Práctica 2:
Algoritmos Divide y Vencerás

Santiago Carbó García y Carlos García Segura

Universidad de Granada, España

1. Ejercicio 1

1.1. Diseño del algoritmo básico.

```
MultiplicaciónBasica(i : número entero, j: número entero)
  Si j == 1, hacer: # Caso base de la recurrencia
        resultado = i
  En otro caso: # Caso General de la recurrencia
    resultado = MultiplicaciónBasica(i , j-1) + i

    Devolver resultado
```

1.2. Análisis y diseño de componentes del algoritmo Divide y Vencerás.

División del problema en subproblemas: El problema se puede dividir en K=1 subproblemas (técnica de simplificación), de tamaño "n/2". El algoritmo funciona dividiendo 'j' entre 2 cada vez que llamamos recursivamente a la función. El subproblema es de la misma naturaleza que el original (realizar la multiplicación mediante sumas) y, al ser un único subproblema, es independiente, tiene el mismo tamaño y se puede resolver por separado.

Existencia de caso base: El caso base se da cuando 'j' es '1', caso en el que la multiplicación seria i*1 lo que da como resultado 'i', siendo esto lo que devuelve la función.

Combinación de soluciones: Una vez obtenida la solución del subproblema, si la división de 'j' entre 2 tiene resto '0', entonces simplemente se sumará la solución consigo misma, en otro caso, a la suma anteriormente mencionada se le sumará 'i'.

1.3. Diseño del algoritmo Divide y Vencerás y de la función de combinación.

```
MultiplicaciónDyV(i : número entero, j: número entero)
  Si j == 1, hacer: # Caso base de la recurrencia
        resultado = i
  En otro caso: # Caso General de la recurrencia
    Si j % 2 == 0, hacer:
        exacto = true
    aux = MultiplicaciónDyV(i , (j/2)-1) + i
    Si exacto == true, hacer:
        resultado = aux + aux
```

En otro caso:

resultado = aux + aux + i

Devolver resultado

1.4. Análisis de la eficiencia de los métodos básico y Divide y Vencerás.

Primero vamos a analizar la eficiencia del método básico.

El primer paso es obtener una Recurrencia Lineal No Homogénea, la cuál sería la siguiente:

$$T(n) = T(n-1) + 1$$

Despejamos, y quedaría así:

$$T(n) - T(n-1) = 1$$

Resolvemos la parte homogénea:

$$T(n) - T(n-1) = 0$$

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

$$x^{m-1}(x-1) = 0$$

Obteniendo la parte homogénea del polinomio característico: $P_H(x) = (x-1)$

Para resolver la parte no homogénea, debemos conseguir un escalar b_1 y un polinomio q_m :

$1 = {b_1}^m q_1(m)^{d_1}$

donde podemos ver que $b_1 = 1$ y $q_1(m) = 1$, donde el grado del polinomio es $d_1 = 0$

El polinomio característico se obtiene como: $P(x) = P_H(x)(x - b_1)^{d_1+1} = (x-1)(x-1) = (x-1)^2$

Tenemos $r = 1$ (ya que solo tenemos una raíz diferente), con valor $R_1 = 1$ y multiplicidad $M_1 = 2$

Si aplicamos la fórmula de la ecuación característica, tenemos que el tiempo de ejecución es el siguiente:

$T(n) = \sum_{i=1}^{m} \sum_{j=0}^{M_i - 1} c_{ij} {R_i}^m m^j = c_{10}1^n + c_{11}1^n n$

Tenemos la siguiente recurrencia:

$T(n) = c_{10}1^n + c_{11} * n * 1^n$

Aplicando la **regla de la suma**, es decir, eligiendo el máximo, obtenemos que el orden de eficiencia del algoritmo en el peor de los casos es $O(n)$.

Vamos a analizar el método Divide y Vencerás

La recurrencia es de la siguiente manera:

$T(n) = T(n/2) + 1$

Como la ecuación no tiene las variables de la forma requerida por la ecuación característica, tenemos que hacer el siguiente cambio de variable:

$n = 2^m$; $m = log_2 n$

De forma que la ecuación quedaría así:

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

Despejamos, y quedaría así:

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

Resolvemos la parte homogénea:

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

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

$x^{m-1}(x - 1) = 0$

Obteniendo la parte homogénea del polinomio característico: $P_H(x) = (x - 1)$

Para resolver la parte no homogénea, debemos conseguir un escalar b_1 y un polinomio q_m :

$1 = b_1{}^m q_1(m)^{d_1}$

donde podemos ver que $b_1 = 1$ y $q_1(m) = 1$, donde el grado del polinomio es $d_1 = 0$

El polinomio característico se obtiene como: $P(x) = P_H(x)(x - b_1)^{d_1+1} = (x - 1)(x - 1) = (x - 1)^2$

Tenemos $r = 1$ (ya que solo tenemos una raíz diferente), con valor $R_1 = 1$ y multiplicidad $M_1 = 2$

Si aplicamos la fórmula de la ecuación característica, tenemos que el tiempo de ejecución (expresado en términos de 2^m) es el siguiente:

$T(2^m) = \sum_{i=1}^{m} \sum_{j=0}^{M_i-1} c_{ij} R_i{}^m m^j = c_{10} 1^m + c_{11} 1^m m$

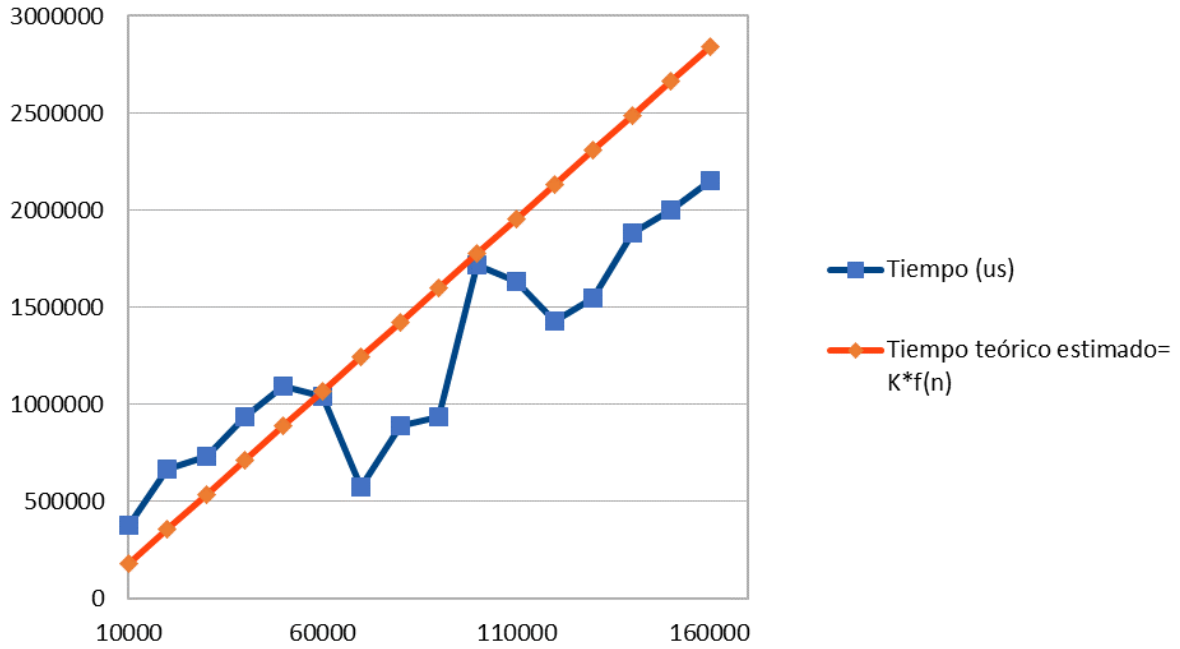
Deshacemos el cambio de variable:

$T(n) = c_{10} + c_{11} log_2(n)$

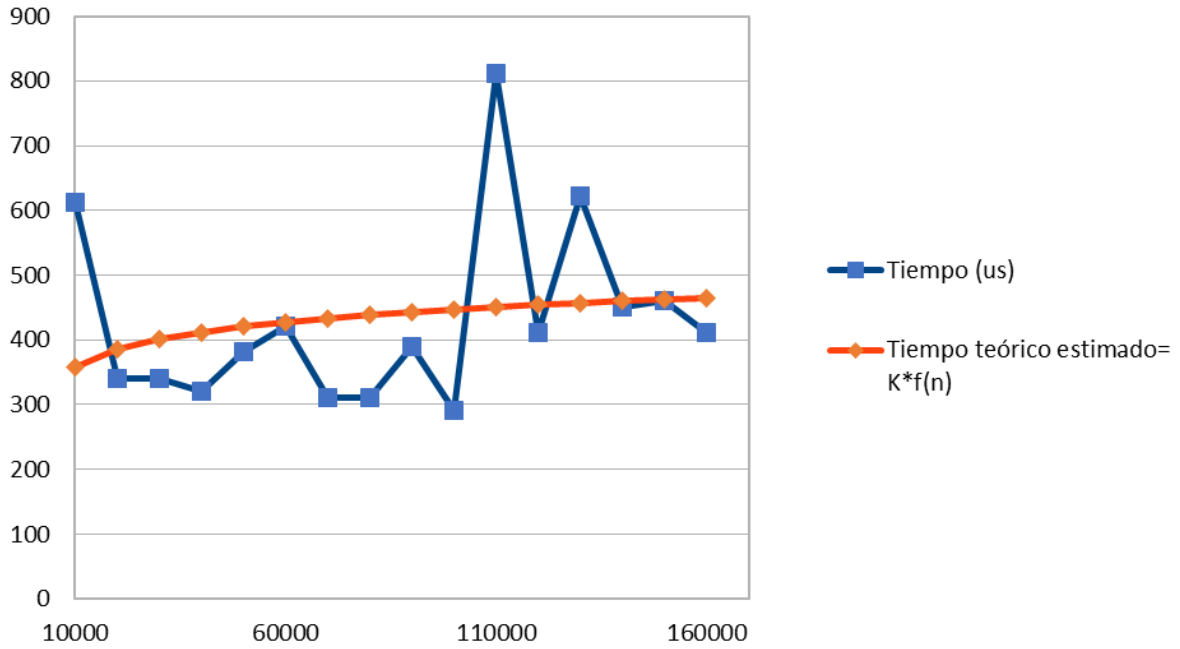
Aplicando la **regla de la suma**, es decir, eligiendo el máximo, obtenemos que el orden de eficiencia del algoritmo en el peor de los casos es $O(log(n))$.

En cuanto a la eficiencia empírica hemos realizado una prueba de ejecución en la que hemos aumentado el tamaño del caso en intervalos de 10000 en 15 ocasiones para comprobar el rendimiento usando ambos métodos. Hemos obtenido los siguientes resultados:

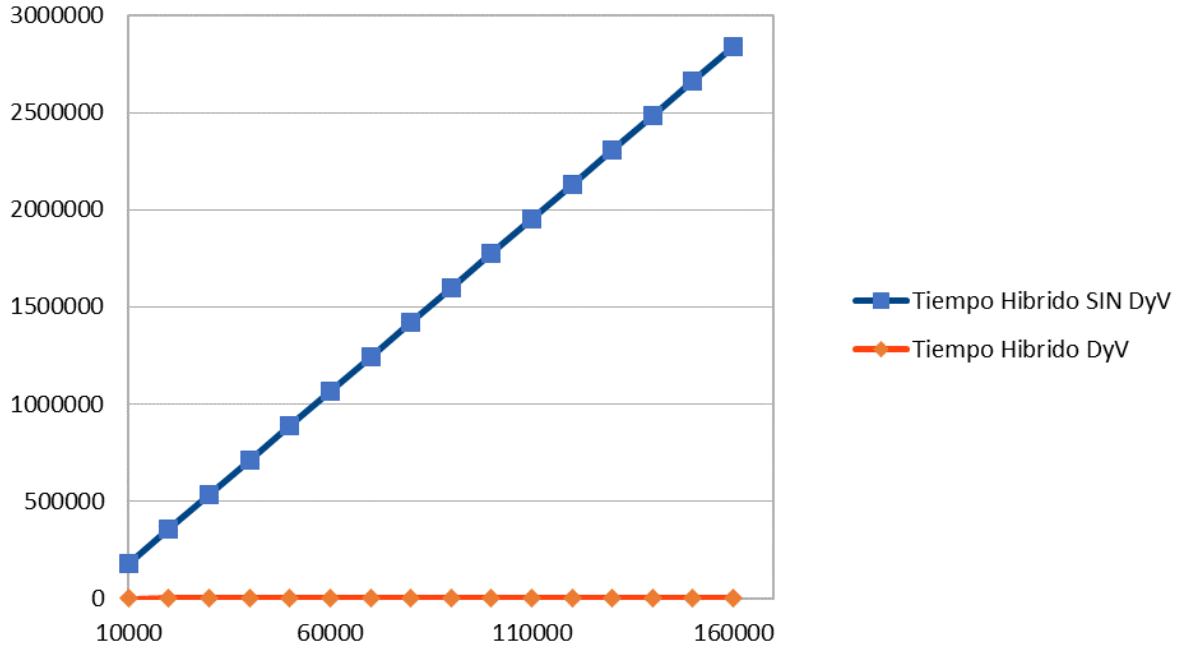
Eficiencia Empírica e Híbrida del método básico:



Eficiencia Empírica e Híbrida del método Divide y Vencerás:



Comparación entre ambos métodos:



Como podemos observar en la gráfica, la eficiencia obtenida con el método Divide y Vencerás es muy superior a la eficiencia obtenida sin Divide y Vencerás.

1.5. Implementación de los métodos básico y Divide y Vencerás.

Metodo basico

```
int ejercicio1FuerzaBruta(int i, int j)
{
    int result = 0;

    if (j == 1)
    {
        result = i;
    }
    else if (j > 1)
    {
        result = ejercicio1FuerzaBruta(i, j - 1) + i;
    }

    return result;
}
```

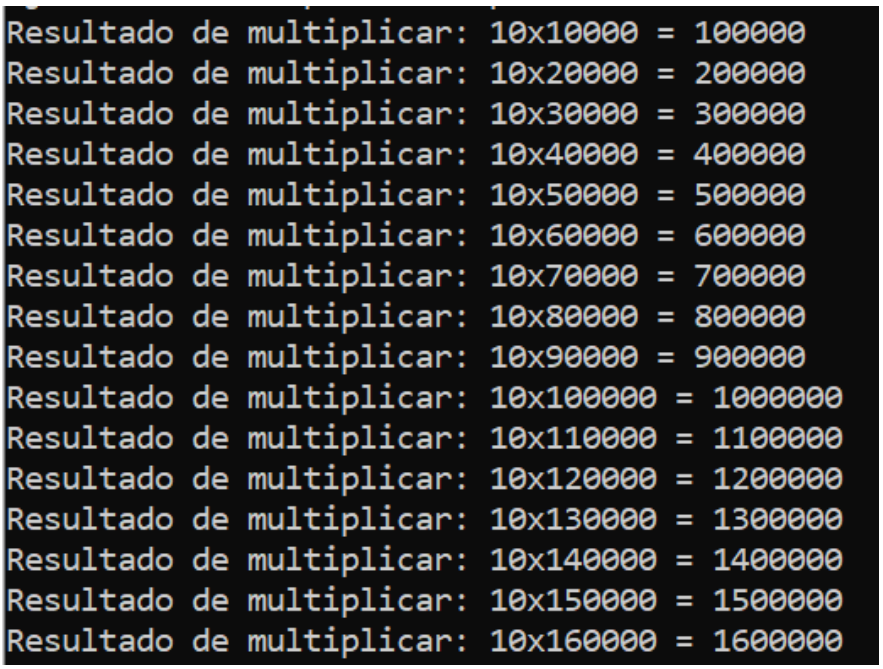
Pruebas de ejecución

```
Resultado de multiplicar: 10x10000 = 100000
Resultado de multiplicar: 10x20000 = 200000
Resultado de multiplicar: 10x30000 = 300000
Resultado de multiplicar: 10x40000 = 400000
Resultado de multiplicar: 10x50000 = 500000
Resultado de multiplicar: 10x60000 = 600000
Resultado de multiplicar: 10x70000 = 700000
Resultado de multiplicar: 10x80000 = 800000
Resultado de multiplicar: 10x90000 = 900000
Resultado de multiplicar: 10x100000 = 1000000
Resultado de multiplicar: 10x110000 = 1100000
Resultado de multiplicar: 10x120000 = 1200000
Resultado de multiplicar: 10x130000 = 1300000
Resultado de multiplicar: 10x140000 = 1400000
Resultado de multiplicar: 10x150000 = 1500000
Resultado de multiplicar: 10x160000 = 1600000
```

Método Divide y Vencerás

```
int ejercicio1DyV(int i, int j)
{
    int result_aux = 0;
    int result = 0;
    bool exacto = false;

    if (j == 1)
    {
        result = i;
    }
    else if (j > 1)
    {

        if (j % 2 == 0)
        {
            exacto = true;
        }

        result_aux = ejercicio1DyV(i, (j / 2) - 1) + i;
        if (exacto)
            result = result_aux + result_aux;
        else
            result = result_aux + result_aux + i;

    }

    return result;
}
```

Pruebas de ejecución



2. Ejercicio 2

2.1. Diseño del algoritmo básico

```
CuadradoPerfectoBasico(n : número entero)
  resultado = -1
  Si n == 0, hacer:
            resultado = 0
 En otro caso Si n == 1, hacer:
            resultado = 1


  En otro caso:
     x = 0
    Para i = 0 hasta n, hacer:
            x = i*i
            Si x == n, hacer:
                resultado = i

  Devolver resultado
```

2.2. Análisis y diseño de componentes del algoritmo

División del problema en subproblemas: se puede dividir en K=1 subproblemas (técnica de simplificación), de tamaño “n/2”. El algoritmo empieza calculando la mitad del número del cual queremos saber si es cuadrado perfecto. A continuación, lo multiplicamos por sí mismo y comprobamos si sobrepasa el número o se queda corto y, en función de ellos, se utiliza un subproblema que va desde 0 hasta la mitad o un subproblema que va desde mitad+1 hasta el final. El subproblema es de la misma naturaleza que el original (buscar el cuadrado perfecto entre un rango de números), y, al ser un único subproblema, es independiente, tiene el mismo tamaño y se puede resolver por separado.

Existencia de caso base: existen dos casos base dentro de este algoritmo:

- Puede que “cini” sea mayor que “cfin”, de forma que nunca se entra en el caso general.
- También es posible que “medio” sea justamente el cuadrado perfecto, de manera que “medio * medio == n” y tampoco entramos en el caso general.

Combinación de soluciones: no es necesario combinar las soluciones, ya que solo tenemos K=1 subproblemas (simplificación).

2.3. Diseño del algoritmo Divide y Vencerás y de la función de combinación

Para el diseño del algoritmo se ha hecho una derivación de la Búsqueda Binaria, con el objetivo de calcular el cuadrado perfecto. Para ello, se calcula la mitad de “n” y se multiplica por sí mismo: si es más grande que “n”, entonces seguimos la búsqueda en la mitad superior del número y si es más pequeño la proseguimos en la mitad inferior.

El algoritmo se ve así escrito en pseudocódigo:

```
CuadradoPerfectoDyV(n : número entero, cini : número entero, cfin : número entero)
  medio = (cini + cfin) / 2
  Si cini > cfin , hacer:
        Devolver -1
  Si (medio * medio)  == n, hacer:
        Devolver medio
  En otro caso, si  (medio * medio)  > n, hacer:
        Devolver CuadradoPerfectoDyV(n, cini, medio - 1)
  En otro caso, hacer:
        Devolver CuadradoPerfectoDyV(n, medio + 1, cfin)
```

2.4. Análisis de eficiencia de los métodos básico y Divide y Vencerás

Primero vamos a analizar la eficiencia del método básico. Calculamos de dentro hacia afuera.

En primer lugar tenemos el bucle “for”: dentro de este tenemos una asignación (O(1)) y un condicional “if” con una asignación (max(O(1),O(1))=O(1)). El bucle se ejecuta “n de veces”, de forma que obtenemos n * O(1) = O(n). El bucle for está dentro de un condicional “else”, los otros dos condicionales “if” e “else if” tienen la evaluación y una asignación (max(O(1),O(1))=O(1)), y nos queda una asignación que es O(1) y devolver la variable O(1). De forma que finalmente: max(O(1),O(1),O(1),O(1), O(n))=O(n). Tenemos una eficiencia O(n).

Vamos a analizar el método Divide y Vencerás
La recurrencia es de la siguiente manera:

$$T(n) = T(n/2) + 1$$

Como la ecuación no tiene las variables de la forma requerida por la ecuación característica, tenemos que hacer el siguiente cambio de variable:

$$n = 2^m \; ; \; m = log_2 n$$

De forma que la ecuación quedaría así:

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

Despejamos, y quedaría así:

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

Resolvemos la parte homogénea:

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

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

$$x^{m-1}(x - 1) = 0$$

Obteniendo la parte homogénea del polinomio característico: $P_H(x) = (x - 1)$

Para resolver la parte no homogénea, debemos conseguir un escalar b_1 y un polinomio q_m :

$$1 = b_1{}^m q_1(m)^{d_1}$$

donde podemos ver que $b_1 = 1$ y $q_1(m) = 1$, donde el grado del polinomio es $d_1 = 0$

El polinomio característico se obtiene como: $P(x) = P_H(x)(x - b_1)^{d_1+1} = (x - 1)(x - 1) = (x - 1)^2$

Tenemos $r = 1$ (ya que solo tenemos una raíz diferente), con valor $R_1 = 1$ y multiplicidad $M_1 = 2$

Si aplicamos la fórmula de la ecuación característica, tenemos que el tiempo de ejecución (expresado en términos de 2^m) es el siguiente:

$T(2^m) = \sum_{i=1}^{m} \sum_{j=0}^{M_i-1} c_{ij} R_i^m m^j = c_{10} 1^m + c_{11} 1^m m$

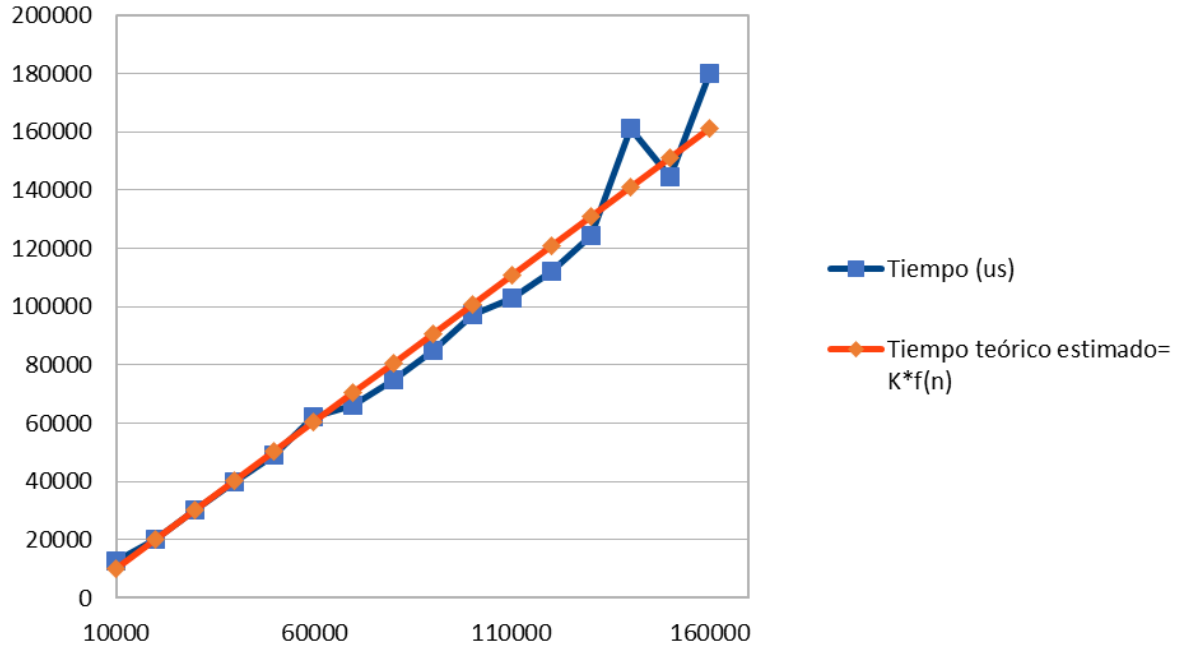
Deshacemos el cambio de variable:

$T(n) = c_{10} + c_{11} log_2(n)$

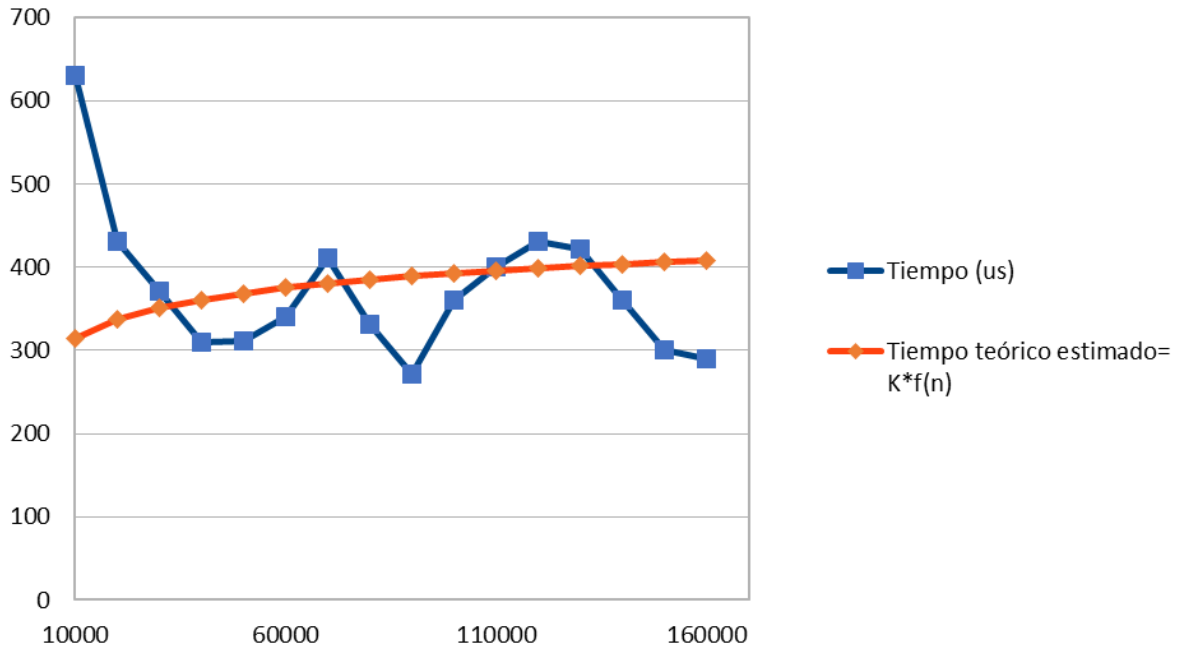
Aplicando la **regla de la suma**, es decir, eligiendo el máximo, obtenemos que el orden de eficiencia del algoritmo en el peor de los casos es $O(log(n))$.

En cuanto a la eficiencia empírica, hemos realizado una prueba de ejecución en la que hemos aumentado el tamaño del caso en intervalos de 10000 en 15 ocasiones para comprobar el rendimiento usando ambos métodos. Hemos obtenido los siguientes resultados:

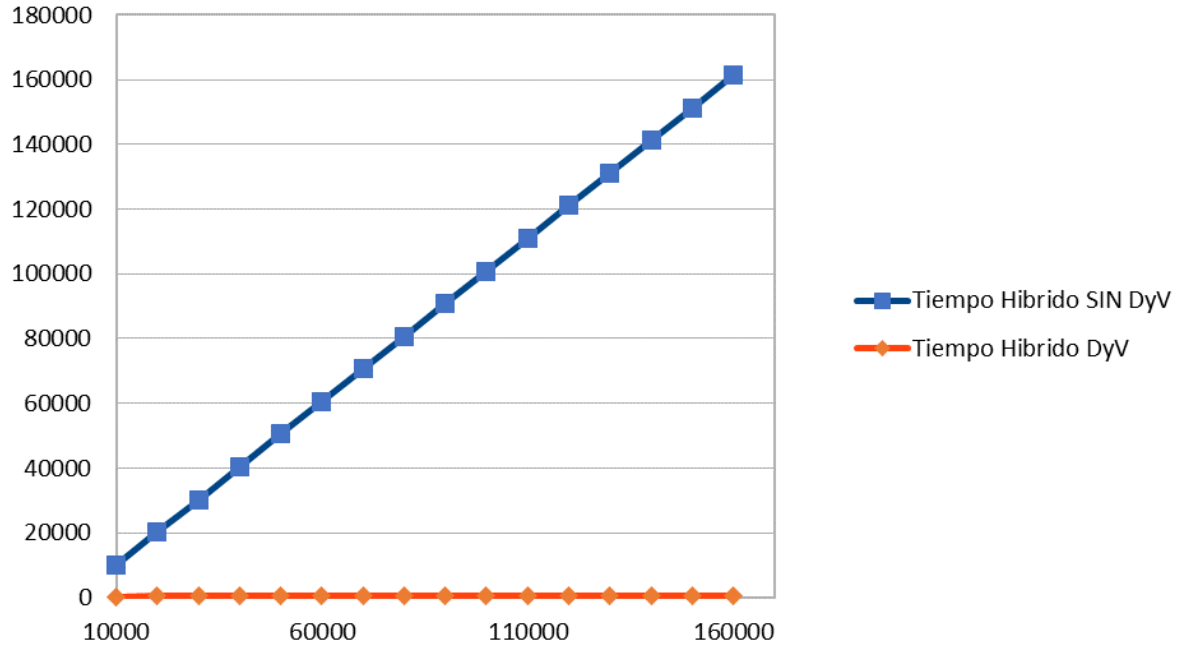
Eficiencia Empírica e Híbrida del método básico:



Eficiencia Empírica e Híbrida del método Divide y Vencerás:



Comparación entre ambos métodos:



2.5. Implementación de los métodos básico y Divide y Vencerás

Metodo Basico

```
int ejercicio2FuerzaBruta(int n)
{
    int result = -1;

    if (n == 0)
    {
        result = 0;
    }
    else if (n == 1)
    {
        result = 1;
    }
    else
    {
        int x = 0;

        for (int i = 0; i <= n;i++)
        {
            x = i * i;

            if (x == n)
                result = i;
        }
    }

    return result;
}
```

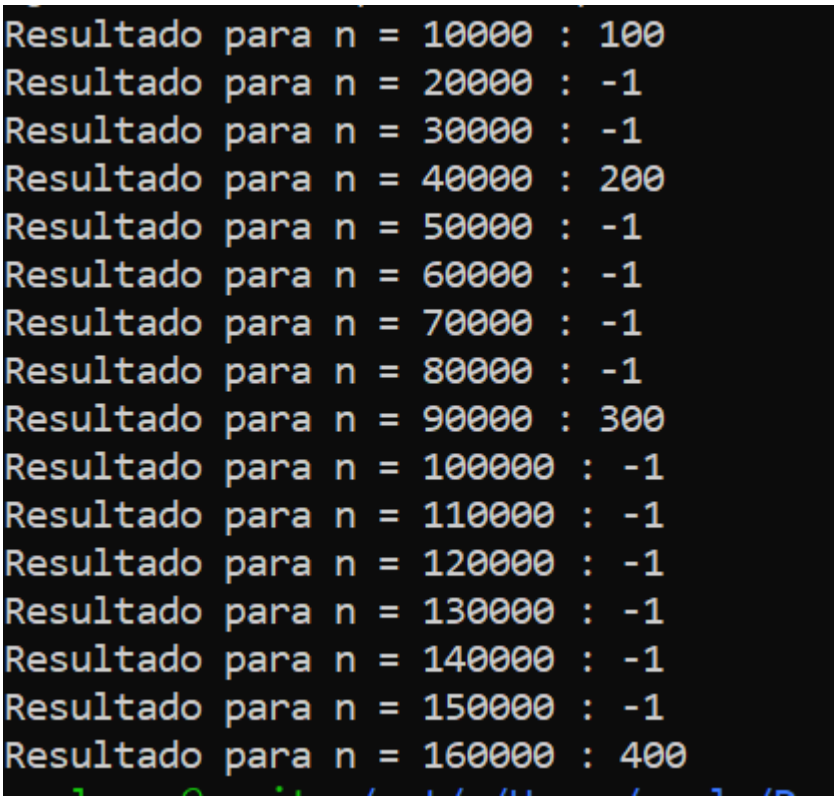
Pruebas de ejecución:

```
Resultado para n = 10000 : 100
Resultado para n = 20000 : -1
Resultado para n = 30000 : -1
Resultado para n = 40000 : 200
Resultado para n = 50000 : -1
Resultado para n = 60000 : -1
Resultado para n = 70000 : -1
Resultado para n = 80000 : -1
Resultado para n = 90000 : 300
Resultado para n = 100000 : -1
Resultado para n = 110000 : -1
Resultado para n = 120000 : -1
Resultado para n = 130000 : -1
Resultado para n = 140000 : -1
Resultado para n = 150000 : -1
Resultado para n = 160000 : 400
```

Método Divide y Vencerás

```
int ejercicio2DyV(int n, int cini, int cfin){
    int medio = (cini + cfin)/2;

    if (cini > cfin) {
        return -1;
    }

    if (medio * medio == n) {
        return medio;
    }
    else if (medio * medio > n) {
        return ejercicio2DyV(n, cini, medio - 1);
    }
    else {
        return ejercicio2DyV(n, medio + 1, cfin);
    }
}
```

Pruebas de ejecución:



3. Ejercicio 3

3.1. Diseño del algoritmo básico

```
Ejercicio3Basico(n : número entero)
  resultado = -1
  Si n == 0, hacer:
          resultado = 0
  En otro caso:
    x = 0
   Para i = 0 hasta n, hacer:
          x = i*(i+1)*(i+2)
          Si x == n, hacer:
              resultado = i

  Devolver resultado
```

3.2. Análisis y diseño de componentes del algoritmo

División del problema en subproblemas: se puede dividir en K=1 subproblemas (técnica de simplificación), de tamaño “n/2”. El algoritmo empieza calculando la mitad del número del cual queremos saber si es cuadrado perfecto. A continuación, lo multiplicamos por sí mismo y comprobamos si sobrepasa el número o se queda corto y, en función de ellos, se utiliza un subproblema que va desde 0 hasta la mitad o un subproblema que va desde mitad+1 hasta el final. El subproblema es de la misma naturaleza que el original (buscar el cuadrado perfecto entre un rango de números), y, al ser un único subproblema, es independiente, tiene el mismo tamaño y se puede resolver por separado.

Existencia de caso base: existen dos casos base dentro de este algoritmo:

- Puede que “cini” sea mayor que “cfin”, de forma que nunca se entra en el caso general.
- También es posible que “medio” sea justamente el cuadrado perfecto, de manera que “medio * medio == n” y tampoco entramos en el caso general

Combinación de soluciones: no es necesario combinar las soluciones, ya que solo tenemos K=1 subproblemas (simplificación).

3.3. Diseño del algoritmo Divide y Vencerás y de la función de combinación

Para el diseño del algoritmo se ha hecho una derivación de la Búsqueda Binaria, con el objetivo de calcular el cuadrado perfecto. Para ello, se calcula la mitad de “n” y se multiplica por sí mismo: si es más grande que “n”, entonces seguimos la búsqueda en la mitad superior del número y si es más pequeño la proseguimos en la mitad inferior.

El algoritmo se ve así escrito en pseudocódigo:

```
Ejercicio3DyV(n : número entero, cini : número entero, cfin : número entero)
  medio = (cini + cfin) / 2
  Si cini > cfin , hacer:
        Devolver -1
  Si (medio * (medio + 1) * (medio + 2))  == n, hacer:
        Devolver medio
  En otro caso, si  (medio * (medio + 1) * (medio + 2))  > n, hacer:
        Devolver Ejercicio3DyV(n, cini, medio - 1)
  En otro caso, hacer:
        Devolver Ejercicio3DyV(n, medio + 1, cfin)
```

3.4. Análisis de eficiencia de los métodos básico y Divide y Vencerás

Primero vamos a analizar la eficiencia del método básico. Calculamos de dentro hacia afuera.

En primer lugar tenemos el bucle “for”: dentro de este tenemos una asignación (O(1)) y un condicional “if” con una asignación (max(O(1),O(1))=O(1)). El bucle se ejecuta “n de veces”, de forma que obtenemos n * O(1) = O(n). El bucle for está dentro de un condicional “else”, el otro condicional “if” tiene la evaluación y una asignación (max(O(1),O(1))=O(1)), y nos queda una asignación que es O(1) y devolver la variable O(1). De forma que finalmente: max(O(1),O(1),O(1) O(1), O(n))=O(n). Tenemos una eficiencia O(n).

Vamos a analizar el método Divide y Vencerás
La recurrencia es de la siguiente manera:
$T(n) = T(n/2) + 1$

Como la ecuación no tiene las variables de la forma requerida por la ecuación característica, tenemos que hacer el siguiente cambio de variable:

$$n = 2^m \; ; \; m = log_2 n$$

De forma que la ecuación quedaría así:

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

Despejamos, y quedaría así:

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

Resolvemos la parte homogénea:

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

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

$$x^{m-1}(x - 1) = 0$$

Obteniendo la parte homogénea del polinomio característico: $P_H(x) = (x - 1)$

Para resolver la parte no homogénea, debemos conseguir un escalar b_1 y un polinomio q_m :

$$1 = b_1{}^m q_1(m)^{d_1}$$

donde podemos ver que $b_1 = 1$ y $q_1(m) = 1$, donde el grado del polinomio es $d_1 = 0$

El polinomio característico se obtiene como: $P(x) = P_H(x)(x - b_1)^{d_1+1} = (x - 1)(x - 1) = (x - 1)^2$

Tenemos $r = 1$ (ya que solo tenemos una raíz diferente), con valor $R_1 = 1$ y multiplicidad $M_1 = 2$

Si aplicamos la fórmula de la ecuación característica, tenemos que el tiempo de ejecución (expresado en términos de 2^m) es el siguiente:

$$T(2^m) = \sum_{i=1}^{m} \sum_{j=0}^{M_i - 1} c_{ij} R_i{}^m m^j = c_{10} 1^m + c_{11} 1^m m$$

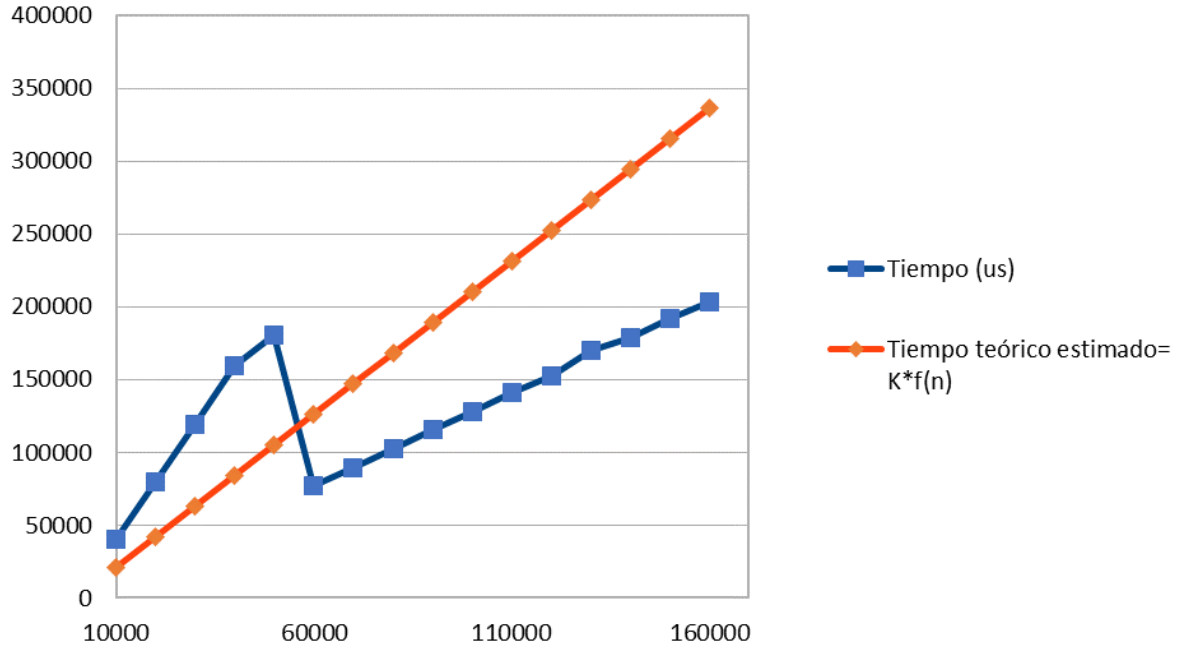
Deshacemos el cambio de variable:

$$T(n) = c_{10} + c_{11} log_2(n)$$

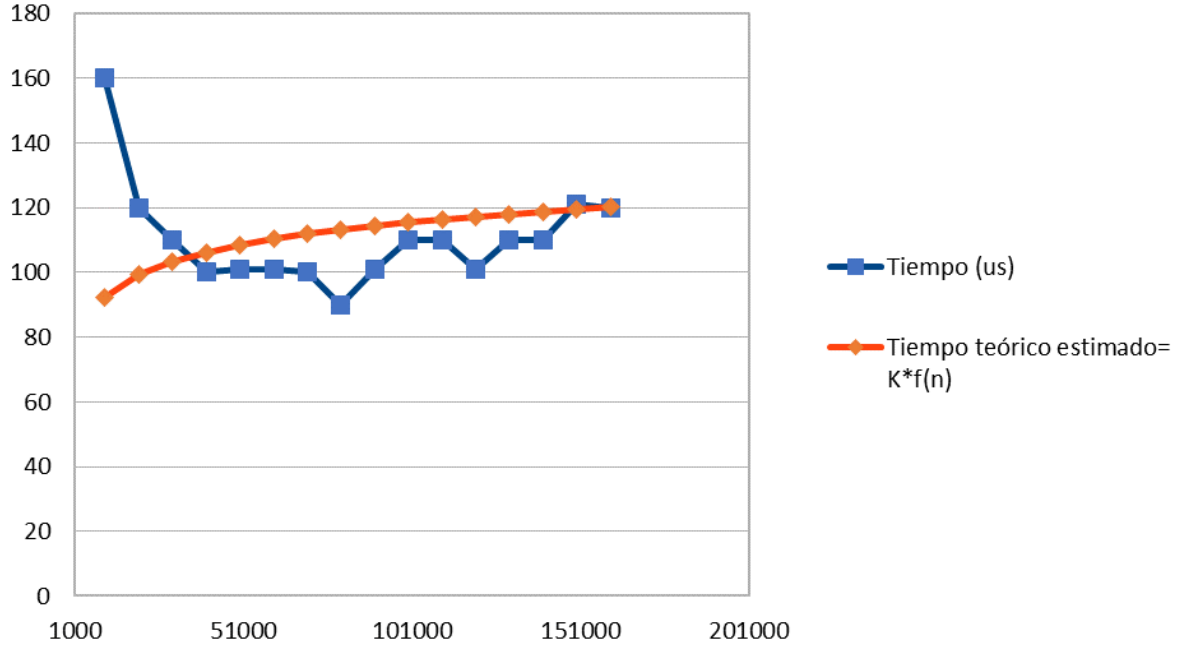
Aplicando la **regla de la suma**, es decir, eligiendo el máximo, obtenemos que el orden de eficiencia del algoritmo en el peor de los casos es $O(log(n))$.

En cuanto a la eficiencia empírica, hemos realizado una prueba de ejecución en la que hemos aumentado el tamaño del caso en intervalos de 10000 en 15 ocasiones para comprobar el rendimiento usando ambos métodos. Hemos obtenido los siguientes resultados:

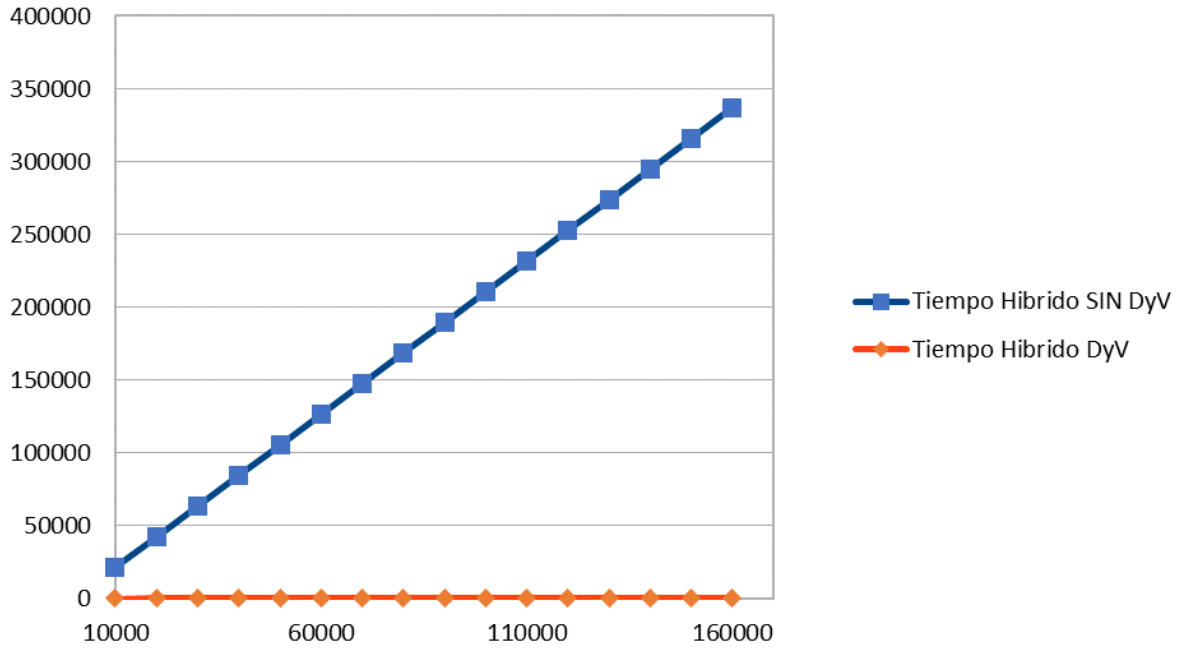
Eficiencia Empírica e Híbrida del método básico:



Eficiencia Empírica e Híbrida del método Divide y Vencerás:



Comparación entre ambos métodos:



3.5. Implementación de los métodos básico y Divide y Vencerás

Metodo Basico

```
int ejercicio3FuerzaBruta(int n)
{
    int result = -1;

    if (n == 0)
    {
        result = 0;
    }
    else
    {
        int x = 0;

        for (int i = 0; i <= n; i++)
        {
            x = i * (i + 1) * (i + 2);

            if (x == n)
                result = i;
        }
    }

    return result;
}
```

Pruebas de ejecución:

```
Resultado para n = 10000 : -1
Resultado para n = 20000 : -1
Resultado para n = 30000 : -1
Resultado para n = 40000 : -1
Resultado para n = 50000 : -1
Resultado para n = 60000 : -1
Resultado para n = 70000 : -1
Resultado para n = 80000 : -1
Resultado para n = 90000 : -1
Resultado para n = 100000 : -1
Resultado para n = 110000 : -1
Resultado para n = 120000 : -1
Resultado para n = 130000 : -1
Resultado para n = 140000 : -1
Resultado para n = 150000 : -1
Resultado para n = 160000 : -1
```

Método Divide y Vencerás

```
int ejercicio3DyV(long int n, long int cini, long int cfin)
{
    long int medio = (cini + cfin) / 2;

    if (cini > cfin)
    {
        return -1;
    }

    if (medio * (medio + 1) * (medio + 2) == n)
    {
        return medio;
    }
    else if (medio * (medio + 1) * (medio + 2) > n)
    {
        return ejercicio3DyV(n, cini, medio - 1);
    }
    else
    {
        return ejercicio3DyV(n, medio + 1, cfin);
    }
}
```

Pruebas de ejecución: