Instituto Politécnico Nacional
Escuela Superior de Computo

Ejercicio 04

"Análisis de Casos"

Mora Ayala José Antonio

Análisis de Algoritmos

Instrucciones

Para los siguientes algoritmos determine las funciones de complejidad temporal, para el mejor caso, peor caso y caso medio de cada algoritmo. Indique cual(es) son las condiciones (instancia de entrada) del peor caso y cual(es) la del mejor caso en términos del problema que resuelven.

Señale las operaciones básicas que considerará para el análisis de cada algoritmo (Si no logra distinguirlas considere todas las asignaciones, aritméticas y condicionales)

CÓDIGO 1

```
void c1CasoMedio(){
    scanf("%d", &n);
    A = malloc(sizeof(int)*n);
    contador = 0;

    for (j = 0; j < n; j++){

        A[j]= rand()%(32000);
    }

    if (A[1] > A[2] && A[1] > A[3]){

        contador++;

        m1 = A[1];      contador++;

        if (A[2] > A[3]){

            contador++;

            m2 = A[2];  contador++;
            m3 = A[3];  contador++;
        }
        else{

            contador++;

            m2 = A[3];  contador++;
            m3 = A[2];  contador++;
        }
    }
    else if (A[2] > A[1] && A[2] > A[3]){

        contador+=2;

        m1 = A[2];      contador++;

        if (A[1] > A[3]){

            contador++;

            m2 = A[1];  contador++;
            m3 = A[3];  contador++;
        }
        else{

            contador++;

            m2 = A[3];  contador++;
            m3 = A[1];  contador++;
        }
    }
    else{
```

```
        contador+=2;

        m1 = A[3];  contador++;

        if (A[1] > A[2]){
                contador++;

            m2 = A[1];  contador++;
            m3 = A[2];  contador++;
        }
        else{

            contador++;

            m2 = A[2];  contador++;
            m3 = A[1];  contador++;
        }
    }

    i = 4;

    while( i <= n ){

        if (A[i] > m1){

            contador++;

            m3 = m2;    contador++;
            m2 = m1;    contador++;
            m1 = A[i];  contador++;
        }
        else if (A[i] > m2){

            contador+=2;

            m3 = m2;    contador++;
            m2 = A[i];  contador++;
        }
        else if (A[i] > m3){

            contador+=3;
            m3 = A[i];  contador++;
        }
        else{

            contador+=3;
        }

        i++;
    }

    promedio+=contador;
}
```

SELECCIÓN DE OPERACIONES BÁSICAS

Como se ha visto durante la clase la selección de operaciones básicas se han definido en relación a que tanto se ven afectadas por el tamaño del problema, contabilizando solo aquellas que son necesarias para el mismo, por lo que usaremos aquellas que emplean el arreglo de tamaño n, estas son presentadas a continuación mediante comentarios en el código anterior

```
void c1CasoMedio(){
    scanf("%d", &n);
    A = malloc(sizeof(int)*n);
    contador = 0;
    for (j = 0; j < n; j++){
        A[j]= rand()%(32000);
    }
    if (A[1] > A[2] && A[1] > A[3]){

        contador++;                     //Contador comp

        m1 = A[1];      contador++;     //contador Asig

        if (A[2] > A[3]){

            contador++;                 //Contador comp

            m2 = A[2];  contador++;     //Contador Asig
            m3 = A[3];  contador++;     //Contador Asig
        }
        else{

            contador++;                 //Contador comp

            m2 = A[3];  contador++;     //Contador asig
            m3 = A[2];  contador++;     //Contador asig
        }
    }
    else if (A[2] > A[1] && A[2] > A[3]){
        contador+=2;                    //Contador comp
        m1 = A[2];      contador++;     //Contador Asig
        if (A[1] > A[3]){

            contador++;                 //Contador comp

            m2 = A[1];  contador++;     //Contador Asig
            m3 = A[3];  contador++;     //Contador Asig
        }
        else{

            contador++;                 //Contador comp
```

```
            m2 = A[3];  contador++;     //Contador Asig
            m3 = A[1];  contador++;     //Contador Asig
        }
    }
    else{

        contador+=2;                    //Contador comp

        m1 = A[3];  contador++;         //Contador Asig

        if (A[1] > A[2]){

                contador++;             //Contador comp

            m2 = A[1];  contador++;     //Contador Asig
            m3 = A[2];  contador++;     //Contador Asig
        }
        else{

            contador++;                 //Contador comp

            m2 = A[2];  contador++;     //Contador comp
            m3 = A[1];  contador++;     //Contador comp
        }
    }

    i = 4;

    while( i <= n ){

        if (A[i] > m1){

            contador++;                 //Contador comp

            m3 = m2;    contador++;     //Contador Asig
            m2 = m1;    contador++;     //Contador Asig
            m1 = A[i];  contador++;     //Contador Asig
        }
        else if (A[i] > m2){

            contador+=2;                //Contador comp

            m3 = m2;    contador++;     //Contador Asig
            m2 = A[i];  contador++;     //Contador Asig
        }
        else if (A[i] > m3){

            contador+=3;                //Contador comp

            m3 = A[i];  contador++;     //Contador Asig
        }
```

```
        else{

            contador+=3;                //Contador comp
        }

        i++;
    }

    promedio+=contador;
}
```

Ya que hemos definido las operaciones básicas podemos proceder a realizar nuestro análisis de casos, donde establecemos la formula correspondiente para cada una de las situaciones: **mejor, peor, caso medio:**

Mejor Caso

Para el mejor caso tendremos en cuenta que los primeros 3 números del arreglo son los 3 números mas grandes, por lo que al momento de entrar en el ciclo WHILE únicamente se efectuaran las comparaciones, sin la necesidad de entrar en ellas.

A continuación, se presentan las imágenes representativas de cada uno de los segmentos del código para la representación del caso correspondiente:

```
if (A[1] > A[2] && A[1] > A[3]){

    contador++;                     //Mejor Caso

    m1 = A[1];    contador++;       //Mejor Caso

    if (A[2] > A[3]){

        contador++;                 //Mejor Caso

        m2 = A[2];  contador++;     //Mejor Caso
        m3 = A[3];  contador++;     //Mejor Caso
    }
    else{...
}
else if (A[2] > A[1] && A[2] > A[3]){...
    else{...

i = 4;

while( i <= n ){

    if (A[i] > m1){                 //Mejor Caso...
    else if (A[i] > m2){            //Mejor Caso...
    else if (A[i] > m3){            //Mejor Caso...
    else{...

    i++;
}
```

Función de Complejidad Temporal

$$\boldsymbol{f_t(n) = 1\ comparación + 1\ asignación + 1\ comparación + 2\ asignaciones + 3(n-3) comparaciones}$$

$$f_t(n) = 1 + 1 + 1 + 2 + 3n - 9 = \boldsymbol{3n - 4}$$

Mora Ayala José Antonio | 3CM12
Análisis de Algoritmos | Ejercicio: "Análisis de Casos"

Peor Caso

```
if (A[1] > A[2] && A[1] > A[3]){        //!Operacion PC   ...
else if (A[2] > A[1] && A[2] > A[3]){   //!Operacion PC

        contador+=2;

        m1 = A[2];                      //!Operacion PC

        if (A[1] > A[3]){ ...
        else{ ...
}
        else{

        contador+=2;

        m1 = A[3];

        if (A[1] > A[2]){               //!Operacion PC



            m2 = A[1];                  //!Operacion PC
            m3 = A[2];                  //!Operacion PC
        }
        else{ ...
}

i = 4;

while( i <= n ){

        if (A[i] > m1){                 //!Operacion PC



            m3 = m2;                    //!Operacion PC
            m2 = m1;                    //!Operacion PC
            m1 = A[i];                  //!Operacion PC
        }
```

Como ya sabemos, el peor caso será aquel que tome la ruta más larga para poder concluir el algoritmo con respecto a las operaciones básicas que hemos establecido, la cual consiste en que los números más grandes del arreglo se encuentren precisamente al final del mismo, obligando al algoritmo a pasar y realizar la comparación por todos los elementos, dando como resultado la siguiente función

$$\begin{aligned} f_t(n) = 2\ comparaciones + 1\ asignación \\ + 1\ comparación \\ + 2\ asignaciones \\ + (n-3\ )comparaciones \\ + 3(n-3)Asignaciones \end{aligned}$$

$$f_t(n) = 6 + 4n - 12 - \mathbf{1} = \mathbf{4n - 7}$$

Podemos observar como se realiza la extracción en una unidad a la última operación para obtener la función final, debido a que nuestro ciclo While incluye la iteración con igualdad a n y sabemos que un arreglo es de tamaño 0----n-1, por lo cual en ese lugar no hay existencia de elemento.

Caso Medio

```
    if (A[1] > A[2] && A[1] > A[3]){         //-Operacion-CM
        m1 = A[1];                           //-Operacion-CM

        if (A[2] > A[3]){                    //-Operacion-CM
            m2 = A[2];                       //-Operacion-CM
            m3 = A[3];                       //-Operacion-CM
        }
        else{ ...
    }
    else if (A[2] > A[1] && A[2] > A[3]){ ...
    else{ ...

    i = 4;

    while( i <= n ){

        if (A[i] > m1){                      //-Operacion-CM
            m3 = m2;                         //-Operacion-CM
            m2 = m1;                         //-Operacion-CM
            m1 = A[i];                       //-Operacion-CM
        }
        else if (A[i] > m2){                 //-Operacion-CM
            m3 = m2;                         //-Operacion-CM
            m2 = A[i];                       //-Operacion-CM
        }
        else if (A[i] > m3){                 //-Operacion-CM
            m3 = A[i];                       //-Operacion-CM
        }
        else{ ...
```

Para sacar una media debemos considerar la mayor cantidad de rutas posibles, desde la perspectiva que hemos tomado y con la contabilización de las operaciones básicas establecidas en un inicio, he llegado a la siguiente función que nos determinara un resultado satisfactorio

1) Todos son false (Mejor Caso)

$$f_t(n) = 3n - 4$$

2. La primera comparación es true lo demás false

$$f_t(n) = 4n - 8$$

Pues contamos con las siguientes operaciones:

- 1 comparación
- 1 asignacion
- 1 comparacion
- 2 asignaciones
- (n-3) comparaciones
- 3(n-3) Asignaciones -1

3. Segunda Condición true el resto falso

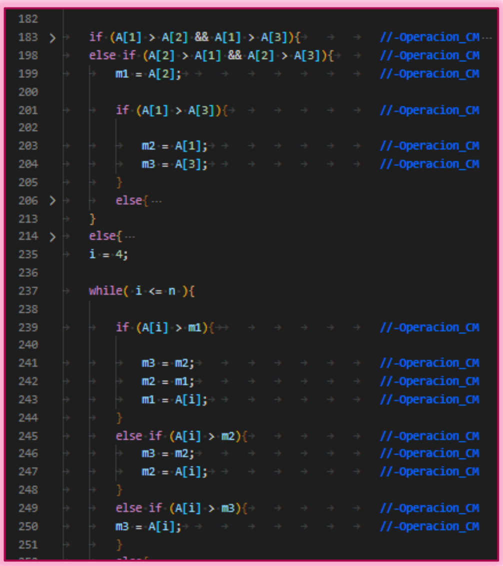
$$f_t(n) = 4n - 8$$

- 1 comparacion
- 1 Asignacion
- 1 comparacion
- 2 asiganaciones
- 2(n-3) comparaciones
- 2(n-3) asignaciones -1

4. **Tercera condición true, el resto falso**

$$f_t(n) = 4n - 8$$

- 1 comparacion
- 1 Asignacion
- 1 comparacion
- 2 asiganaciones
- 3(n-3) comparaciones
- (n-3) asignaciones -1



```
if (A[1] > A[2] && A[1] > A[3]){                //-Operacion_CM···
else if (A[2] > A[1] && A[2] > A[3]){           //-Operacion_CM
    m1 = A[2];                                  //-Operacion_CM

    if (A[1] > A[3]){                           //-Operacion_CM

        m2 = A[1];                              //-Operacion_CM
        m3 = A[3];                              //-Operacion_CM
    }
    else{···
}
else{···
i = 4;

while( i <= n ){

    if (A[i] > m1){                             //-Operacion_CM

        m3 = m2;                                //-Operacion_CM
        m2 = m1;                                //-Operacion_CM
        m1 = A[i];                              //-Operacion_CM
    }
    else if (A[i] > m2){                        //-Operacion_CM
        m3 = m2;                                //-Operacion_CM
        m2 = A[i];                              //-Operacion_CM
    }
    else if (A[i] > m3){                        //-Operacion_CM
    m3 = A[i];                                  //-Operacion_CM
    }
```

Si nos ponemos a observar el segundo camino posible será muy similar al primero, solo que en lugar de entrar en la primera comparativa como verdadero, será falso por lo que en la segunda también, debido a que son dependientes por lo que entraría a nuestro caso ELSE

1. Todos son FALSE (Peor Caso)

$$\boldsymbol{f_t(n) = 4n - 7}$$

2. Primera comparativa true, el resto false.

$$\boldsymbol{f_n(n) = 4n - 7}$$

Ya que contamos con:

- 2 Comparaciones
- 1 Asignación
- 1 Comparación
- 2 Asignaciones
- (n-3) Comparaciones
- 3(n-3) Asignaciones

3. Segunda condición true

$$\boldsymbol{f_t(n) = 4n - 7}$$

- 2 Comparaciones
- 1 Asignación

- 1 Comparación
- 2 Asignaciones
- 2(n-3) Comparaciones
- 2(n-3) Asignaciones -1

4. Tercera condición true

$$\boldsymbol{f_t(n) = 4n - 7}$$

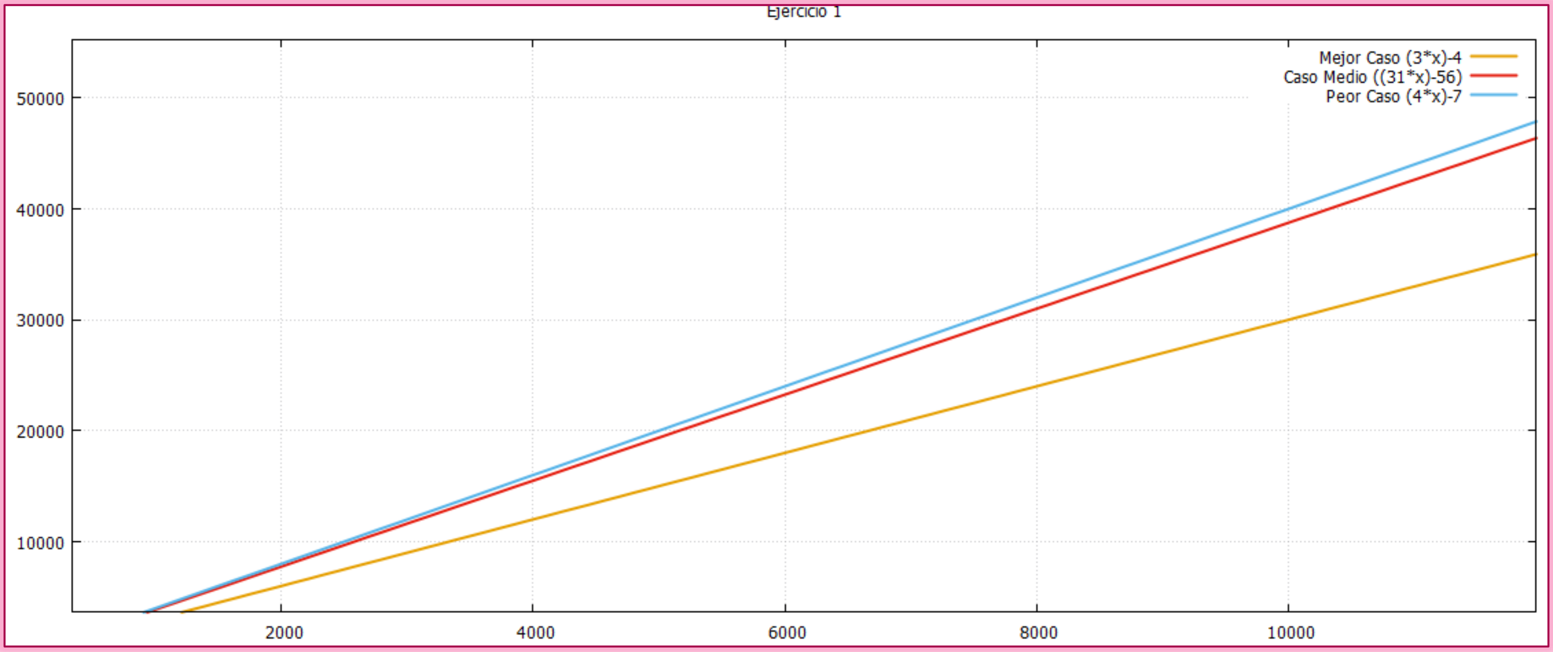
- 2 Comparaciones
- 1 Asignación
- 1 Comparación
- 2 Asignaciones
- 3(n-3) Comparaciones
- (n-3) Asignaciones -1

Por último, recordemos que cada camino que pudiese tomar es considerado como equiprobable por lo que nuestra función quedaría expresada de la siguiente forma:

$$f(n) = \frac{1}{8}\big(3n - 4 + 3(4n - 8) + 4(4n - 7)\big)$$

$$f(n) = \frac{1}{8}(3n - 4 + 12n - 24 + 16n - 28)$$

$$\mathbf{f(n) = \frac{31n - 56}{8}}$$


Ejercicio 1
Mejor Caso (3*x)-4
Caso Medio ((31*x)-56)
Peor Caso (4*x)-7
50000
40000
30000
20000
10000
2000
4000
6000
8000
10000

CÓDIGO 2

```
int MaximoComunDivisor(m, n){
    a=max(n,m);
    b=min(n,m);
    residuo=1;
    while (residuo>0)
    {
        residuo = a%b;
        a=b;
        b=residuo;
    }
    MaximoComunDivisor=a;
    return MaximoComunDivisor;
}
```

Para este caso en particular, la operación básica asignada a seguir fue aquella donde estuviera involucrada la operación de modulo entre a y b, la cual esta asignada a la variable de residuo, establecido esto, podemos proceder a realizar el análisis de los casos

Mejor Caso

El mejor caso será aquel donde encontremos que las dos entradas dadas para el algoritmo sean múltiplos, por ejemplo: 5 y 10 , pues el modulo siempre será 0 y solo se realizaría una vez dicha operación

$$f_t(n,m) = 1 \; si \; y \; solo \; si \; MCD(n,m) = n \; Donde \; n \; es \; múltiplo \; de \; m \; y \; a \; su \; vez \; menor \; que \; m$$

Peor Caso

El peor caso será aquel donde los números ingresados, sean 2 números consecutivos pertenecientes a la serie de Fibbonacci, mientras más grandes sean los valores de esta serie mayor será el numero de operaciones a realizar

Gracias a las fuentes consultas en internet acerca del algoritmo en cuestión (Algoritmo de Euclides) estará establecida aproximadamente por la siguiente formula:

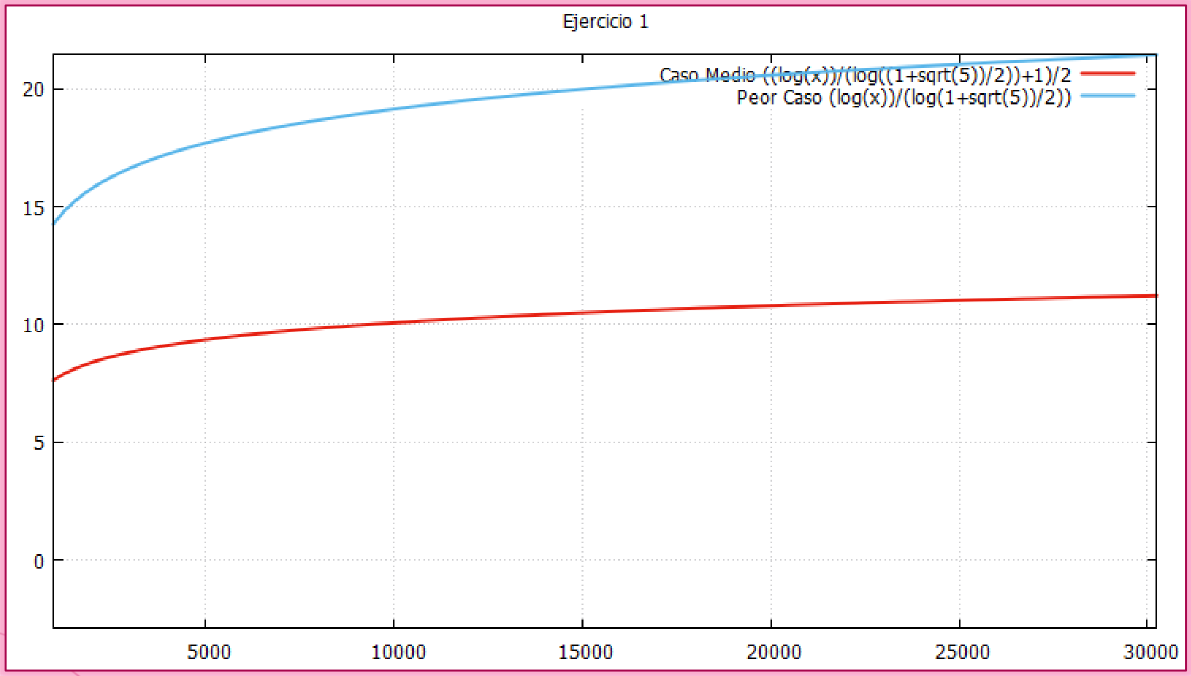
$$\boldsymbol{f_t(a) = \frac{\log a}{\log \Phi} \; donde \; a \geq b \; y \; \Phi = \frac{1+\sqrt{5}}{2}}$$

Caso Medio

Dado el análisis anterior pude notar que solo tendríamos la posibilidad de que el MCD únicamente se haga en una operación o que se haga x cantidad de veces, lo cual sería la expresión del peor caso, por lo que propongo la siguiente expresión para esta cuestión:

$$f(n,m) = \frac{1}{2}\left(\frac{log(a)}{log(\phi)} + 1\right)\ donde\ a = min\ (n,m)\ [1]\,[2]\,[3]$$

Gráfica



CÓDIGO 3

```
int SumaCuadratica3MayoresV2(){
    scanf("%d", &n);
    A = malloc(sizeof(int)*n);
    for (j = 0; j < n; j++){

        A[j]= rand()%(32000);
    }
    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);
}
```

Mejor Caso

$$f_t(n) = 3n - 5$$

Dado que tendríamos la entrada al ciclo de i y la entrada al ciclo de j, si la condición no se cumple tendremos como consecuencia lo siguiente:

- (n-1) Comparaciones
- (n-2) Comparaciones
- (n-3) Comparaciones
- 1 Asignación

Peor Caso

$$f_t(n) = 12n - 23$$

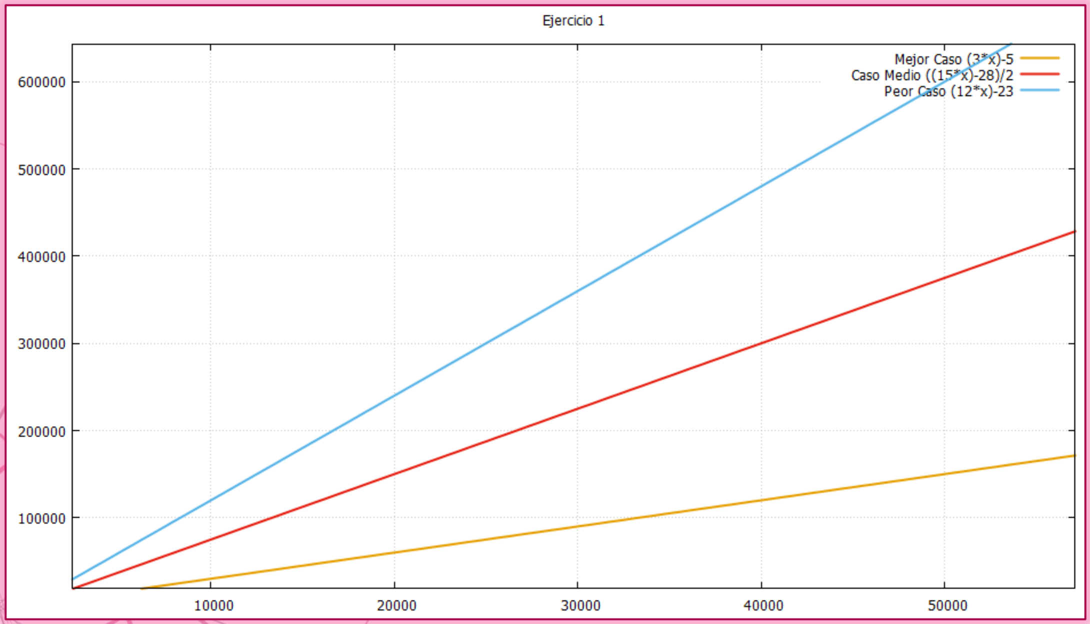
El peor caso estará dado cuando los números estén ordenados de forma descendente, por lo que se encontraran en las ultimas posiciones, dando como consecuencia que la condicional sea efectiva y las asignaciones realizadas teniendo como resultado las siguientes operaciones:

- (n-1) Comparaciones
    - 3(n-1) Asignaciones
- (n-2) Comparaciones
    - 3(n-2) Asignaciones
- (n-3) Comparaciones
    - 3(n-3) Asignaciones
- 1 Asignación

Caso Medio

$$f_t(n) = \frac{15n - 28}{2}$$

Para el caso medios consideraron las rutas analizadas anteriormente, tomando aquellas donde sucede en el mejor caso junto con el peor caso, realizando una división entre 2.



CÓDIGO 4

```
    int Codigo4(){
    int f,i,j,num,ftemp,ntemp,n,A[n];
    scanf("%d",&n);
    i=1;
    while (i<=n)
    {
        scanf("%d",A[i]);
        i+=1;
    }
    f=0;
    i=1;
    while (i<=n)
    {
        ntemp=A[i];
        j=1;
        ftemp=0;
        while (j<=n)
        {
            if (ntemp=A[j])
            {
                ftemp=ftemp+1;
            }
            j=j+1;
        }

        if (f<ftemp)
        {
            f=ftemp;
            num=ntemp;
        }
        i=i+1;
    }
    printf("d",num);
}
```

Para este código se tendrá en consideración las siguientes operaciones:

- Aquellas que tienen que ver con el tamaño del problema (ósea que están relacionadas directamente con el arreglo)
- Variables que determinan la cantidad de operaciones que se van a realizar, las cuales están dadas por **ftrmp** y **ntemp**

Mejor Caso

Para este caso tendremos un arreglo donde los números que este contiene nunca se repitan, cada valor es único dentro del arreglo de tamaño n, de tal forma que se evitaría la realización de operaciones a la variable ftemp, además de que no siempre se realizaría la asignación de num en el IF que esta fuera del cilo WHILE de j por lo que tendremos:

$$f_t(n) = n^2 + 4n + 2$$

- 2 Asignaciones
- (n) Asignaciones + (n) Asignaciones
- (n^2) Comparaciones + (n) Asignaciones +
- (n) Comparaciones + 2 Asignaciones

Peor Caso

Para este caso tendremos la situación de un arreglo lleno del mismo numero una n cantidad de veces, dada esta situación la condicional dentro del while siempre será cumplida

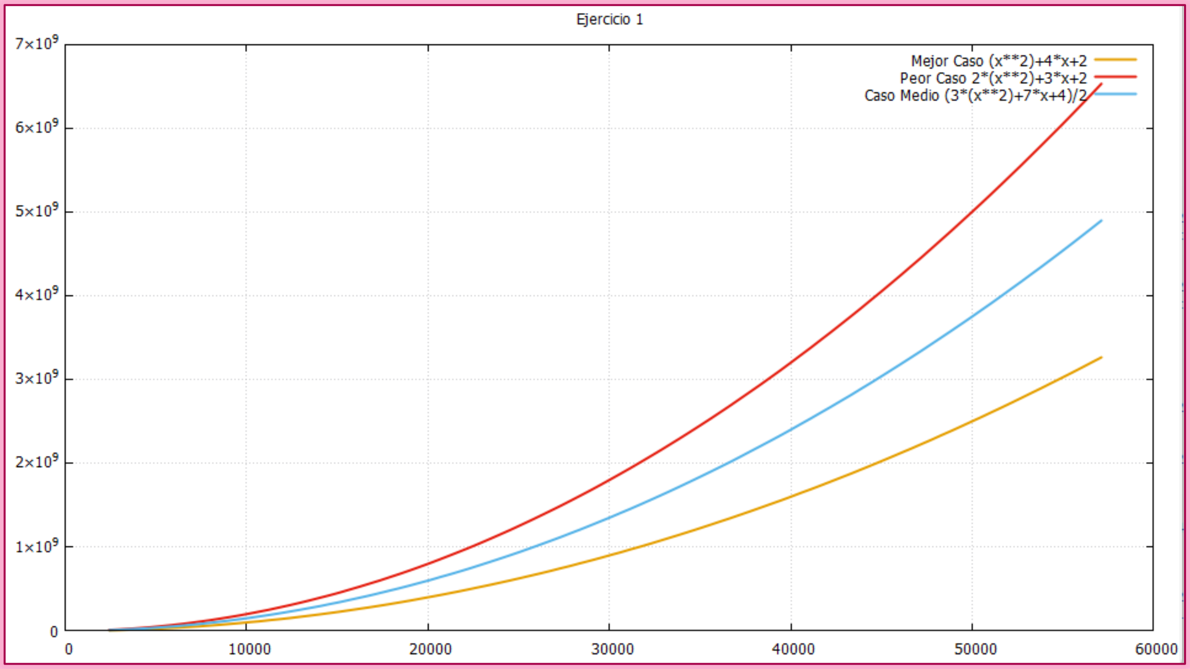
$$f_t(n) = 2n^3 + 3n + 2$$

- n Asignaciones + (n) Asignaciones + (n^2) Comparaciones
- (n^2) Comparaciones + (n) Asignaciones + 2 Asignaciones

Caso Medio

Para este caso, puede llegar a ser algo complejo determinar un caso medio debido a la naturaleza que posee el objetivo del código y realmente no existen muchas alternativas posibles que seguir o considerar, por lo que realizaremos la unión del mejor y peor caso, dando una división entre 2, estoy consciente de que probablemente no es la mejor opción, pero me parece una buena alternativa para encontrar una buena solución con resultados muy cercanos a la realidad de ejecución en pruebas a posteriori.

$$f_t(n) = \frac{3n^2 + 7n + 4}{2}$$


Ejercicio 1
Mejor Caso (x**2)+4*x+2
Peor Caso 2*(x**2)+3*x+2
Caso Medio (3*(x**2)+7*x+4)/2
7×10^9
6×10^9
5×10^9
4×10^9
3×10^9
2×10^9
1×10^9
0
0
10000
20000
30000
40000
50000
60000

CICLO 5

```
polinomio=0;

for (i = 0; i <= n; i++)
{
    polinomio=polinomio*z + A[n-i];
    cont+=1;
}
printf("%d\n",polinomio);
printf("%d",cont);
```

Mejor Caso

$$f_t(n) = 1 \rightarrow Cuando\ n = 0$$

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

Con respecto a las operaciones previamente establecidas, he determinado que el mejor caso será aquel cuando n=0 o n=1 ya que si le damos un arreglo vacío la operación será realizada únicamente en una iteración, aunque son muy poco probables aquellas situaciones en las que un arreglo será usado de forma vacía precisamente con la intención de operar, por lo que lo mejor sería poder mandar un arreglo con al menos un elemento.

Peor Caso

El peor caso desde mi perspectiva sería aquel donde el tamaño del arreglo sea mayor que 1, pues las operaciones que se irían realizando serían cada vez más y más, por lo que el tamaño máximo que aguantaría sería aquellas donde sea del tamaño máximo del tipo de variable con la cual el arreglo en cuestión fue declarado.

Caso Medio

Con respecto a lo anteriormente expuesto el caso medio como tal no podría estar determinado, pues, dependemos directamente del valor de "n.

BIBLIOGRAFÍA

[1] M. R. Castañeda, «Super Prof Material Didactico,» 18 08 2017. [En línea]. Available: https://www.superprof.es/apuntes/escolar/matematicas/aritmetica/divisibilidad/algoritmo-de-euclides.html. [Último acceso: 17 Septiembre 2021].

[2] UNAM, «Instituto de Matemáticas UNAM,» 21 Febrero 2009. [En línea]. Available: https://paginas.matem.unam.mx/cprieto/biografias-de-matematicos-a-e/198-euclides. [Último acceso: 17 09 2021].

[3] G. Brassard y P. Bratley, Fundamentos de Algoritmia., Madrid: PrenticeHall, 1997.