + derecha;
    return resultado;
```

Cálculo de eficiencia teórica





$$T(2^m) = 2T(2^{m-1}) + 1$$



$$T(2^m) - 2T(2^{m-1}) = 1$$

Parte homogénea

$$T(2^{m}) - 2T(2^{m-1}) = 0$$

$$X^{m} - 2X^{m-1} = 0$$

$$(X^{m-1}) \cdot (X-2) = 0$$



Parte no homogénea

$$1 = b1 \cdot q1(m)$$



$$T(2^m) = \sum_{i=1}^r \sum_{j=0}^{M-1} C_{ij} Ri^m m^j = c_{10} 2^m + c_{20} 1^m$$



$$T(n) = c_{10} n + c_{20}$$



O(n)

Main y ejecuciones:

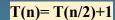
```
int main () {
    cout << "Prueba para algoritmo básico: " << endl;
    for (int i = 0; i < 5; i++) {
        for (int j = 5; j > 0; j--) {
            cout << "Resultado de la multiplicación de " << i << " - " << j << ": " << multiplicador(i, j) << endl;
    }
    }
    cout << endl << "Prueba para algoritmo DyV: " << endl;
    for (int i = 0; i < 5; i++) {
        for (int j = 5; j > 0; j--) {
            cout << "Resultado de la multiplicación de " << i << " - " << j << ": " << multiplicadorDyV(i, j) << endl;
    }
}
return 0;
}</pre>
```

```
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Documentos/Escritorio/Practica
Prueba para algoritmo básico:
Resultado de la multiplicación de 0 - 5: 0
Resultado de la multiplicación de 0 - 4: 0
Resultado de la multiplicación de 0 - 3: 0
Resultado de la multiplicación de 0 - 1: 0
Resultado de la multiplicación de 1 - 5: 5
Resultado de la multiplicación de 1 - 4: 4
Resultado de la multiplicación de 1 - 2: 2
Resultado de la multiplicación de 1 - 1: 1
Resultado de la multiplicación de 2 - 5: 10
Resultado de la multiplicación de 2 - 4: 8
Resultado de la multiplicación de 2 - 3: 6
Resultado de la multiplicación de 2 - 2: 4
Resultado de la multiplicación de 2 - 1: 2
Resultado de la multiplicación de 3 - 5: 15
Resultado de la multiplicación de 3 - 4: 12
Resultado de la multiplicación de 3 - 3: 9
Resultado de la multiplicación de 3 - 2: 6
Resultado de la multiplicación de 3 - 1: 3
Resultado de la multiplicación de 4 - 5: 20
Resultado de la multiplicación de 4 - 4: 16
Resultado de la multiplicación de 4 - 3: 12
Resultado de la multiplicación de 4 - 2: 8
Resultado de la multiplicación de 4 - 1: 4
Prueba para algoritmo DyV:
Resultado de la multiplicación de 0 - 5: 0
Resultado de la multiplicación de 0 - 4: 0
Resultado de la multiplicación de 0 - 3: 0
Resultado de la multiplicación de 0 - 2: 0
Resultado de la multiplicación de 0 - 1: 0
Resultado de la multiplicación de 1 - 5: 5
Resultado de la multiplicación de 1 - 4: 4
Resultado de la multiplicación de 1 - 3: 3
Resultado de la multiplicación de 1 - 2: 2
Resultado de la multiplicación de 1 - 1: 1
Resultado de la multiplicación de 2 - 5: 10
Resultado de la multiplicación de 2 - 4: 8
Resultado de la multiplicación de 2 - 3: 6
Resultado de la multiplicación de 2 - 2: 4
Resultado de la multiplicación de 2 - 1: 2
Resultado de la multiplicación de 3 - 5: 15
Resultado de la multiplicación de 3 - 4: 12
Resultado de la multiplicación de 3 - 3: 9
Resultado de la multiplicación de 3 - 2: 6
Resultado de la multiplicación de 3 - 1: 3
Resultado de la multiplicación de 4 - 5: 20
Resultado de la multiplicación de 4 - 4: 16
Resultado de la multiplicación de 4 - 3: 12
Resultado de la multiplicación de 4 - 2: 8
Resultado de la multiplicación de 4 - 1: 4
```

Algoritmo 2: CuadradoPerfecto

```
bool CuadradoPerfecto(int n) {
    bool cuadrado perfecto = false;
    if (n == 0 || n == 1) cuadrado perfecto = true;
        int valor = 0;
        for (int i = 2; valor <= n; i++) {
             valor = i*i;
             if (valor == n) cuadrado perfecto = true;
                                     bool CuadradoPerfectoDyV(const int n, int izquierda, int derecha)
    return cuadrado perfecto;
                                         if (n == 0 || n == 1) return true;
                                         int centro = (izquierda + derecha) / 2;
                                         int valor = centro*centro;
                                         if (valor == n) return true;
                                         if (izquierda == derecha) return false;
                                         if (valor > n)
                                             derecha = centro-1;
                                             izquierda = centro+1;
                                         return CuadradoPerfectoDyV(n, izquierda, derecha);
```

Cálculo de eficiencia teórica





$$T(2^m) = T(2^{m-1}) + 1$$



$$T(2^m) - T(2^{m-1}) = 1$$

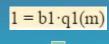
Parte homogénea

$$T(2^{m}) - T(2^{m-1}) = 0$$

$$X^{m} - X^{m-1} = 0$$

$$(X^{m-1}) \cdot (X-1) = 0$$

Parte no homogénea





$$T(2^m) = \sum_{i=1}^r \sum_{j=0}^{M-1} C_{ij} Ri^m m^j = c_{10} 1^m + c_{21} 1^m m$$



$$T(n) = c_{10} + c_{21} \log(n)$$



O(log(n))

Main y ejecuciones:

```
int main() {
   cout << "Ejecución para algoritmo básico: " << endl;
   for (int i = 0; i < 10; i++) {
      int valor = pow(i, 2);
      if (CuadradoPerfecto(valor)) cout << valor << ": es cuadrado perfecto" << endl;
      valor++;
      if (!CuadradoPerfecto(valor)) cout << valor << ": NO es cuadrado perfecto" << endl;
   }
   cout << endl;
   cout << "Ejecución para algoritmo DyV: " << endl;
   for (int i = 0; i < 10; i++) {
      int valor = pow(i, 2);
      if (CuadradoPerfectoDyV(valor, 0, valor)) cout << valor << ": es cuadrado perfecto" << endl;
      valor++;
      if (!CuadradoPerfectoDyV(valor, 0, valor)) cout << valor << ": NO es cuadrado perfecto" << endl;
   }
}</pre>
```

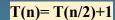
```
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Documentos/Escritori
Ejecución para algoritmo básico:
0: es cuadrado perfecto
1: es cuadrado perfecto
2: NO es cuadrado perfecto
4: es cuadrado perfecto
5: NO es cuadrado perfecto
9: es cuadrado perfecto
10: NO es cuadrado perfecto
16: es cuadrado perfecto
17: NO es cuadrado perfecto
25: es cuadrado perfecto
26: NO es cuadrado perfecto
36: es cuadrado perfecto
37: NO es cuadrado perfecto
49: es cuadrado perfecto
50: NO es cuadrado perfecto
64: es cuadrado perfecto
65: NO es cuadrado perfecto
81: es cuadrado perfecto
82: NO es cuadrado perfecto
Ejecución para algoritmo DyV:
0: es cuadrado perfecto
1: es cuadrado perfecto
2: NO es cuadrado perfecto
4: es cuadrado perfecto
5: NO es cuadrado perfecto
9: es cuadrado perfecto
10: NO es cuadrado perfecto
16: es cuadrado perfecto
17: NO es cuadrado perfecto
25: es cuadrado perfecto
26: NO es cuadrado perfecto
36: es cuadrado perfecto
37: NO es cuadrado perfecto
49: es cuadrado perfecto
50: NO es cuadrado perfecto
64: es cuadrado perfecto
65: NO es cuadrado perfecto
81: es cuadrado perfecto
82: NO es cuadrado perfecto
```

Algoritmo 3: Existe

bool existe(int n) {

```
if (n <= 5) return false;
bool encontrado = false;
int valor = 0:
for(int i = 0; valor <= n; i++) {
    valor = i*(i+1)*(i+2);
    if (valor == n) encontrado = true;
                                              bool existeDyV(const int n, int izquierda, int derecha) {
return encontrado;
                                                  if (n <= 5) return false;
                                                  int centro = (izquierda + derecha) / 2;
                                                  int valor = centro*(centro+1)*(centro+2);
                                                  if (valor == n) return true;
                                                  if (izquierda == derecha || izquierda > derecha) return false;
                                                  if (valor > n)
                                                     derecha = centro-1;
                                                      izquierda = centro+1;
                                                  return existeDyV(n, izquierda, derecha);
```

Cálculo de eficiencia teórica





$$T(2^m) = T(2^{m-1}) + 1$$



$$T(2^m) - T(2^{m-1}) = 1$$

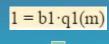
Parte homogénea

$$T(2^{m}) - T(2^{m-1}) = 0$$

$$X^{m} - X^{m-1} = 0$$

$$(X^{m-1}) \cdot (X-1) = 0$$

Parte no homogénea





$$T(2^m) = \sum_{i=1}^r \sum_{j=0}^{M-1} C_{ij} Ri^m m^j = c_{10} 1^m + c_{21} 1^m m$$



$$T(n) = c_{10} + c_{21} \log(n)$$



O(log(n))

Main y ejecuciones:

```
antonio@antonio-IdeaPad-Gaming-3-15ARH05:~/Documentos/E
Ejecución para algoritmo básico:
0: NO tiene 3 consecutivos que lo forman
6: tiene 3 consecutivos que lo forman
12: NO tiene 3 consecutivos que lo forman
18: NO tiene 3 consecutivos que lo forman
24: tiene 3 consecutivos que lo forman
30: NO tiene 3 consecutivos que lo forman
36: NO tiene 3 consecutivos que lo forman
42: NO tiene 3 consecutivos que lo forman
48: NO tiene 3 consecutivos que lo forman
54: NO tiene 3 consecutivos que lo forman
60: tiene 3 consecutivos que lo forman
66: NO tiene 3 consecutivos que lo forman
72: NO tiene 3 consecutivos que lo forman
78: NO tiene 3 consecutivos que lo forman
84: NO tiene 3 consecutivos que lo forman
90: NO tiene 3 consecutivos que lo forman
96: NO tiene 3 consecutivos que lo forman
102: NO tiene 3 consecutivos que lo forman
108: NO tiene 3 consecutivos que lo forman
114: NO tiene 3 consecutivos que lo forman
120: tiene 3 consecutivos que lo forman
126: NO tiene 3 consecutivos que lo forman
132: NO tiene 3 consecutivos que lo forman
138: NO tiene 3 consecutivos que lo forman
144: NO tiene 3 consecutivos que lo forman
150: NO tiene 3 consecutivos que lo forman
156: NO tiene 3 consecutivos que lo forman
162: NO tiene 3 consecutivos que lo forman
168: NO tiene 3 consecutivos que lo forman
174: NO tiene 3 consecutivos que lo forman
180: NO tiene 3 consecutivos que lo forman
186: NO tiene 3 consecutivos que lo forman
192: NO tiene 3 consecutivos que lo forman
198: NO tiene 3 consecutivos que lo forman
204: NO tiene 3 consecutivos que lo forman
210: tiene 3 consecutivos que lo forman
```

```
Eiecución para algoritmo DvV:
0: NO tiene 3 consecutivos que lo forman
6: tiene 3 consecutivos que lo forman
12: NO tiene 3 consecutivos que lo forman
18: NO tiene 3 consecutivos que lo forman
24: tiene 3 consecutivos que lo forman
30: NO tiene 3 consecutivos que lo forman
36: NO tiene 3 consecutivos que lo forman
42: NO tiene 3 consecutivos que lo forman
48: NO tiene 3 consecutivos que lo forman
54: NO tiene 3 consecutivos que lo forman
60: tiene 3 consecutivos que lo forman
66: NO tiene 3 consecutivos que lo forman
72: NO tiene 3 consecutivos que lo forman
78: NO tiene 3 consecutivos que lo forman
84: NO tiene 3 consecutivos que lo forman
90: NO tiene 3 consecutivos que lo forman
96: NO tiene 3 consecutivos que lo forman
102: NO tiene 3 consecutivos que lo forman
108: NO tiene 3 consecutivos que lo forman
114: NO tiene 3 consecutivos que lo forman
120: tiene 3 consecutivos que lo forman
126: NO tiene 3 consecutivos que lo forman
132: NO tiene 3 consecutivos que lo forman
138: NO tiene 3 consecutivos que lo forman
144: NO tiene 3 consecutivos que lo forman
150: NO tiene 3 consecutivos que lo forman
156: NO tiene 3 consecutivos que lo forman
162: NO tiene 3 consecutivos que lo forman
168: NO tiene 3 consecutivos que lo forman
174: NO tiene 3 consecutivos que lo forman
180: NO tiene 3 consecutivos que lo forman
186: NO tiene 3 consecutivos que lo forman
192: NO tiene 3 consecutivos que lo forman
198: NO tiene 3 consecutivos que lo forman
204: NO tiene 3 consecutivos que lo forman
210: tiene 3 consecutivos que lo forman
```

```
int main() {
   cout << "Ejecución para algoritmo básico: " << endl;
   for (int i = 0; i < 211; i+=6) {
      if (existe(i)) cout << i << ": tiene 3 consecutivos que lo forman" << endl;
      if (!existe(i)) cout << i << ": NO tiene 3 consecutivos que lo forman" << endl;
   }
   cout << endl << "Ejecución para algoritmo DyV: " << endl;
   for (int i = 0; i < 211; i+=6) {
      if (existeDyV(i,0,i)) cout << i << ": tiene 3 consecutivos que lo forman" << endl;
      if (!existeDyV(i,0,i)) cout << i << ": NO tiene 3 consecutivos que lo forman" << endl;
   }
}</pre>
```