INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

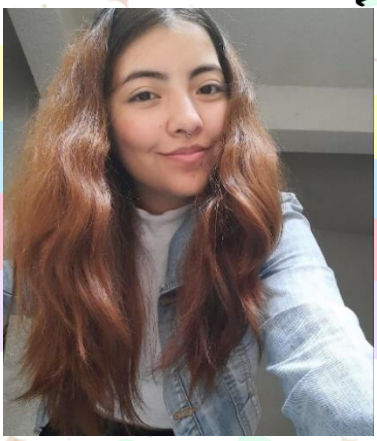
ANÁLISIS DE ALGORITMOS

EJERCICIOS 02:

DETERMINANDO FUNCIONES DE COMPLEJIDAD

ALUMNA: MÉNDEZ CASTAÑEDA AURORA

GRUPO: 3CM13



PROFESOR: FRANCO MARTÍNEZ EDGARDO ADRIÁN

Determinar para los siguientes códigos la función de complejidad temporal y espacial en términos de n. *Considere las operaciones de: asignación, aritméticas y condicionales.

- Contraste sus funciones con la prueba empírica (Verifique el conteo correcto de las instrucciones) para los 20 valores de n = {−1, 0, 1, 2, 3, 5, 15, 20, 100, 409, 500, 593, 1000, 1471, 1500, 2801, 3000, 5000, 10000, 20000}

Código 01.

```
1   for(i=1;i<n;i++)
2       for(j=n;j>1;j/=2)
3       {
4           temp = A[j];
5           A[j] = A[j+1];
6           A[j+1] = temp;
7       }
```

Para obtener la función de complejidad temporal se analizará sentencia por sentencia el algoritmo y se considerará asignaciones, aritméticas y condicionales, tenemos que

1. Del primer ciclo se tiene una asignación, n veces se realiza la condición y (n-1) el incremento.
2. Del segundo ciclo tenemos una asignación que se hace (n-1) veces, la condición $(n-1)\,(\log_2 n+1)$, esto se obtuvo dado el incremento no lineal, ya que este iría de la forma $\frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \dots, \frac{n}{2^x}=1$, despejando x obtenemos el logaritmo $\log_2 n$, le sumamos uno más que sería cuando este de falso, y esto se multiplica por (n-1), que son las veces que entra al primer ciclo. Por último, el incremento se realiza $(n-1)\log_2 n$ de igual forma.
    1. Ahora, contando lo que está dentro del segundo ciclo, tenemos una asignación que multiplicaremos por las veces que entra en ambos ciclos, es decir: $(1)(n-1)\log_2 n$.
    2. Para la siguiente sentencia tenemos una asignación y una aritmética, por lo que son 2 instrucciones por el número de veces de los ciclos: $(2)(n-1)\log_2 n$.
    3. Al igual que la anterior, se trata de dos instrucciones: $(2)(n-1)\log_2 n$.

Finalmente, sumando las expresiones que obtuvimos tenemos que:

$$f_t(n) = 1+n+(n-1)+(n-1)+(n-1)(\log_2 n+1)+(\mathrm{n}-1)\log_2 n+(1)(n-1)\log_2 n + (2)(n-1)\log_2 n+(2)(n-1)\log_2 n$$

$$f_t(n) = 1+n+n-1+n-1+n\log_2 n+n-\log_2 n-1+(\mathrm{n}-1)\log_2 n+(n-1)\log_2 n + 2(n-1)\log_2 n+2(n-1)\log_2 n$$

$f_t(n) = 4n-2+7(n-1)\log_2 n$, tomando el piso del logaritmo.

Para la función de complejidad espacial tenemos:

- 1 (variable “i”)
- 1 (variable “j”)
- 1 (variable “temp”)
- n+1 (variables del arreglo “A”)

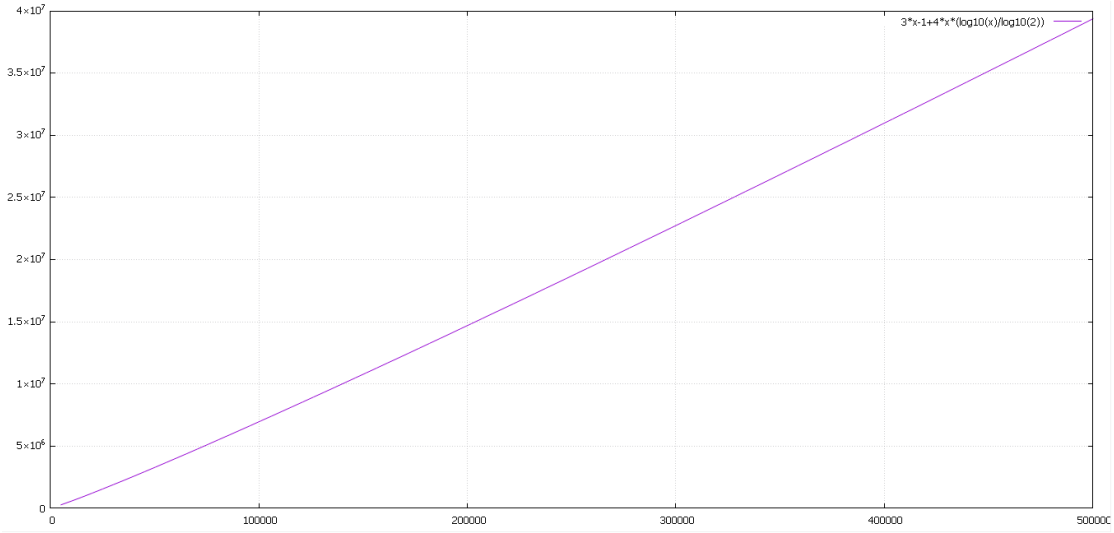
Entonces, $f_e(n) = 1+1+1+n+1 = 4+n$

Por lo tanto,
Función de complejidad temporal: $f_t(n) = 4n - 2 + 7(n-1)\log_2 n$
Función de complejidad espacial: $f_e(n) = 4 + n$

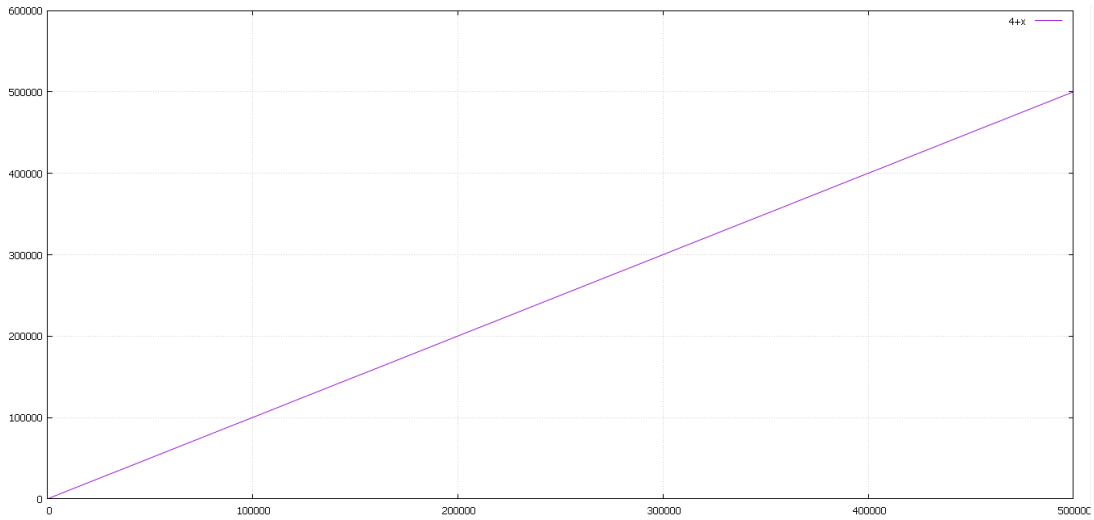
Comparando resultados teóricos y empíricos

n	Valor teórico		Valor empírico	
	f_t	f_e	$Instrucciones$	$Celdas\ de\ memoria$
-1	Indefinido	3	2	3
0	Indefinido	4	2	4
1	2	5	2	5
2	13	6	13	6
3	24	7	24	7
5	74	9	74	9
15	352	19	352	19
20	610	24	610	24
100	4556	104	4556	104
409	24482	413	24482	413
500	29942	504	29942	504
593	3966	597	39666	597
1000	66935	1004	66935	1004
1471	108782	1475	108782	1475
1500	110928	1504	110928	1504
2801	226802	2805	226802	2805
3000	242921	3004	242921	3004
5000	439914	5004	439914	5004
10000	949907	10004	949907	10004
20000	2039900	20004	2039900	20004

Gráfica de la función temporal $f_t(n) = 4n - 2 + 7(n-1)\log_2 n$



Gráfica de la función espacial $f_e(n) = 4 + n$



Código en C

```c
#include <stdio.h>
#include <stdlib.h>

int main (int narg, char** varg){
    int n_inst = 0, n_var = 0;
    int i, j, n, temp=0; n_var+=3;//variables i, j y temp

    n = atoi(varg[1]);

    int A[n+1]; n_var+= n+1; //arreglo A de tamaño n+1

    for(i=1,n_inst++; i<n; i++,n_inst++){ //asignación, condición, aritmética
        n_inst++;   //true i
        for(j=n, n_inst++; j>1; j/=2, n_inst++){
            n_inst++;   //true j
            temp = A[j]; n_inst++; //asignacion
            A[j] = A[j+1]; n_inst+=2; //asignacion, aritmetica
            A[j+1] = temp; n_inst+=2; //asignacion
        }
        n_inst++; //false j
    }
    n_inst++; //false i

    printf("Instrucciones: %d", n_inst);
    printf("\nVariables: %d",n_var);
}
```

Las instrucciones de compilación y ejecución son las siguientes:

→ gcc Codigo01.c -o Codigo01.exe
→ Codigo01.exe (valor de n)

Funcionamiento

```
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo01.exe 1
Instrucciones: 2
Variables: 5
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo01.exe 2
Instrucciones: 13
Variables: 6
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo01.exe 3
Instrucciones: 24
Variables: 7
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo01.exe 5
Instrucciones: 74
Variables: 9
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo01.exe 15
Instrucciones: 352
Variables: 19
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo01.exe 20
Instrucciones: 610
Variables: 24
```

Código 02

```
1   polinomio=0;
2   for(i=0;i<=n;i++)
3   {
4      polinomio=polinomio*z + A[n-i];
5   }
```

Análisis para obtener la función de complejidad temporal:

1. En la línea 1 nos encontramos con una asignación = 1
2. Tenemos un ciclo en el que hay una asignación, una condición que se realiza (n+2) y un incremento que pasa (n+1) veces = $1 + (n+2) + (n+1)$
3. Dentro del ciclo nos encontramos una sentencia que incluye una asignación y tres operaciones aritméticas y estas las multiplicaremos por las veces que se entra al ciclo = $4(n+1)$

Finalmente, nuestra función queda de la siguiente forma:

$f_t(n) = 1 + 1 + (n+2) + (n+1) + 4(n+1)$

$f_t(n) = 2 + n + 2 + n + 1 + 4n + 4$

$f_t(n) = 9 + 6n$

Análisis para obtener la función de complejidad espacial:

- 1 (variable “polinomio”)
- 1 (variable “i”)
- 1 (variable “z”)
- n (variables del arreglo “A”)

Entonces, $f_e(n) = 1 + 1 + 1 + n = 3 + n$

Por lo tanto,

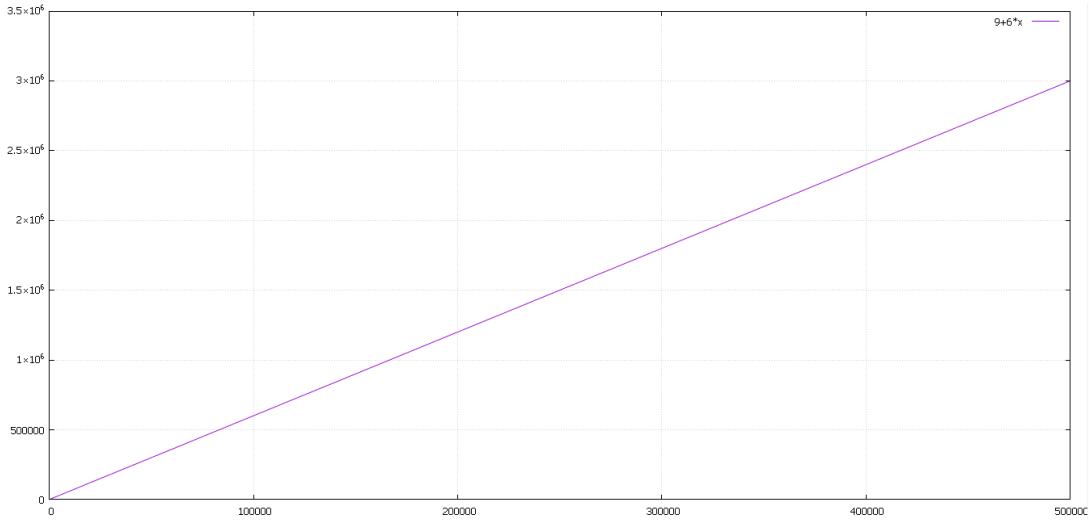
Función de complejidad temporal: $\boldsymbol{f_t(n) = 9 + 6n}$

Función de complejidad espacial: $\boldsymbol{f_e(n) = 3 + n}$

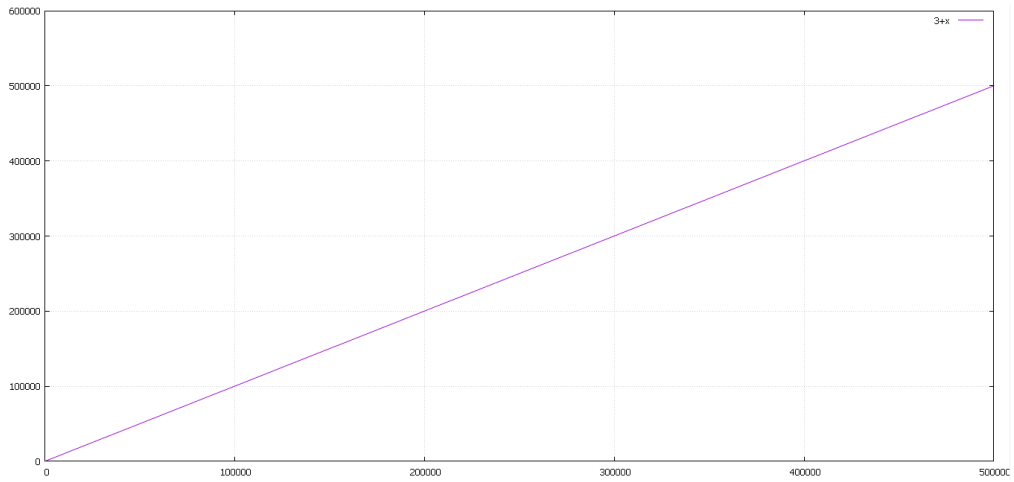
Comparando resultados teóricos y empíricos

n	Valor teórico		Valor empírico	
	f_t	f_e	$Instrucciones$	$Celdas\ de\ memoria$
-1	3	2	3	2
0	9	3	9	3
1	15	4	15	4
2	21	5	21	5
3	27	6	27	6
5	39	8	39	8
15	99	18	99	18
20	129	23	129	23
100	609	103	609	103
409	2463	412	2463	412
500	3009	503	3009	503
593	3567	596	3567	596
1000	6009	1003	6009	1003
1471	8835	1474	8835	1474
1500	9009	1503	9009	1503
2801	16815	2804	16815	2804
3000	18009	3003	18009	3003
5000	30009	5003	30009	5003
10000	60009	10003	60009	10003
20000	120009	20003	120009	20003

Gráfica de la función temporal $f_t(n) = 9 + 6n$



Gráfica de la función espacial $f_e(n) = 3 + n$



Código en C

```c
#include <stdio.h>
#include <stdlib.h>

int main (int narg, char** varg){
    int n_inst = 0, n_var = 0;
    int i; n_var++;

    int n = atoi(varg[1]);

    int polinomio = 0; n_var++; n_inst++;
    int A[n]; n_var+=n;
    int z = 1; n_var++;

    for(i=0, n_inst++; i<=n ; i++,n_inst++){
        n_inst++; //true i
        polinomio = polinomio * z + A[n-i]; n_inst+=4;
    }
    n_inst++; //false i

    printf("Instrucciones: %d", n_inst);
    printf("\nVariables: %d",n_var);
}
```

Las instrucciones de compilación y ejecución son las siguientes:

→ gcc Codigo02.c -o Codigo02.exe
→ Codigo02.exe (valor de n)

Funcionamiento

```
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo02.exe -1
Instrucciones: 3
Variables: 2
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo02.exe 0
Instrucciones: 9
Variables: 3
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo02.exe 1
Instrucciones: 15
Variables: 4
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo02.exe 2
Instrucciones: 21
Variables: 5
```

Código 03

```
1   for i = 1 to n do
2       for j = 1 to n do
3           C[i,j] = 0;
4           for k = 1 to n do
5               C[i,j] = C[i,j] + A[i,k]*B[k,j];
```

Análisis para obtener la función de complejidad temporal:

1. Para el primer ciclo que se presenta, tendremos una asignación, (n+1) de la condición y n veces que se realiza la incrementación $= 1 + (n + 1) + n$
2. Dentro del primer ciclo tenemos el segundo ciclo, este cuenta con una asignación, la condición que se realiza (n+1) veces y el incremento n veces. Además de que, por estar dentro de un ciclo, todo el ciclo se realiza las veces que entra en el primer ciclo, es decir n veces $= (1 + (n + 1) + n)(n)$
3. Luego dentro del segundo ciclo nos encontramos con una asignación, pero esta se realiza las veces que entra al segundo ciclo y por ende, al primer ciclo. $= 1(n)(n)$
4. Ahora tenemos un tercer ciclo con una asignación, una condición que se realiza (n+1) veces y su incremento n veces. Y como esta dentro de los ciclos anteriores, se repite las veces que entre a ellos $= (1 + (n + 1) + n)(n)(n)$
5. La última sentencia se trata de una asignación y dos operaciones aritméticas, y como está dentro de nuestro tercer ciclo, se realizará las veces que entre al tercer ciclo y, por ende, al primero y segundo $= (3)(n)(n)(n)$

Finalmente, nuestra función queda de la siguiente forma:

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

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

$$f_t(n) = 5n^3 + 5n^2 + 4n + 2$$

Análisis para obtener la función de complejidad espacial:

- 1 (variable “i”)
- 1 (variable “j”)
- 1 (variable “k”)
- n*n (variables de la matriz “A”)
- n*n (variables de la matriz “B”)
- n*n (variables de la matriz “C”)

Entonces, $f_e(n) = 1 + 1 + 1 + n^2 + n^2 + n^2 = 3 + 3n^2$

Por lo tanto,

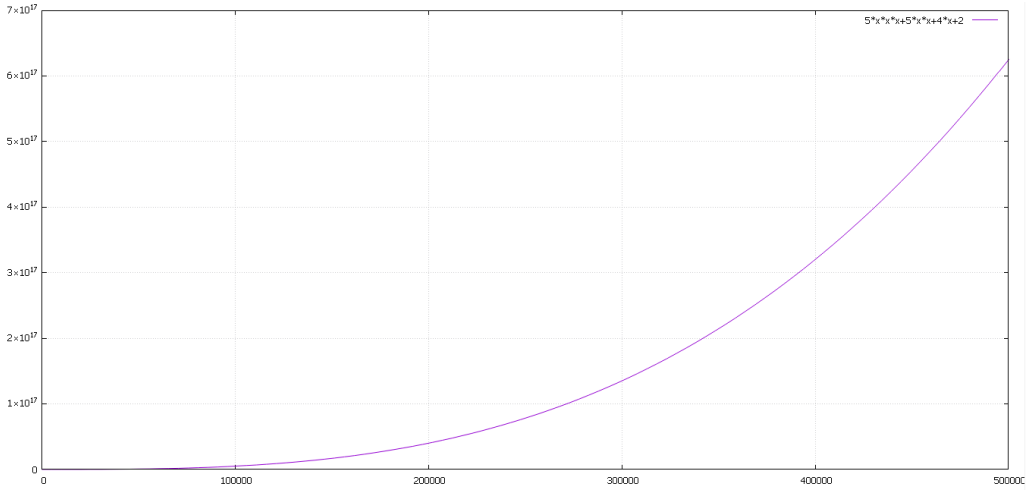
Función de complejidad temporal: $\boldsymbol{f_t(n) = 5n^3 + 5n^2 + 4n + 2}$

Función de complejidad espacial: $\boldsymbol{f_e(n) = 3 + 3n^2}$

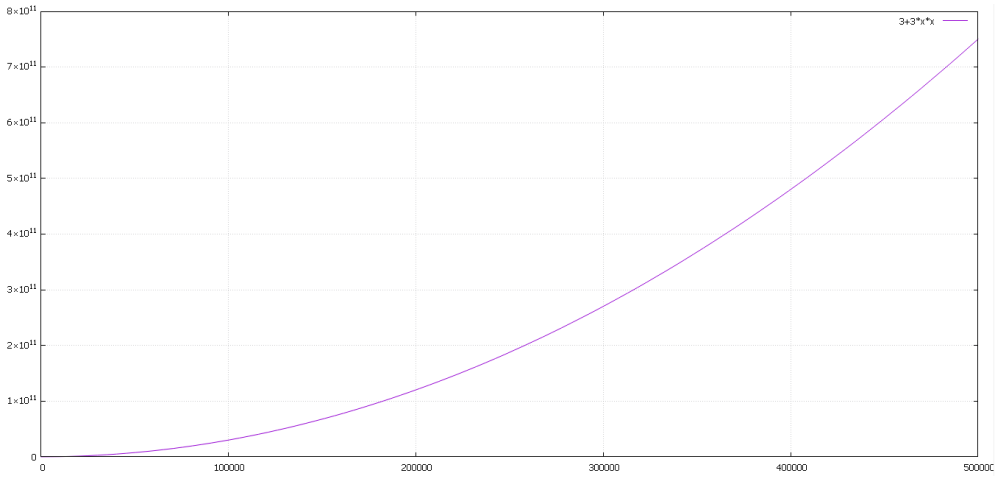
Comparando resultados teóricos y empíricos

n	Valor teórico		Valor empírico	
	f_t	f_e	f_t	f_e
-1	-2	6	2	6
0	2	3	2	3
1	16	6	16	6
2	70	15	70	15
3	194	30	194	30
5	772	78	772	78
15	18062	678	18062	678
20	42083	1203	42082	1203
100	5050402	30003	5050402	30003
409	342927688	501846	342927688	501846
500	626252002	750003	626252002	750003
593	1044399904	1054950	1044399904	1054950
1000	5005004002	3000003	710036706	3000003
1471	15925875646	6491526	-1253993538	6491526
1500	16886256002	6750003	-293613182	6750003
2801	109916881216	23536806	-1752268480	23536806
3000	135045012002	27000003	1901025826	27000003
5000	625125020002	75000003	-1940205214	75000003
10000	5000500040002	300000003	1158107458	300000003
20000	40002000080002	1200000003	-132531492	1200000003

Gráfica de la función temporal $f_t(n) = 5n^3 + 5n^2 + 4n + 2$



Gráfica de la función espacial $f_e(n) = 3 + 3n^2$



Código en C

```c
int main(int narg, char** varg){
    int n_inst = 0, n_var = 0;
    int n = atoi(varg[1]);

    int i, k, j; n_var+=3;
    int A[n][n]; n_var += n*n;
    int B[n][n]; n_var += n*n;
    int C[n][n]; n_var += n*n;

    for(i=1, n_inst++; i<=n; i++, n_inst++){
        n_inst++; //true i
        for(j=1,n_inst++; j<=n; j++,n_inst++){
            n_inst++; //true j
            C[i][j]=0; n_inst++;
            for(k=1,n_inst++; k<=n; k++,n_inst++){
                n_inst++; //true k
                C[i][j] = C[i][j] + A[i][j] * B[k][j]; n_inst+=3;
            }
            n_inst++; //false k
        }
        n_inst++; //false j
    }
    n_inst++;//false i

    printf("Instrucciones: %d", n_inst);
    printf("\nVariables: %d",n_var);
}
```

Las instrucciones de compilación y ejecución son las siguientes:

→ gcc Codigo03.c -o Codigo03.exe
→ Codigo03.exe (valor de n)

Funcionamiento

```
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo03
.exe 1
Instrucciones: 16
Variables: 6
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo03
.exe 2
Instrucciones: 70
Variables: 15
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo03
.exe 3
Instrucciones: 194
Variables: 30
```

Código 04

```
1   anterior = 1;
2   actual = 1;
3   while (n>2)
4   {
5       aux = anterior + actual;
6       anterior = actual;
7       actual = aux;
8       n = n - 1;
9   }
```

Análisis para obtener la función de complejidad temporal:

1. En las primeras dos sentencias nos encontramos con dos asignaciones $= 2$
2. Después tenemos un while que se realiza (n - 2) porque la iteración de la n es un decremento = $= (n-2)$
3. Dentro del while nos encontramos con una primera sentencia que se trata de una asignación y una operación aritmética, además de que se realiza (n – 2) veces que son las veces que entra al ciclo $= (2)(n-2)$
4. La siguiente sentencia se encuentra del ciclo entonces igual se realiza la cantidad que se cumple su condición, además de que se trata de una asignación $= (1)(n-2)$
5. Sigue una sentencia más, la cuál se trata de una asignación y aplicando lo anterior, tenemos $= (1)(n-2)$
6. Seguimos con una última sentencia que se trata de la forma en que se comporta n y la cual nos ayudo con el punto número 2, esta se trata de una asignación y una operación aritmética $= (2)(n-2)$
7. Y por último contamos el falso del ciclo $= 1$

Finalmente, nuestra función queda de la siguiente forma:

$$f_t(n) = 2 + (n-2) + (2)(n-2) + (1)(n-2) + (1)(n-2) + (2)(n-2) + 1$$

$$f_t(n) = 2 + n - 2 + 2n - 4 + n - 2 + n - 2 + 2n - 4 + 1$$

$$f_t(n) = 7n - 11$$

Análisis para obtener la función de complejidad espacial:

- 1 (variable “anterior”)
- 1 (variable “actual”)
- 1 (variable “aux”)

Entonces, $f_e(n) = 1 + 1 + 1 = 3$

Por lo tanto,

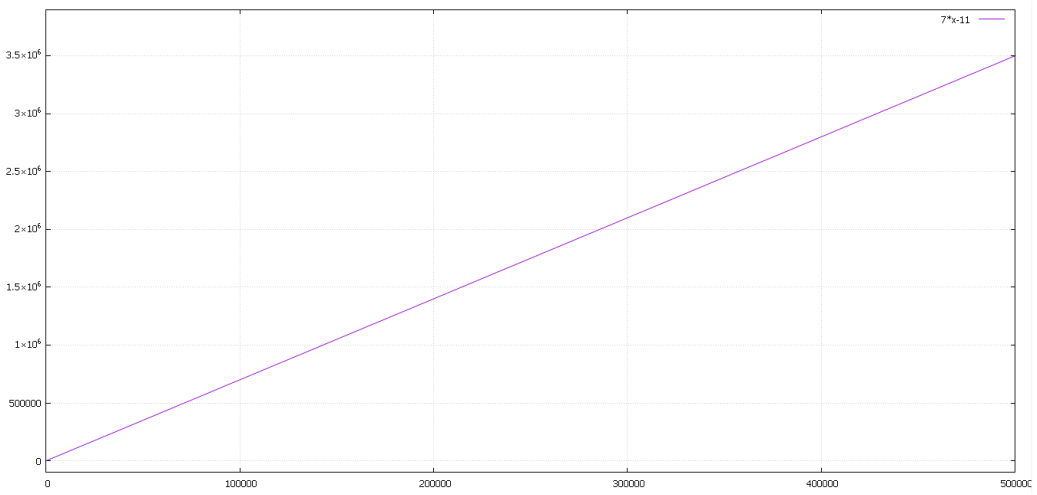
Función de complejidad temporal: $\boldsymbol{f_t(n) = 7n - 11}$

Función de complejidad espacial: $\boldsymbol{f_e(n) = 3}$

Comparando resultados teóricos y empíricos

n	Valor teórico		Valor empírico	
	f_t	f_e	f_t	f_e
-1	-18	3	3	3
0	-11	3	3	3
1	3	3	-4	3
2	3	3	3	3
3	10	3	10	3
5	24	3	24	3
15	94	3	94	3
20	129	3	129	3
100	689	3	689	3
409	2852	3	2852	3
500	3489	3	3489	3
593	4140	3	4140	3
1000	6989	3	6989	3
1471	10286	3	10286	3
1500	10489	3	10489	3
2801	19596	3	29596	3
3000	20989	3	20989	3
5000	34989	3	34989	3
10000	69989	3	69989	3
20000	139989	3	139989	3

Gráfica de la función temporal $f_t(n) = 7n - 11$



Código en C

```c
#include <stdio.h>
#include <stdlib.h>

int main(int narg, char** varg){
    int n_inst = 0, n_var = 0;
    int n = atoi(varg[1]);

    int anterior, actual, aux; n_var+=3;

    anterior = 1; n_inst++;
    actual = 1; n_inst++;

    while(n > 2){
        n_inst++;//true while
        aux = anterior + actual; n_inst+=2;
        anterior = actual; n_inst++;
        actual = aux; n_inst++;
        n = n - 1; n_inst+=2;
    }
    n_inst++; //false while

    printf("Instrucciones: %d", n_inst);
    printf("\nVariables: %d",n_var);
}
```

Las instrucciones de compilación y ejecución son las siguientes:

→ gcc Codigo04.c -o Codigo04.exe
→ Codigo04.exe (valor de n)

Funcionamiento

```
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo04
.exe 3
Instrucciones: 10
Variables: 3
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo04
.exe 100
Instrucciones: 689
Variables: 3
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo04
.exe 1000
Instrucciones: 6989
Variables: 3
```

Código 05

```
1   for (i = n - 1;j=0;i>=0;i--,j++)
2      s2[j]= s[i];
3   for (k = 0,k<n;k++)
4      s[i]= s2[i];
```

Se encontraron dos errores en el código, el primero es el ; que esta entre la inicialización de la variable i y la variable j, el otro error esta en el segundo for que aparentemente se ve independiente del primero, sin embargo, en la sentencia dentro del for se utilizan variables que son parte del primer for, por lo tanto, consideraré al segundo for como parte del primero.

Análisis para obtener la función de complejidad temporal:

1. El primer ciclo cuenta con dos asignaciones, la condición se realiza (n+1) veces, el decremento de "i" n veces y el incremento de "j" n veces $= 1 + 1 + (n + 1) + n + n$
2. Dentro del ciclo tenemos una asignación y se realiza n veces $= (1)(n)$
3. Luego está un segundo ciclo dentro del primero, este tiene una asignación, su condición se realiza (n+1) veces y su incremento n veces, además de que en conjunto se realiza n veces $= (1 + (n + 1) + n)(n)$
4. Dentro del segundo ciclo tenemos una asignación, que se realiza n veces del segundo ciclo y n veces del primer ciclo $= (1)(n)(n)$

Finalmente, nuestra función queda de la siguiente forma:

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

$$f_t(n) = 1 + 1 + n + 1 + n + n + n + (2n + 2)(n) + n^2$$

$$f_t(n) = 1 + 1 + n + 1 + n + n + n + 2n^2 + 2n + n^2$$

$$f_t(n) = 3n^2 + 6n + 3$$

Análisis para obtener la función de complejidad espacial:

- 1 (variable "i")
- 1 (variable "j")
- n (variables del arreglo "s")
- n (variables del arreglo "s2")

Entonces, $f_e(n) = 1 + 1 + 1 + n + n = 3 + 2n$

Por lo tanto,

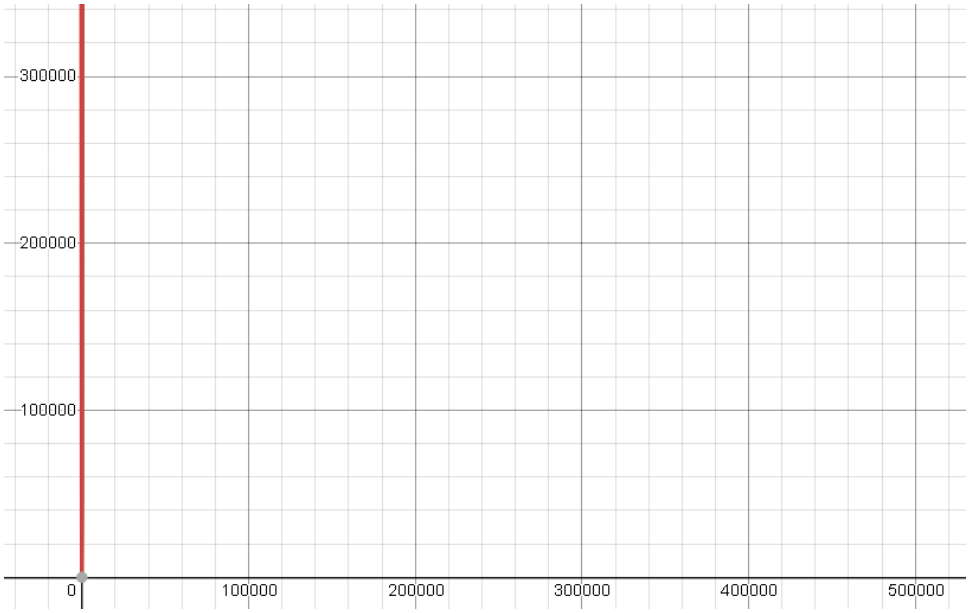
Función de complejidad temporal: $\boldsymbol{f_t(n) = 3n^2 + 6n + 3}$

Función de complejidad espacial: $\boldsymbol{f_e(n) = 3 + 2n}$

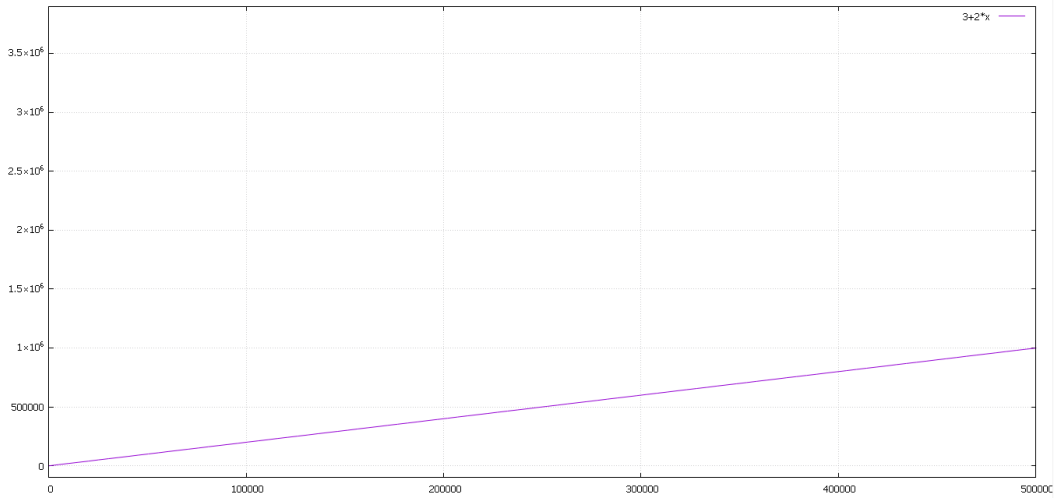
Comparando resultados teóricos y empíricos

n	Valor teórico		Valor empírico	
	f_t	f_e	f_t	f_e
-1	3	1	3	1
0	3	3	3	3
1	12	5	12	5
2	27	7	27	7
3	48	9	48	9
5	108	13	108	13
15	768	33	768	33
20	1323	45	1323	45
100	30603	203	30603	203
409	504300	821	504300	821
500	753003	1003	753003	1003
593	1058508	1189	1058508	1189
1000	3006003	2003	3006003	2003
1471	6500352	2945	6500352	2945
1500	6759003	3003	6759003	3003
2801	23553612	5605	23553612	5605
3000	27018003	6003	27018003	6003
5000	75030003	10003	75030003	10003
10000	300060003	20003	300060003	20003
20000	1200120003	40003	1200120003	40003

Gráfica de la función temporal $f_t(n) = 3n^2 + 6n + 3$



Gráfica de la función espacial $f_e(n) = 3 + 2n$



Código en C

```c
#include <stdio.h>
#include <stdlib.h>

int main(int narg, char** varg){
    int n_inst = 0, n_var = 0;
    int n = atoi(varg[1]);

    int i,j,k; n_var+=3;
    int s[n]; n_var+= n;
    int s2[n]; n_var+= n;

    for(i=n-1,j=0,n_inst+=2; i>=0; i--,j++,n_inst+=2){
        n_inst++; //true i
        s2[j] = s[j]; n_inst++;
        for(k=0,n_inst++; k<n; k++,n_inst++){
            n_inst++; //true k
            s[i] = s2[j]; n_inst++;
        }
        n_inst++; //false k
    }
    n_inst++; //false i

    printf("Instrucciones: %d", n_inst);
    printf("\nVariables: %d",n_var);
}
```

Las instrucciones de compilación y ejecución son las siguientes:

→ gcc Codigo05.c -o Codigo05.exe
→ Codigo05.exe (valor de n)

Funcionamiento

```
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo05
.exe 1
Instrucciones: 12
Variables: 5
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo05
.exe 2
Instrucciones: 27
Variables: 7
C:\Users\jorge\Desktop\ESCOM\5to Semestre\Algoritmos\Programas\Ejercicio02>Codigo05
.exe 20000
Instrucciones: 1200120003
Variables: 40003
```

Código 06

```
1   l=(a<b)?a:b;
2   r=1;
3   for(i=2;i<=l;i++)
4   {
5      if(a%i==0&&b%i==0)
6         r=i;
7   }
```

Observando el programa anterior, pude notar que no podríamos obtener la función de complejidad en términos de n, ya que este no depende de ella. Por lo tanto, puedo concluir que no existe una función de complejidad temporal y espacial en términos de n para este código.

Código 07

```
1   for (i=1; i<n; i++)
2   {
3      for j=0 ; j<n - 1; j++)
4      {
5      if (lista[j] > lista[j+1])
6         {
7            temp = lista[j];
8            lista[j] = lista[j+1];
9            lista[j+1] = temp;
10        }
11     }
12  }
```

Como podemos observar, bien podríamos definir cuantas veces se realiza los dos ciclos del código, sin embargo, a la hora de llegar a la sentencia dónde se encuentra una condicional es dónde nos encontramos con un problema diferente. Lo que hace la condición es comparar dos elementos de un arreglo y mientras se cumpla, se ejecutan algunas asignaciones con tales elementos, el problema entra aquí, ya que cada elemento del arreglo podría ser cualquier número, nos encontramos con infinitas posibilidades y es por eso por lo que no podríamos obtener una función de complejidad en términos de n para este problema.