INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

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

ACTIVIDAD

Determinando funciones de complejidad

PROFESOR:

Franco Martínez Edgardo Adrián

ALUMNO:

Meza Vargas Brandon David – 2020630288

GRUPO:

3CM13

Antes de comenzar es importante recalcar que para el análisis y obtención de las funciones se están tomando los siguientes criterios:

- **Asignaciones**
- **Operaciones aritméticas**
- **Saltos (gF – falso, gT - true)**
- **Comparaciones**

Además, en el análisis no entra la variable nInstru y las operaciones con esta, ya que solo es para determinar cuántas instrucciones fueron en total al verificar el código. También aclaramos que para las funciones complejidad espacial no se toma en cuenta la variable n.

Código 01

```
int main(int argc, char **argv){

    int i, temp, k, j, n, nInstru=0;
    int *A;

    if(argc != 2)
        exit(1);

    n = atoi(argv[1]);

    A = (int *)malloc(40000*sizeof(int));

    for(i=1, nInstru++;i<n;i++, nInstru++) {
        nInstru++; //Condicional TRUE en i
        for(j=n, nInstru++;j>1;j/=2, nInstru++){
            nInstru++, //Condicional TRUE en j
            temp = A[j]; nInstru++;
            A[j] = A[j+1]; nInstru+=2;
            A[j+1] = temp; nInstru+=2;
            nInstru++; //saltos TRUE j
        }

        nInstru++; //Saltos FALSE de j
        nInstru++; //Condicional FALSE en j
        nInstru++; //Saltos TRUE en i
    }

    nInstru++; //Salto FALSE en i
    nInstru++; //Comparacion FALSE en i

    printf("\nInstrucciones: %d", nInstru);
```

Función complejidad temporal

```
for(i=1, nInstru++;i<n;i++, nInstru++) {
```

- **(1+n+n-1)(asignación+comparaciones+aritmeticas) = 2n**
- **1 (gF i)**
- **n-1 (gT i)**

```
for(j=n, nInstru++;j>1;j/=2, nInstru++){
    nInstru++, //Condicional TRUE en j
    temp = A[j]; nInstru++;
    A[j] = A[j+1]; nInstru+=2;
    A[j+1] = temp; nInstru+=2;
    nInstru++; //saltos TRUE j
}
```

- **(1+(log2(n)+1)+log2(n))*(n-1) (asignación+comparaciones+aritmeticas)*(gT i) = 2nlog2(n)-2log2(n)+2n-2**
- **n-1 (gF j)**
- **log2(n)*(n-1) (veces que se ejecuta for interno)*(gT i) = nlog(n)-log2(n) (gT j)**
- **(3+2)*(nlog(n)-log2(n)) (comparaciones+aritméticas dentro de for interno) (gT j) = 5nlog2(n)-5log2(n)**

De esta forma, recordando que el logaritmo es piso, al reducir términos nos queda:

$$f(n) = 8n\lfloor \log_2(n) \rfloor - 8\lfloor \log_2(n) \rfloor + 6n - 3$$

Siempre y cuando n > 0, pues para n < 0 será 6n - 3

Función complejidad espacial

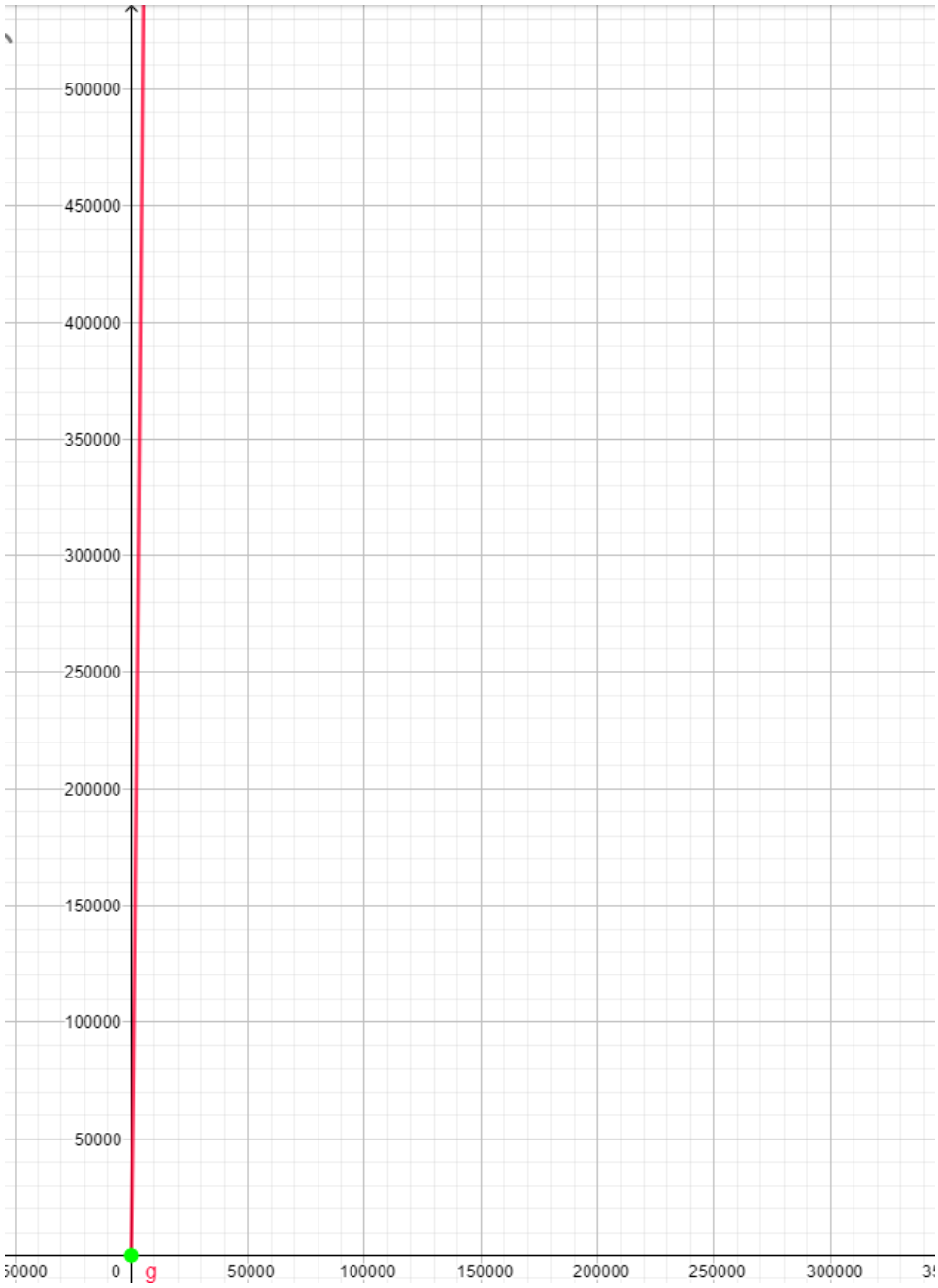
- **1 variable i**
- **1 variable j**
- **1 variable temp**
- **n variables del arreglo A**

$$f(n) = n + 3$$

Tabla Comparativa función complejidad temporal

N	Resultados Teóricos	Resultados Empíricos
-1	**3**	**3**
0	**3**	**3**
1	**3**	**3**
2	**17**	**17**
3	**31**	**31**
5	**91**	**91**
15	**423**	**423**
20	**725**	**725**
100	**5349**	**5349**
409	**28563**	**28563**
500	**34933**	**34933**
593	**46179**	**46179**
1000	**77925**	**77925**
1471	**126423**	**126423**
1500	**128917**	**128917**
2801	**263203**	**263203**
3000	**281909**	**281909**
5000	**509901**	**509901**
10000	**1099893**	**1099893**
20000	**2359885**	**2359885**

Gráfica


500000
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
g
50000
100000
150000
200000
250000
300000

Código 02

```c
int main(int argc, char **argv){

    int i, n, nInstru=0;
    int *A, z;
    if(argc != 2)
        exit(1);

    n = atoi(argv[1]);
    A = (int *)malloc(40000*sizeof(int));

    int polinomio = 0; nInstru++;
    for(i=0, nInstru++;i<=n;i++, nInstru++) {
        nInstru++; //Condicional TRUE en i
        polinomio = polinomio*z+A[n-i]; nInstru+=4;
        nInstru++; //Saltos TRUE en i
    }

    nInstru++; //Salto FALSE en i
    nInstru++; //Comparacion FALSE en i

    printf("\nInstrucciones: %d", nInstru);
```

Función complejidad temporal

- **1 (asignación)**
- **(1+(n+1+1)+(n+1)) (asignación+comparaciones+aritmeticas) = 2n+4**
- **n+1 (gT i)**
- **1 (gF i)**
- **(1+3)*(n+1) (asignación+operaciones)*(gT i) = 4n+4**

Así, al sumar los términos nos queda la siguiente función:

$$f(n) = 7n + 11$$

Función complejidad espacial

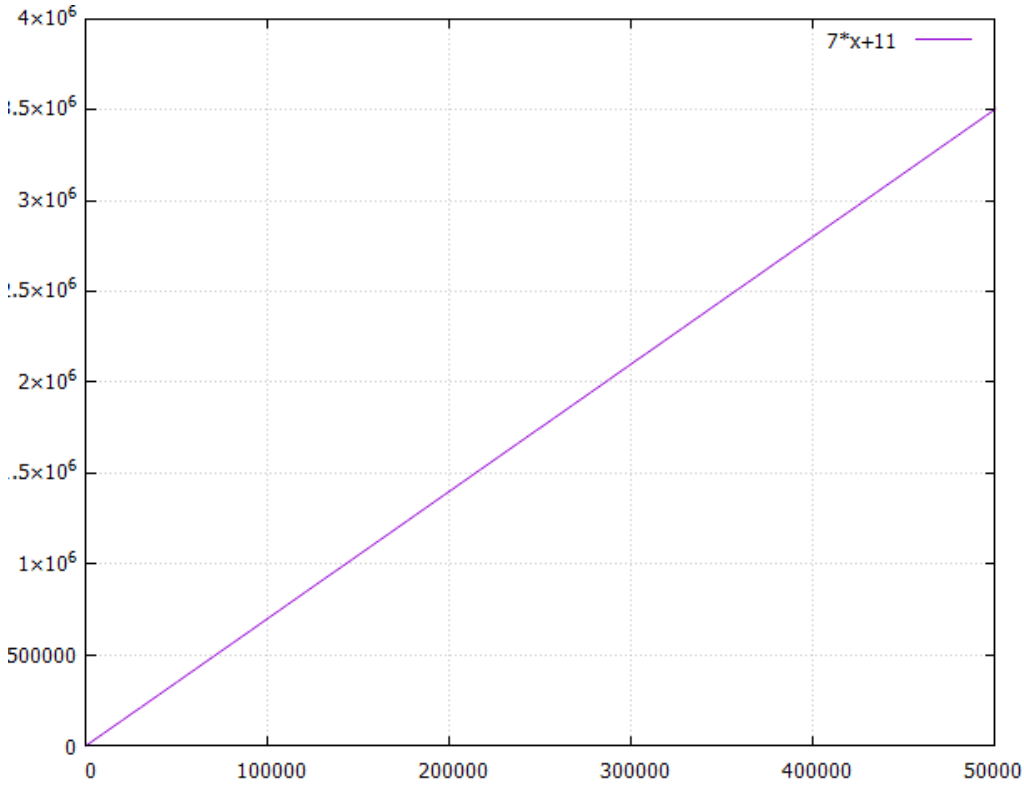
- **1 variable polinomio**
- **1 variable i**
- **1 variable z**
- **n variables del arreglo A**

$$f(n) = n + 3$$

Tabla Comparativa función complejidad temporal

N	Resultados Teóricos	Resultados Empíricos
-1	**4**	**4**
0	**11**	**11**
1	**18**	**18**
2	**25**	**25**
3	**32**	**32**
5	**46**	**46**
15	**116**	**116**
20	**151**	**151**
100	**711**	**711**
409	**2874**	**2874**
500	**3511**	**3511**
593	**4162**	**4162**
1000	**7011**	**7011**
1471	**10308**	**10308**
1500	**10511**	**10511**
2801	**19618**	**19618**
3000	**21011**	**21011**
5000	**35011**	**35011**
10000	**70011**	**70011**
20000	**140011**	**140011**

Gráfica



Código 03

```
int main(int argc, char **argv){

    int i, j, k, n, nInstru=0;
    int *A, *B, *C;

    A = (int *)malloc(500000*sizeof(int));
    B = (int *)malloc(500000*sizeof(int));
    C = (int *)malloc(500000*sizeof(int));

    if(argc != 2)
        exit(1);

    n = atoi(argv[1]);

    for(i=1, nInstru++; i<=n; i++, nInstru++){
        nInstru++; //Condicional TRUE en i
        for (j=1, nInstru++; j<=n; j++, nInstru++){
            nInstru++; //Condicional TRUE en j
            C[i,j] = 0; nInstru++;
            for(k=1, nInstru++; k<=n; k++, nInstru++){
                nInstru++; //Condicional TRUE en k
                C[i,j] = C[i,j] + A[i,k] * B[k,j]; nInstru += 3;
                nInstru++; //Saltos TRUE k
            }
            nInstru++; //Salto FALSE k
            nInstru++; //Condicional FALSE k
            nInstru++; //Salto TRUE j
        }

        nInstru++; //Salto FALSE j
        nInstru++; //Condicional FALSE j
        nInstru++; //Saltos TRUE i
    }
    nstru++; //Salto FALSE i
    tru++; // Comparacion FALSE i
    "\nInstrucciones: %d", nInstru);
```

Función complejidad temporal

```
for(i=1, nInstru++; i<=n; i++, nInstru++){
```

- **(1+(n+1)+n) (asignaciones + comparaciones + aritméticas) = 2n+2**
- **n (gT i)**
- **1 (gF i)**

```
for (j=1, nInstru++; j<=n; j++, nInstru++){
    nInstru++; //Condicional TRUE en j
    C[i,j] = 0; nInstru++;
```

- **(1+(n+1)+n)*n (asignaciones + comparaciones + aritméticas)*(gT i) = $2n^2+2n$**
- **n*n (gT i)(Veces que entra el for intero) = n^2 (gT j)**
- **n (gF j)**

- **1*(n^2) (asignación)*(gT j) = n^2**

```
for(k=1, nInstru++; k<=n; k++, nInstru++){
    nInstru++; //Condicional TRUE en k
    C[i,j] = C[i,j] + A[i,k] * B[k,j]; nInstru += 3;
    nInstru++; //Saltos TRUE k
}
```

- **(1+ (n+1) +n)*n^2 (asignaciones + comparaciones+aritméticas)*(gT j) = $2n^3+2n^2$**
- **n(n^2) (veces que entra el for en k) (gT j) = n^3 (gT k)**
- **(1+2)*n^3 (asignación + aritméticas)*(gT k) = $3n^3$**

De esta forma, al sumar los términos anteriores, obtenemos la siguiente función:

$$f(n) = 6n^3 + 7n^2 + 6n + 3$$

Tomando en cuenta que para valores menores de 1 serán siempre 3 instrucciones.

Función complejidad espacial

- **3 variables i, j, k**
- **n*n variables de la matriz C. A y B**

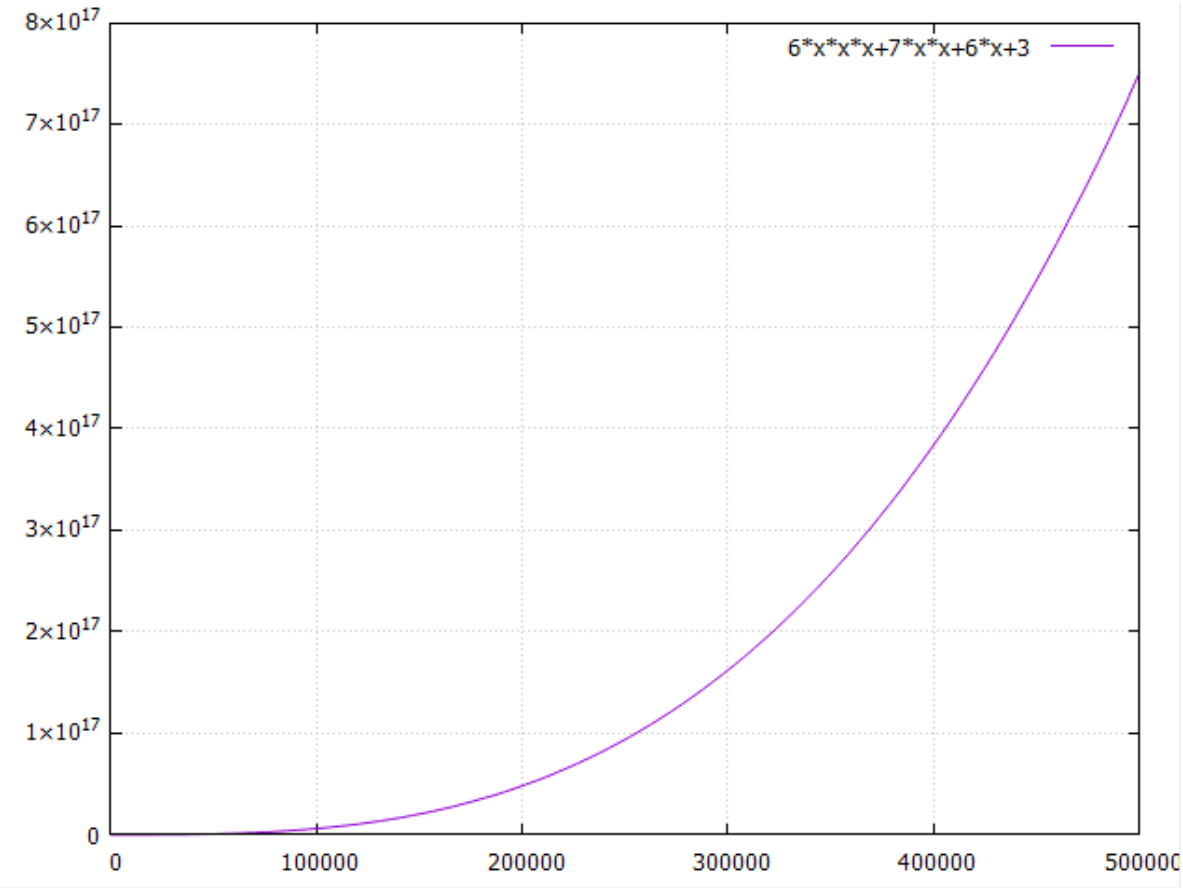
$$f(n) = 3n^2 + 3$$

Tabla Comparativa función complejidad temporal

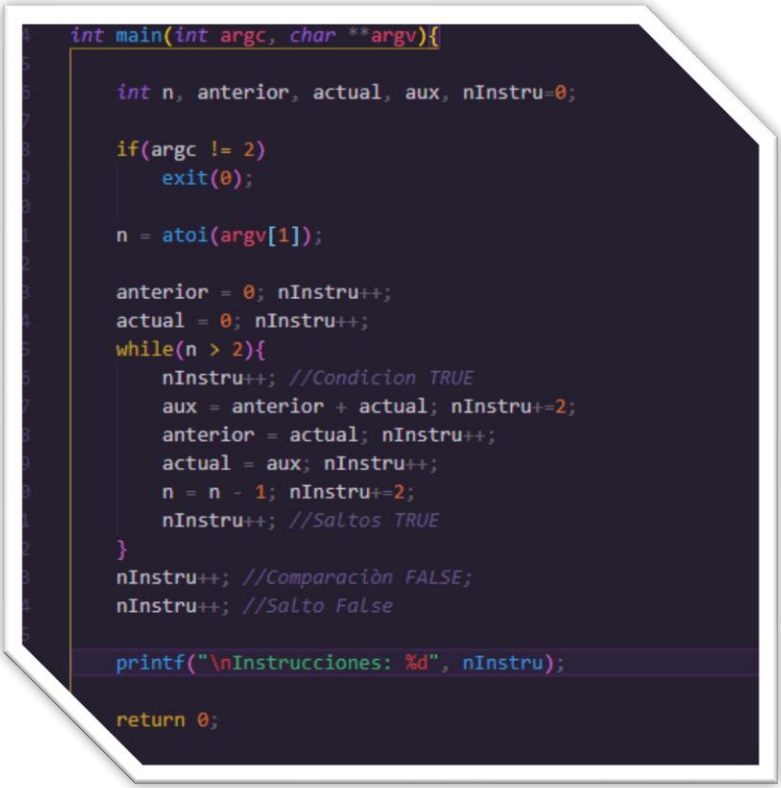
N	Resultados Teóricos	Resultados Empíricos
-1	**3**	**3**
0	**3**	**3**
1	**22**	**22**
2	**91**	**91**
3	**246**	**246**
5	**958**	**958**
15	**21918**	**21918**
20	**50923**	**50923**
100	**6070603**	**6070603**
409	**411680998**	**411680998**
500	**751753003**	**751753003**
593	**1253632246**	**1253632246**
1000	**1712038707**	**1712038707**
1471	**$1.91x10^{10}$**	**1933347198**
1500	**$2.02\ x10^{10}$**	**-1209077477**
2801	**$1.31\ x10^{11}$**	**-1235879754**
3000	**$1.62\ x10^{11}$**	**-1255894754**
5000	**$7.5\ x10^{11}$**	**-1431239914**
10000	**$6\ x10^{12}$**	**-2031455133**
20000	**$4.8\ x10^{13}$**	**-2533155690**

Como vemos en la tabla, en los resultados empíricos a partir del 1500 empieza a dar cifras erróneas debido a la gran cantidad de instrucciones generadas

Gráfica



Código 04



```c
int main(int argc, char **argv){

    int n, anterior, actual, aux, nInstru=0;

    if(argc != 2)
        exit(0);

    n = atoi(argv[1]);

    anterior = 0; nInstru++;
    actual = 0; nInstru++;
    while(n > 2){
        nInstru++; //Condicion TRUE
        aux = anterior + actual; nInstru+=2;
        anterior = actual; nInstru++;
        actual = aux; nInstru++;
        n = n - 1; nInstru+=2;
        nInstru++; //Saltos TRUE
    }
    nInstru++; //Comparaciòn FALSE;
    nInstru++; //Salto False

    printf("\nInstrucciones: %d", nInstru);

    return 0;
```

Función complejidad temporal

- **1+1 (asignación + asignación)**
- **n-1 (comparaciones)**
- **n-2 (gT ciclo)**
- **1 (gF ciclo)**
- **(4 + 2) * (n-2)(asignaciones+aritmeticas)*(gT ciclo) = 6n-12**

De esta forma nos queda la siguiente función tomando en cuenta n > 2, ya que si n<=2 el número de instrucciones siempre será 4

$$f(n) = 8n - 12$$

Función complejidad espacial

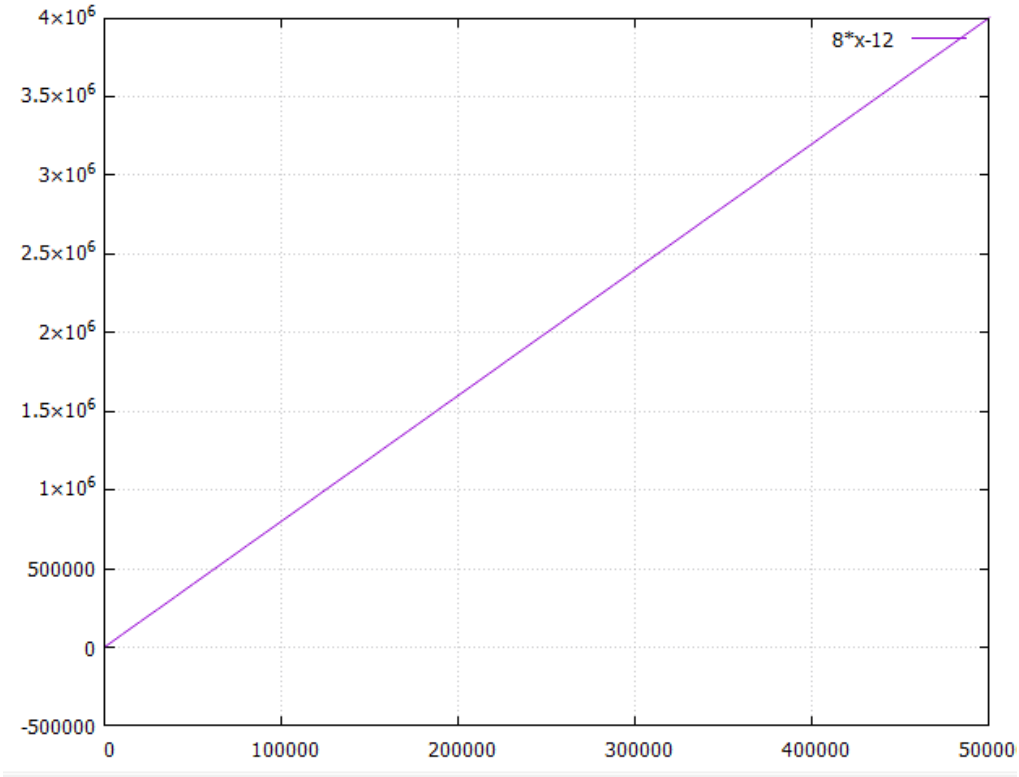
- **2 variables anterior y actual**
- **1 variable aux**

$$f(n) = 3$$

Tabla Comparativa función complejidad temporal

N	Resultados Teóricos	Resultados Empíricos
-1	**4**	**4**
0	**4**	**4**
1	**4**	**4**
2	**4**	**4**
3	**12**	**12**
5	**28**	**28**
15	**108**	**108**
20	**148**	**148**
100	**788**	**788**
409	**3260**	**3260**
500	**3988**	**3988**
593	**4732**	**4732**
1000	**7988**	**7988**
1471	**11756**	**11756**
1500	**11988**	**11988**
2801	**22396**	**22396**
3000	**23988**	**23988**
5000	**39988**	**39988**
10000	**79988**	**79988**
20000	**159988**	**159988**

Gráfica



Código 05

```c
int main(int argc, char **argv){

    int i, k, nInstru=0, n, j;
    int *s, *s2;
    s = (int *)malloc(500000*sizeof(int));
    s2 = (int *)malloc(500000*sizeof(int));

    if(argc != 2)
        exit(1);

    n = atoi(argv[1]);

    for(i=n-1,j=0,nInstru+=2;i>=0;i--,j++,nInstru+=2){
        nInstru++; //Condicional TRUE
        s2[j] = s[i]; nInstru++;
        nInstru++; //Salto TRUE
    }
    nInstru++; //Salto FALSE
    nInstru++; //Comparacion FALSE


    for(k=0,nInstru++;k<n;k++,nInstru++){
        nInstru++; //Condicional TRUE
        s[i] = s2[i]; nInstru++;
        nInstru++; //Saltos TRUE

    }
    nInstru++; //Salto FALSE
    nInstru++; //Comparacion FALSE

    printf("\nInstrucciones: %d", nInstru);


    return 0;
```

Tabla Comparativa función complejidad temporal

```c
for(i=n-1,j=0,nInstru+=2;i>=0;i--,j++,nInstru+=2){
    nInstru++; //Condicional TRUE
    s2[j] = s[i]; nInstru++;
    nInstru++; //Salto TRUE
}
```

- ➔ **1+1 (asignación i, asignación j) = 2**
- ➔ **n+1 (comparaciones)**
- ➔ **n+n (aritméticas + aritméticas j)**
- ➔ **n (asignaciones)**
- ➔ **1 (gF)**
- ➔ **n (gT)**

```
for(k=0,nInstru++;k<n;k++,nInstru++){
    nInstru++; //Condicional TRUE
    s[i] = s2[i]; nInstru++;
    nInstru++; //Saltos TRUE

}
```

- ➔ **(1+(n+1)+n) (asignación + comparaciones + aritméticas) = 2n+2**
- ➔ **n (asignaciones)**
- ➔ **1 (gf)**
- ➔ **n (gT)**

De esta forma, al sumar los términos, nos queda la siguiente función:

$$f(n) = 9n + 7$$

Tengamos en cuenta que para valores menores a 1, las operaciones serán solo 7

Función complejidad espacial

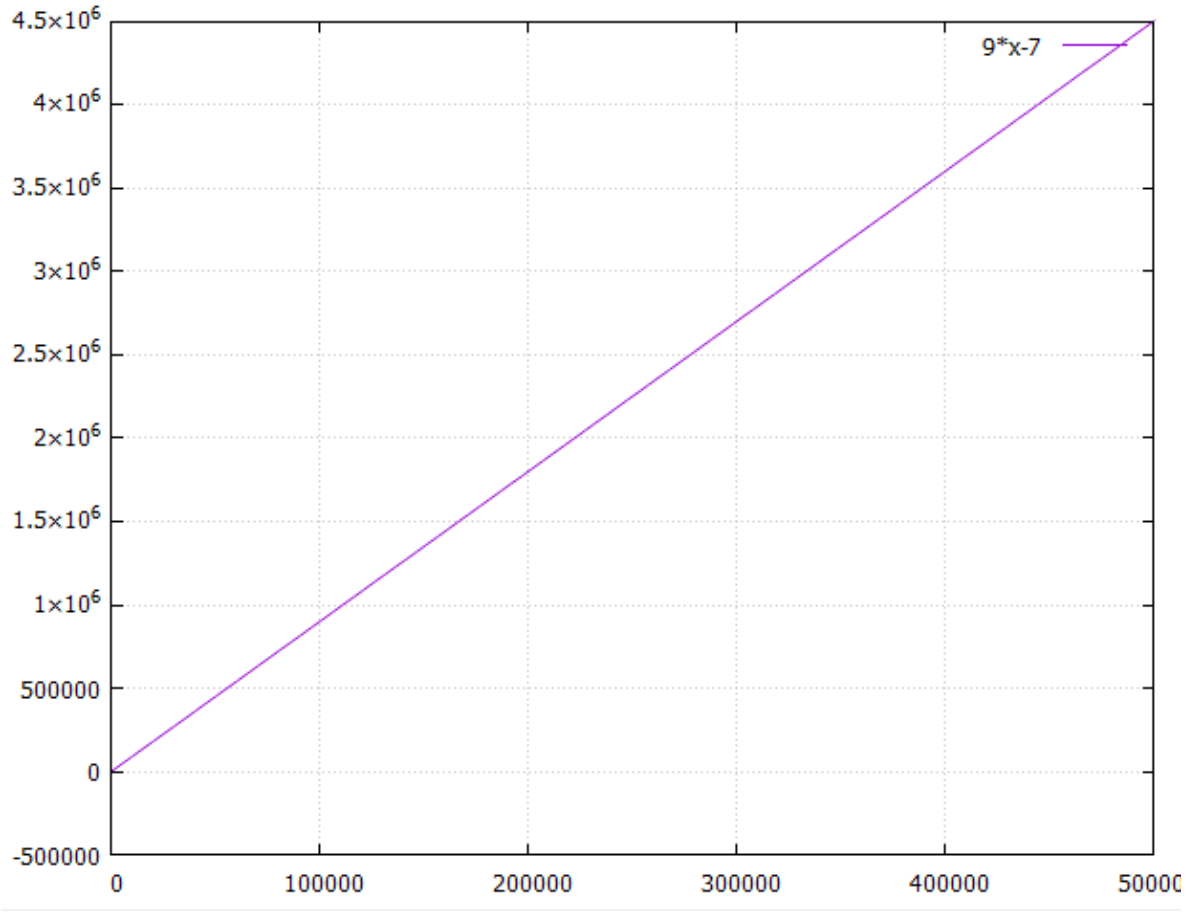
- ➔ **3 variables i, j, k**
- ➔ **n variables de los arreglos s y s2**

$$f(n) = 2n + 3$$

Tabla Comparativa

N	Resultados Teóricos	Resultados Empíricos
-1	**7**	**7**
0	**7**	**7**
1	**16**	**16**
2	**25**	**25**
3	**34**	**34**
5	**52**	**52**
15	**142**	**142**
20	**187**	**187**
100	**907**	**907**
409	**3688**	**3688**
500	**4507**	**4507**
593	**5344**	**5344**
1000	**9007**	**9007**
1471	**13246**	**13246**
1500	**13507**	**13507**
2801	**25216**	**25216**
3000	**27007**	**27007**
5000	**45007**	**45007**
10000	**90007**	**90007**
20000	**180007**	**180007**

Gráfica


9*x-7
4.5×10^6
4×10^6
3.5×10^6
3×10^6
2.5×10^6
2×10^6
1.5×10^6
1×10^6
500000
0
-500000
0
100000
200000
300000
400000
50000

Código 06

```
int l, a, b, nInstru=0, r, i, auxa=0, auxb=0;

if(argc != 3)
    exit(1);

a = atoi(argv[1]);
b = atoi(argv[2]);

l = (a<b)?a:b; nInstru+=2;
r=1; nInstru++;

for(i=2,nInstru++;i<=l;i++,nInstru++){
    nInstru++; //Comparacions true
    if(a%i==0&&b%i==0)
        printf("\n true %d %d %d",a%i,b%i, i);

    if(a%i==0&&b%i==0){
        nInstru+=5;
        nInstru++; //Condicional TRUE if
        r=i;nInstru++;
        nInstru++; //Salto TRUE if
        auxa++;
    }else{
        nInstru+=5;
        nInstru++; //salto false
        nInstru++; //condicional false
    }
    nInstru++; //salto true
}
nInstru++; //comparacion FALSE i
nInstru++; //Salto FALSE i
printf("\nInstrucciones: %d", nInstru);
```

Función complejidad temporal

Analizando el código me doy cuenta de que todo va bien hasta llegar al if, puesto que este entrará siempre y cuando su condición se cumpla, dicha condición resulta algo complicada de analizar, pero si vemos bien entramos en razón que la cantidad de veces que se cumplirá serán el numero de divisores que tiene el máximo factor común de a y b. A partir de esta afirmación podemos establecer una función de complejidad de la siguiente forma:

- ➔ (1+1) (asignación y comparación del primer if)
- ➔ 1 asignación de r
- ➔ (1+n+(n-1)) (asignaciones + comparaciones + aritméticas del if) = 2n
- ➔ n-1 (gT for)
- ➔ 1 (gF for)
- ➔ (3+2)(n-1) (operaciones + comparaciones en if)*(gT for) = 5n-5
- ➔ (1)*(cantidad de divisores del MFC de a y b, lo llamaremos cDiv)* = cDiv

- ➔ cDiv (gT if)
- ➔ n – cDiv (gF if)

$$f(n) = cDiv + 9n - 2$$

Cuando l toma un valor menor a 2 las instrucciones siempre serán 6

Función complejidad espacial

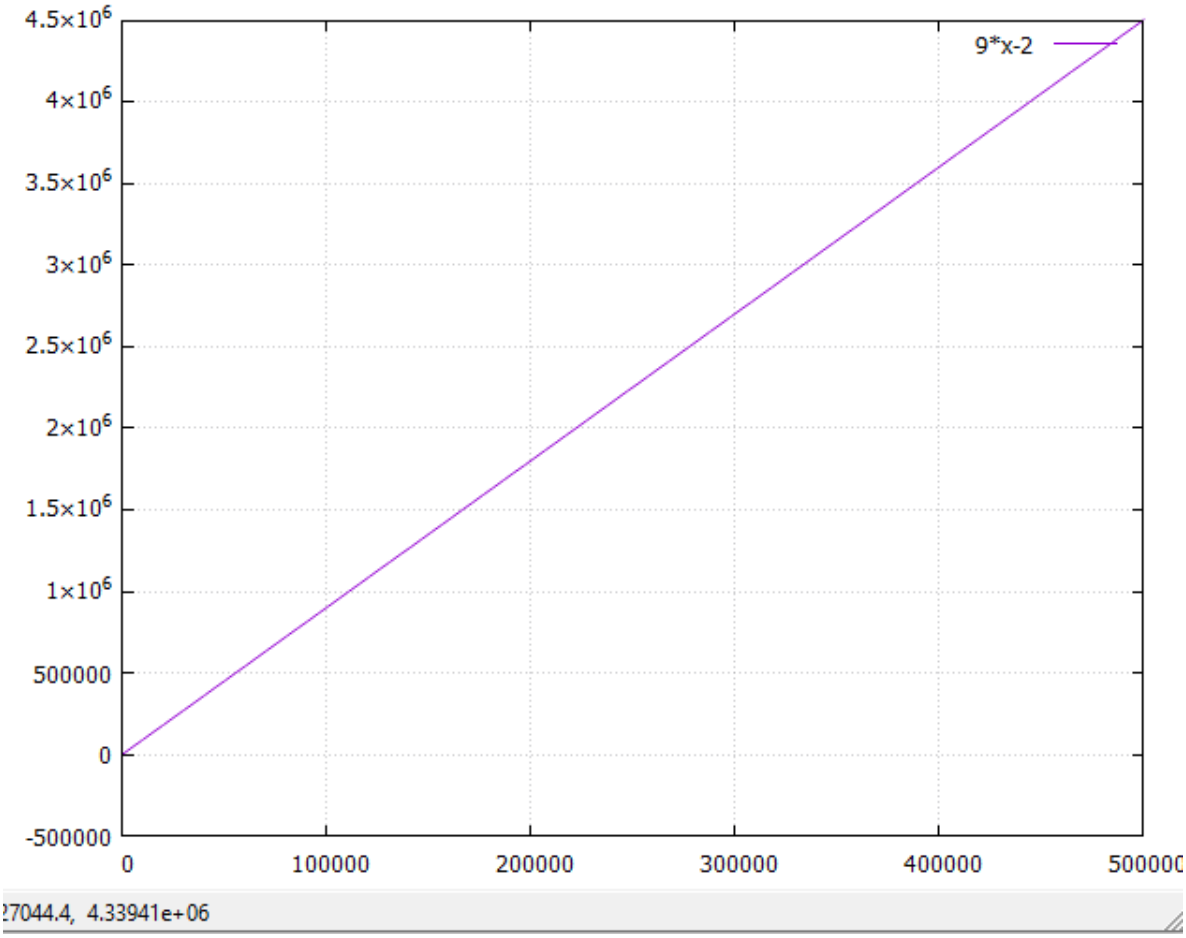
- ➔ **4 variables l, r, a y b**
- ➔ **1 variable i**

$$f(n) = 5$$

Tabla Comparativa función complejidad temporal

a, b	Resultados Teóricos	Resultados Empíricos
-1, 0	**6**	**6**
0, 2	**6**	**6**
1, 2	**6**	**6**
2, 4	**17**	**17**
3, 6	**26**	**26**
5, 10	**44**	**44**
15, 30	**136**	**136**
20, 40	**183**	**183**
100, 200	**906**	**906**
409, 818	**3680**	**3680**
500, 1000	**4509**	**4509**
593, 1800	**5335**	**5335**
1000, 2000	**9013**	**9013**
1471, 2800	**13237**	**13237**
1500, 3000	**13521**	**13521**
2801, 6000	**25207**	**25207**
3000, 6500	**27009**	**27009**
5000, 10000	**45017**	**45017**
10000, 20000	**90022**	**90022**
20000, 40000	**180027**	**180027**

Gráfica


9*x-2
4.5×10^6
4×10^6
3.5×10^6
3×10^6
2.5×10^6
2×10^6
1.5×10^6
1×10^6
500000
0
-500000
0
100000
200000
300000
400000
500000
27044.4, 4.33941e+06

Código 07

```
int main(int argc, char **argv){

    int n, nInstru = 0, i, j, temp;
    int *lista = (int *)malloc(500000*sizeof(int));


    if(argc != 2)
        exit(1);

    n = atoi(argv[1]);

    for(i=1,nInstru++;i<n;i++,nInstru++){
        nInstru++; //Condicional TRUE i
        for(j=0,nInstru++; j<n-1; j++, nInstru++){
            nInstru++; //Condicional TRUE j
            if(lista[j]>lista[j+1], nInstru+=2){

                nInstru++; // Condicional TRUE IF
                temp = lista[j];nInstru++;
                lista[j] = lista[j+1];nInstru+=2;
                lista[j+1] = temp; nInstru+=2;
                nInstru++; // Salto TRUE IF
            }

            nInstru++; // Saltos TRUE j
        }
        nInstru++; //Saltos FALSE de j
        nInstru++; //Condicional FALSE en j
        nInstru++; //Saltos TRUE en i
    }
    nInstru++; //Salto FALSE en i
    nInstru++; //Comparacion FALSE en i

    printf("\nInstrucciones: %d", nInstru);


    return 0;
```

Función complejidad temporal

```
for(i=1,nInstru++;i<n;i++,nInstru++){
```

- **(1+n+(n-1)) (asignación + comparación + aritméticas) = 2n**
- **n-1 (gT i)**
- **1 (gF i)**

```
for(j=0,nInstru++; j<n-1; j++, nInstru++){
    nInstru++; //Condicional TRUE j
```

- **(a+n+(n-1)) * (n-1) (asignaciones + comparaciones + aritméticas) * (gT i) = $2n^2$-2n**
- **n-1 (gF j)**
- **(n-1)(n-1) (gT i)(veces que se repite el for interno) = n^2-2n+1**

```
if(lista[j]>lista[j+1], nInstru+=2){

    nInstru++; // Condicional TRUE IF
    temp = lista[j];nInstru++;
    lista[j] = lista[j+1];nInstru+=2;
    lista[j+1] = temp; nInstru+=2;
    nInstru++; // Salto TRUE IF
}
```

- ➔ **En esta parte consideramos que el IF siempre será verdadero, pues no sabemos que hay dentro del arreglo de lista**
- ➔ **(1 + 3 + 3)*(n-1)(n-1) (comparaciones + aritméticas + asignaciones)*(veces que pasa por el if) = $7n^2$-14n+7**
- ➔ **(n-1)(n-1) = n^2-2n+1 (gT IF)**

Si sumamos los términos nos queda la siguiente función:

$$f(n) = 11n^2 - 16n + 8$$

Teniendo en cuenta que, para valores menores de 2, el número de operaciones siempre serán 3

Función complejidad espacial

- ➔ **2 variables i, j**
- ➔ **n variables del arreglo lista**
- ➔ **1 variable temp**

$$f(n) = n + 3$$

Tabla Comparativa función complejidad temporal

N	Resultados Teóricos	Resultados Empíricos
-1	**3**	**3**
0	**3**	**3**
1	**3**	**3**
2	**20**	**20**
3	**59**	**59**
5	**203**	**203**
15	**2243**	**2243**
20	**4088**	**4088**
100	**108408**	**108408**
409	**1833555**	**1833555**
500	**2742008**	**2742008**
593	**3858659**	**3858659**
1000	**10984008**	**10984008**
1471	**23778723**	**23778723**

1500	**24726008**	**24726008**
2801	**86256803**	**86256803**
3000	**98952008**	**98952008**
5000	**274920008**	**274920008**
10000	**1099840008**	**1099840008**
20000	**4399680008**	**104712712**

Gráfica