INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

----------ANÁLISIS DE ALGORITMOS----------

ACTIVIDAD

Análisis de casos

PROFESOR:

Franco Martínez Edgardo Adrián

ALUMNO:

Meza Vargas Brandon David – 2020630288

GRUPO:

3CM13

ÍNDICE

Código 1

```
func sumaCuadratica3Mayores(A, n){

    if(A[1] > A[2] && A[1] > A[3]){
        m1 = A[1];
        if(A[2] > A[3]){
            m2 = a[2];
            m3 = A[3];
        }else{
            m2 = A[3];
            m3 = A[2];
        }else if(A[2] > A[1] && A[2] > A[3]){
            m1 = A[2];
            if(A[1] > A[3]){
                m2 = A[1];
                m3 = A[3];
            }else{
                m2 = A[3];
                m3 = A[1];
            }
        }else{
            m1 = A[3];
            if(A[1] > A[2]){
                m2 = A[1];
                m3 = A[2];
            }else{
                m2 = A[2];
                m3 = A[1];
            }
        }
    }
    i = 4;
    while(i <= n){
        if(A[i] > m1){
            m3 = m2;
            m2 = m1;
            m1 = A[i];
        }else if(A[i] > m2){
            m3 = m2;
            m2 = A[i];
        }else if(A[i] > m3)
            m3 = A[i];

            i = i +1;
    }
    return = pow(m1 + m2 +m3, 2);
```

Imagen 1. Código 1.

Funciones de complejidad temporal

Para este problema, las operaciones básicas que se están considerando son:

- Comparaciones entre elementos del arreglo A
- Asignaciones a m1, m2, m3

Una vez sabiendo las operaciones básicas a considerar, las identificamos en el código de la imagen 1

```
if(A[1] > A[2] && A[1] > A[3]){ ---------> 2 comparaciones
    m1 = A[1];                  ---------> 1 asignacion
    if(A[2] > A[3]){            ---------> 1 comparacion
        m2 = a[2];              ---------> 1 asignacion
        m3 = A[3];              ---------> 1 asignacion
    }else{
        m2 = A[3];              ---------> 1 asignacion
        m3 = A[2];              ---------> 1 asignacion
    }else if(A[2] > A[1] && A[2] > A[3]){    ---------> 2 comparaciones
        m1 = A[2];              ---------> 1 asignacion
        if(A[1] > A[3]){        ---------> 1 comparacion
            m2 = A[1];          ---------> 1 asignacion
            m3 = A[3];          ---------> 1 asignacion
        }else{
            m2 = A[3];          ---------> 1 asignacion
            m3 = A[1];          ---------> 1 asignacion
        }
    }else{
        m1 = A[3];              ---------> 1 asignacion
        if(A[1] > A[2]){        ---------> 1 comparacion
            m2 = A[1];          ---------> 1 asignacion
            m3 = A[2];          ---------> 1 asigancion
        }else{
            m2 = A[2];          ---------> 1 asignacion
            m3 = A[1];          ---------> 1 asignacion
        }
    }
}
i = 4;      You, 2 days ago • analisis de casos
while(i <= n){
    if(A[i] > m1){              ---------> n-3 comparaciones
        m3 = m2;                ---------> n-3 asignaciones
        m2 = m1;                ---------> n-3 asignaciones
        m1 = A[i];              ---------> n-3 asignaciones
    }else if(A[i] > m2){        ---------> n-3 comparaciones
        m3 = m2;                ---------> n-3 asignaciones
        m2 = A[i];              ---------> n-3 asignaciones
    }else if(A[i] > m3)
        m3 = A[i];              ---------> n-3 asignaciones

        i = i +1;
}
```

Imagen 2. Operaciones del código 1.

Mejor caso

Este mejor caso ocurre cuando los 3 números mayores están al inicio del arreglo A, pero el primer mayor es el que esta en la primera posición, de esta forma solo se hace el primer if del inicio, haciéndose 6 operaciones. Posteriormente en el while no se cumple ninguna condición, sin embargo, se hacen las comparaciones, así tenemos que se hacen 3(n-3) comparaciones, siendo la función de complejidad temporal para el mejor caso:

$$f_t = 6 + 3(n-3) = 6 + 3n - 9 = 3n - 3$$

Peor caso

Este peor caso se da cuando los números dentro del arreglo están ordenados de manera ascendente, por lo que al inicio ninguna comparación se cumple, realizándose 8 operaciones, posteriormente en el ciclo, se cumple la primera condición hasta que se comparan todos los números del arreglo, haciendo 4(n-3) comparaciones, siendo la función de complejidad temporal para el peor caso:

$$f_t = 8 + 4(n-3) = 8 + 4n - 12 = 4n - 4$$

Caso medio

Para el caso medio, tenemos que considerar todos los caminos que se pueden presentar en lo general, estos son 8:

C1: el primer camino que se presenta es cuando es el mejor caso: **3n-3**

C2: el segundo camino sucede cuando se cumple el primer if y, además, dentro del ciclo de igual forma se cumple el primer if (A[i] > m1): **4n-6**

C3: el tercer camino sucede cuando se cumple el primer if y, además dentro del ciclo, se cumple el segundo if todas las veces (A[i] > m2): **4n-6**

C4: el cuarto camino pasa cuando se cumple el primer if y, además, se cumple todas las veces el tercer if dentro del ciclo (A[i] > m3): **4n-6**

Cuando al inicio se cumple que el primer mayor es el de la segunda posición y cuando se cumple que el primer mayor es el de la tercera posición, tenemos que el costo es de 8 en ambos casos, por esta razón se tiene un solo camino en esta parte, pero en el ciclo tenemos otros 4 similares a los 4 primeros mencionados anteriormente.

C5: sucede cuando el primer mayor se encuentra en la segunda o tercera posición del arreglo y, además, dentro del ciclo no se cumple ninguna condición: **3n-1**

C6: sucede cuando el primer mayor se encuentra en la segunda o tercera posición del arreglo y, además, dentro del ciclo se cumple el primer if todas las veces: **4n-4**

C7: sucede cuando el primer mayor se encuentra en la segunda o tercera posición del arreglo y, además, dentro del ciclo se cumple el segundo if: **4n-4**

C8: sucede cuando el primer mayor se encuentra en la segunda o tercera posición del arreglo y, además, dentro del ciclo se cumple el último if: **4n-4**

De esta forma, si cada caso tiene la misma probabilidad de ocurrencia en promedio se harán:

$$f_t = \frac{1}{8}\big((3n-3)+3(4n-6)+(3n-1)+3(4n-4)\big) = \frac{30n-34}{8}$$

Gráfica comparativa

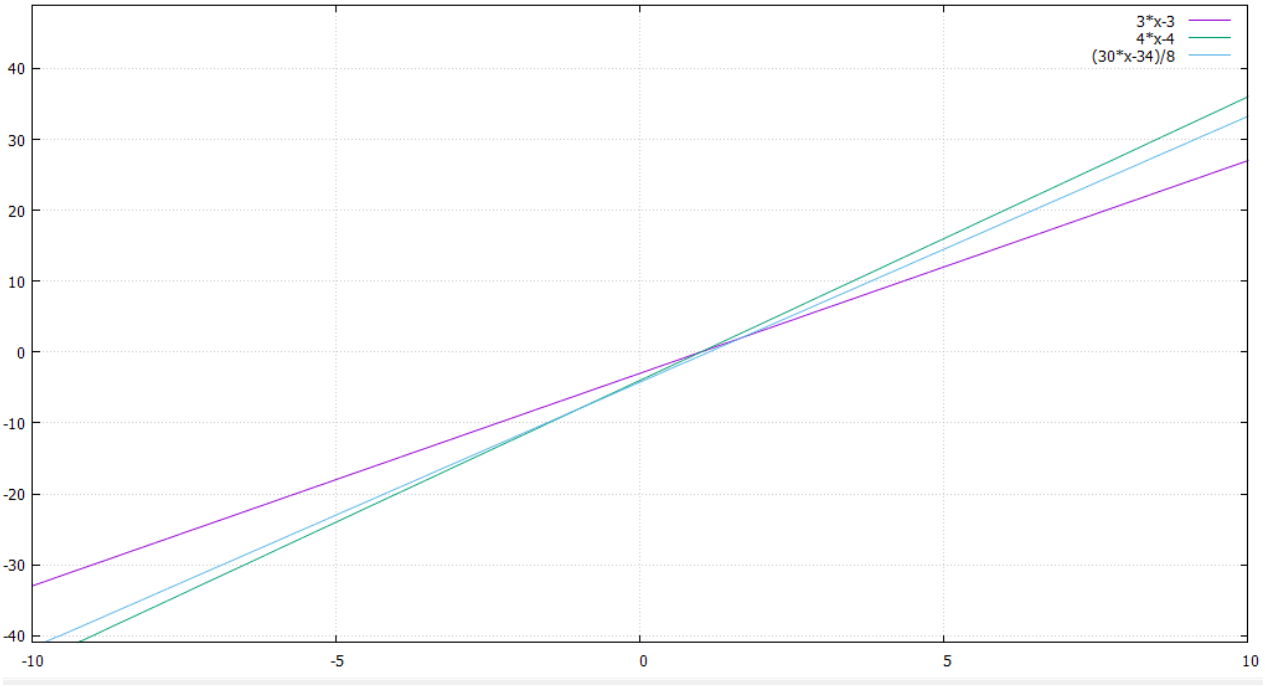

Imagen 3. Grafica de casos del código 1.

PEOR CASO

MEJOR CASO

CASO MEDIO

Código 2

```
func maximoComunDivisor(m, n){
    a = max(n, m);
    b = min(n, m);
    residuo = 1;

    mientras(residuo > 0){
        residuo = a mod b;
        a = b;
        b = residuo;
    }
    maximoComunDivisor = a;

    return maximoComunDivisor;
```

Imagen 4. Código 2.

Funciones de complejidad temporal

Para este problema, las operaciones básicas que se están considerando son:

- Operación módulo de a y b

Una vez sabiendo las operaciones básicas a considerar, las identificamos en el código de la imagen 4

```
mientras(residuo > 0){
    residuo = a mod b; ----------> 1 operacion mod
    a = b;
    b = residuo;
}
maximoComunDivisor = a;
```

Imagen 6. Operaciones código 2.

Mejor caso

El mejor caso se dará cuando en la primera entrada al ciclo, el residuo sea 0, es decir, el mejor caso es:

$$f_t = 1$$

Peor caso

El peor caso, si tomamos en cuenta que n y m son números consecutivos de la serie de Fibonacci, nos damos cuenta de que las veces que entrará al ciclo while es la cantidad de números anteriores a m de la serie de Fibonacci, ejemplo:

Los primeros números de la serie de Fibonacci son: 1, 1, 2, 3, 5, 8, 13, 21, 34

Si hacemos n = 34 y m = 21, las veces que se hará la operación modulo serán 7, de esta forma:

$$f_t = cantidad\ de\ números\ de\ la\ serie\ de\ Fibonacci\ anteriores\ a\ m$$

Para graficarla, diremos que n = cantidad de números de la serie de Fibonacci anteriores a m

Caso medio

Para el caso medio, si el peor y mejor caso son igual de probables, tendremos:

$$f_t = \frac{1}{2}(n+1) = \frac{n+1}{2}$$

Gráfica comparativa

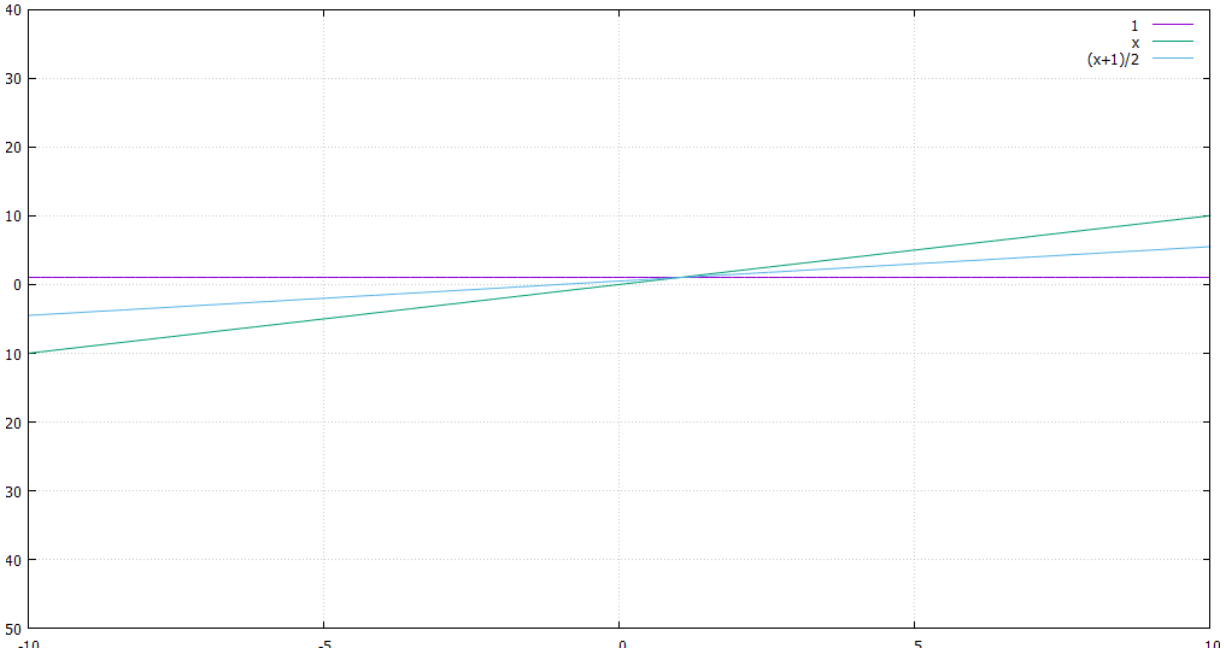

Imagen 7. Gráfica de casos del código 2.

PEOR CASO

MEJOR CASO

CASO MEDIO

Código 3

```
func sumaCuadratica3MayoresV2(A, n){
    for(i=0;i<3;i++)
        for(j=0;j<n-1-i;j++)
            if(A[j]>A[j+1]){
                aux = A[j];
                A[j] = A[j+1];
                A[j+1] = aux;
            }

    r = A[n-1] + A[n-2] + A[n-3];
    return pow(r, 2);
```

Imagen 8. Código 3

Funciones de complejidad temporal

Para este problema, las operaciones básicas que se están considerando son:

- Comparaciones entre elementos del arreglo A
- Asignaciones al arreglo

Una vez sabiendo las operaciones básicas a considerar, las identificamos en el código de la imagen 8

```
func sumaCuadratica3MayoresV2(A, n){
    for(i=0;i<3;i++)
        for(j=0;j<n-1-i;j++)
            if(A[j]>A[j+1]){          ---------> 1 comparacion
                aux = A[j];
                A[j] = A[j+1];        ---------> 1 asignacion
                A[j+1] = aux;         ---------> 1 asignacion
            }

    r = A[n-1] + A[n-2] + A[n-3];      You, 2 days ago • analisis
    return pow(r, 2);
}
```

Imagen 10. Operaciones código 3.

Mejor caso

Este mejor caso se da cuando los 3 primeros números del arreglo se encuentran en orden, pero para determinar la función tenemos que analizar los for que se encuentran en el algoritmo, el primer for se hace 3 veces, el segundo se hace n-1-i, si los números se encuentran en orden solo se hará la comparación del if, de esta forma con los fors se hacen (3)(n-1-i)(1) operaciones, de esta forma, la función de complejidad temporal del mejor caso será:

$$f_t = 3\sum_{i=0}^{2}(n-1-i) = 9n-18$$

Peor caso

El peor caso, sucede cuando los elementos están ordenados de forma ascendente, de esta forma, haciendo el mismo análisis de los for que en el mejor caso, pero ahora aumentando las comparaciones dentro del if tenemos que se hacen = (3)(n-1-i)(3) operaciones, quedando la función de complejidad de la siguiente forma:

$$f_t = 9\sum_{i=0}^{2}(n-1-i) = 27n-54$$

Caso medio

En este caso podemos distinguir varios caminos que sigue el algoritmo, siendo los primeros dos caminos identificados los correspondientes al mejor y peor caso.

Sin embargo, identifique dos más, uno donde el if se cumple solo una vez:

(1)(n-1-i)(3)+(2)(n-1-i)(1) – la primera parte corresponde cuando se cumple el if en la primera iteración del primer for y la segunda parte es cuando no se cumple el if en las siguientes iteraciones:

$$3\sum_{i=0}^{0}(n-1-i) + 2\sum_{i=1}^{2}(n-1-i) = 3n-3+4n-10 = 7n-13$$

El segundo camino es aquel en donde el if se cumple dos veces:

(2)(n-1-i)(3)+(1)(n-1-i)(1) – la primera parte corresponde cuando se cumple el if dos veces, y la segunda cuando no se cumple en las iteraciones sobrantes:

$$6\sum_{i=0}^{1}(n-1-i) + \sum_{i=2}^{2}(n-1-i) = 12n-18+n-3 = 13n-21$$

De esta forma, considerando que los 4 caminos tienen la misma probabilidad, la función para el caso medio quedaría:

$$f_t = \frac{1}{4}\big((9n-18)+(27n-54)+(7n-13)+(13n-21)\big) = \left(\frac{56n-106}{4}\right)$$

Gráfica comparativa

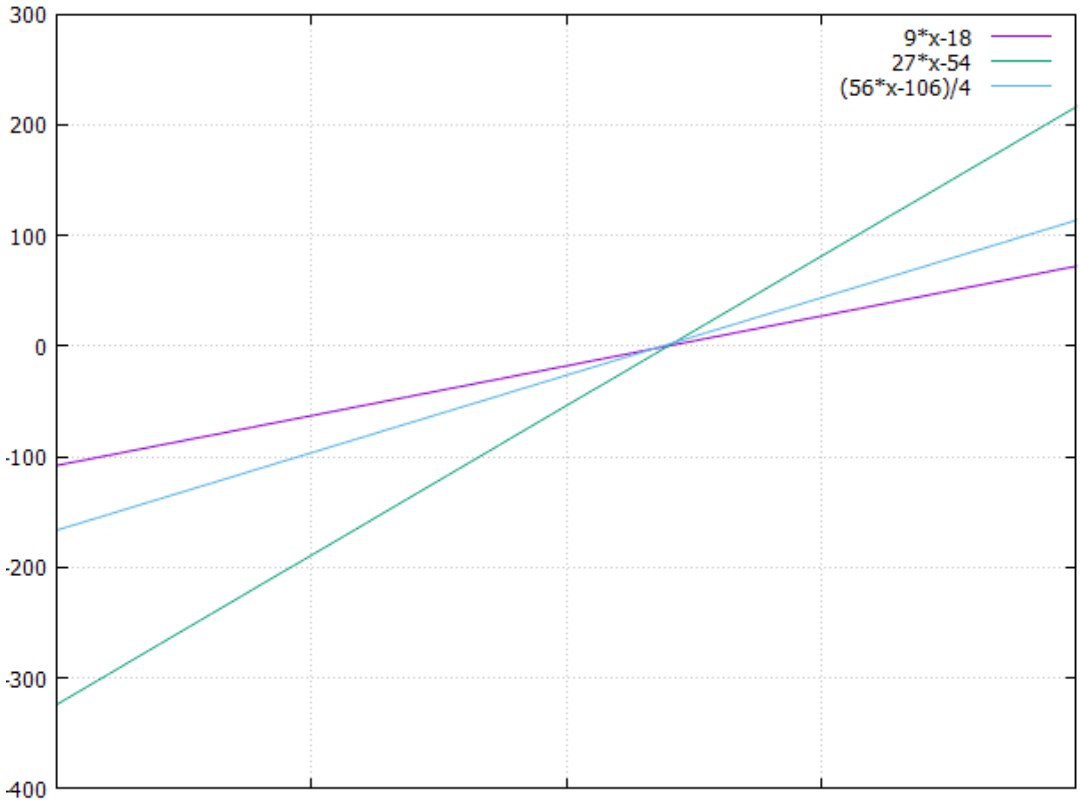

Imagen 11. Gráfica de casos del código 3.

PEOR CASO

MEJOR CASO

CASO MEDIO

Código 4

```
Algoritmo frecuenciaMinNumeros
    Leer nanDimension A[n]
    i = 1
    Mientras i <= n
        Leer A[i]
        i = i+1
    FinMientras

    f = 0
    i = 1
    Mientras i <= n
        ntemp = A[i]
        j = 1
        ftemp = 0
        Mientras j <= n
            si ntemp = A[j]
                ftemp = ftemp +1
            FinSi
            j = j +1
        FinMientras

        si f < ftemp
            f = ftemp
            num=ntemp
        FinSi

        i = i + 1
    FinMientras
    Escribir num
goritmo
```

Imagen 12. Código 4

Funciones de complejidad temporal

Para este problema, las operaciones básicas que se están considerando son:

- Comparaciones y asignaciones de las variables f, ntemp, ftemp y num

Una vez sabiendo las operaciones básicas a considerar, las identificamos en el código de la imagen 12

```
Algoritmo frecuenciaMinNumeros      You, 2 days ago • analisis de casos
    Leer nanDimension A[n]
    i = 1
    Mientras i <= n
        Leer A[i]
        i = i+1
    FinMientras

    f = 0                                   ---------> 1 asignacion
    i = 1
    Mientras i <= n
        ntemp = A[i]                        ---------> 1 asignacion
        j = 1
        ftemp = 0                           ---------> 1 asignacion
        Mientras j <= n
            si ntemp = A[j]                 ---------> 1 comparacion
                ftemp = ftemp +1            ---------> 2 asignacion + operacion
            FinSi
            j = j +1
        FinMientras

        si f < ftemp                        ---------> 1 comparacion
            f = ftemp                       ---------> 1 asignacion
            num=ntemp                       ---------> 1 asignacion
        FinSi

        i = i + 1
    FinMientras
```

Imagen 13. Operaciones código 4.

Al analizar el pseudocódigo de la imagen 13, nos damos cuenta de que gracias a las primeras líneas, en el arreglo se almacenan números desde 1 hasta n, en las posiciones 1 hasta n, debido a esto al entrar en el segundo for, no importa el tamaño n del arreglo, la comparación ntemp = A[j] solo se hace una sola vez, de igual forma, en la última comparación f < ftemp, de igual forma solo se hace una vez.

De esta forma solo existirá un camino para este algoritmo, dicho esto, la función de complejidad temporal para el mejor, peor y caso medio siempre será el mismo:

$$f_t = 1 + n\Big(2 + n\big(3 + (n-1)\big) + 3 + (n-1)\Big) = 1 + n(n^2 + 3n + 4)$$
$$= n^3 + 3n^2 + 4n + 1$$

Gráfica comparativa

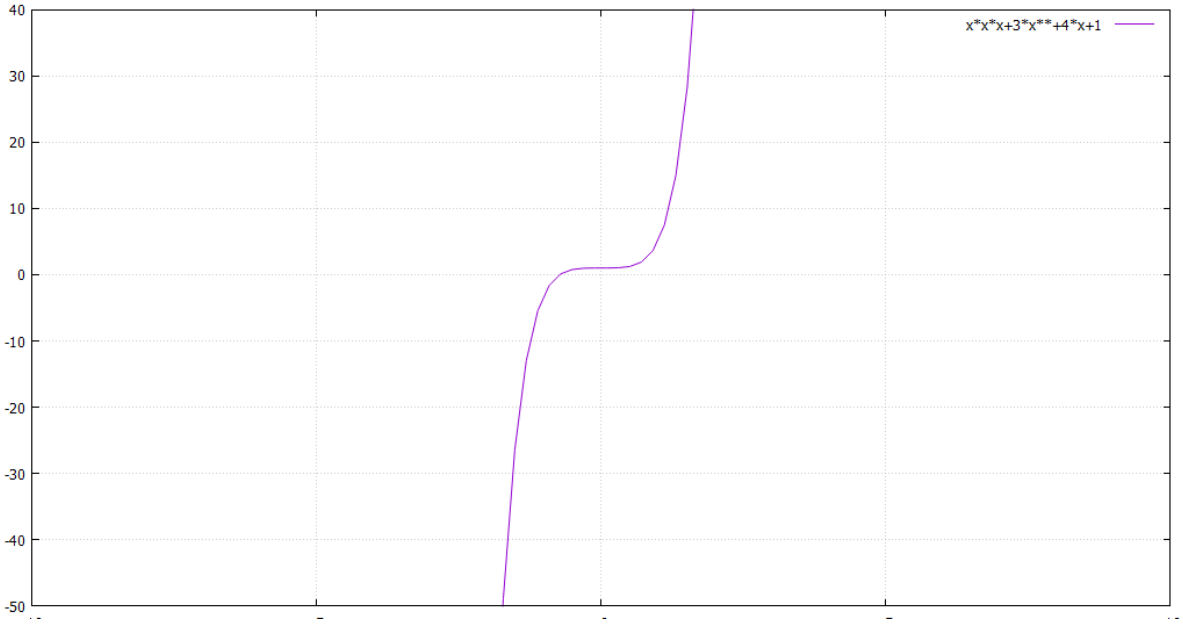

Imagen 14. Gráfica casos del código 4.

Código 5

```
polinomio = 0;
for(i=0;i<=n;i++)
    polinomio = polinomio * z + A[n-i];
```

Imagen 15. Código 5

Funciones de complejidad temporal

Para este problema, las operaciones básicas que se están considerando son:

- Operaciones con la variable polinomio

Si analizamos el código, vemos que es algo ambiguo, pues solo sabemos que hace operaciones sobre la variable polinomio, pero no tenemos alguna condición que nos pueda indicar caminos para un mejor o peor caso, lo único que tenemos es un ciclo for que se hace n+1 veces

Mejor caso

El mejor caso con lo que tenemos a la mano sería que solo se haga la asignación inicial y que el for no se cumpla debido a una n negativa, de esta forma, la función temporal de complejidad del mejor caso sería:

$$f_t = 1$$

Peor caso

El peor caso se da al entrar al for, este se hará n+1 veces, dentro del for hay tres operaciones con la variable polinomio (asignación + 2 aritméticas), la suma no esta directamente relacionada con la variable polinomio, pero se tomará en cuenta, de esta forma la función complejidad para el peor caso sería:

$$f_t = 1 + (n+1)(3) = 1 + 3n + 3 = 3n + 4$$

Caso medio

Para el caso medio, siendo que ambos casos anteriores tienen la misma probabilidad de suceder, tenemos:

$$f_t = \frac{1}{2}(1 + 3n + 4) = \frac{3n + 5}{2}$$

Gráfica comparativa

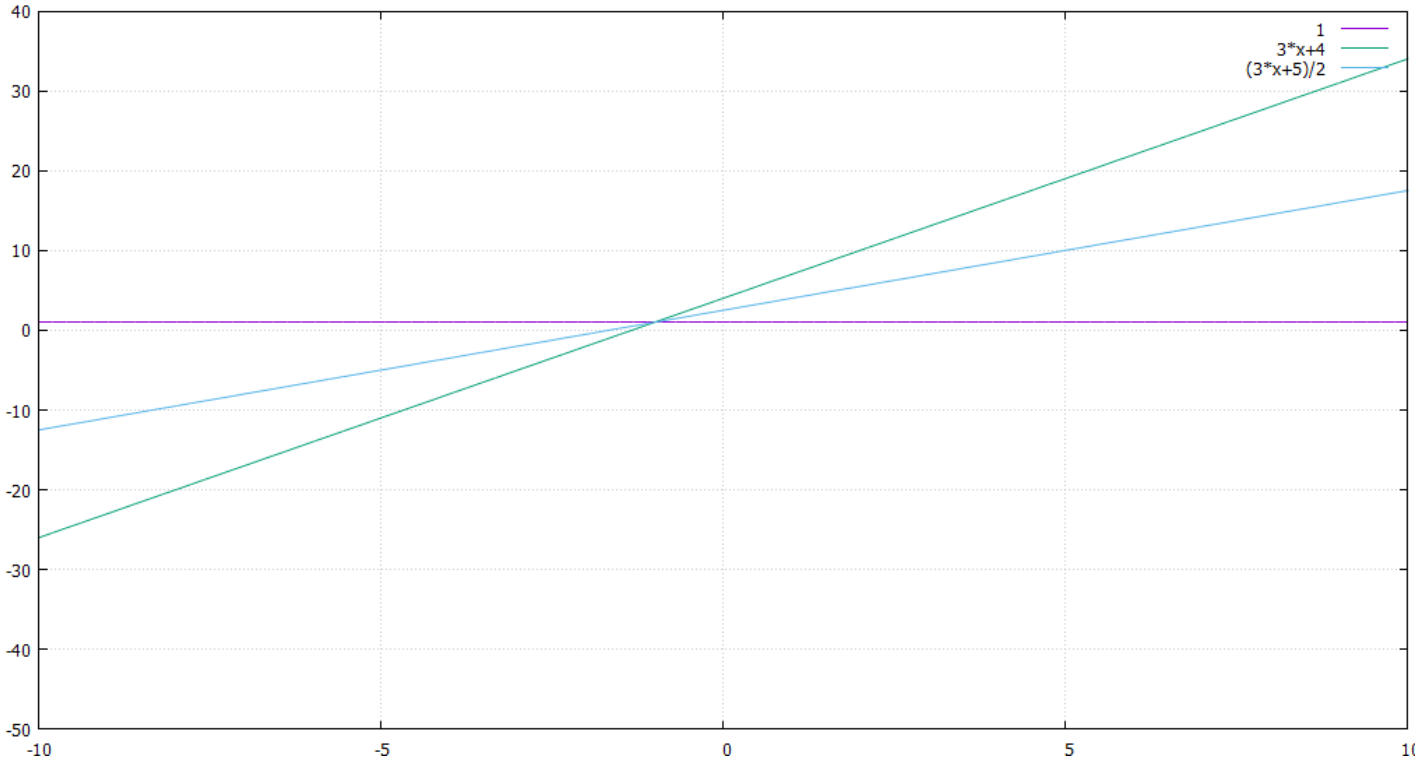

Imagen 16. Gráfica de casos del código 5.

PEOR CASO

MEJOR CASO

CASO MEDIO