Carrera **INGENIERIA EN INFORMATICA**

Asignatura 3652 - Programación Avanzada

Tema Complejidad Computacional

Unidad 2

Objetivo comprender y desarrollar conceptos de complejidad computacional. Comparar el orden de diversos algoritmos

Competencia/s a desarrollar

Genéricas

- **Competencias tecnológicas**

1. **Identificación, formulación y resolución de problemas de ingeniería en sistemas de información/informática.**
2. **Concepción, diseño y desarrollo de proyectos de ingeniería en sistemas de información / informática.**
3. **Gestión, planificación, ejecución y control de proyectos de ingeniería en sistemas de información / informática.**
4. **Utilización de técnicas y herramientas de aplicación en la ingeniería en sistemas de información / informática.**

- **Competencias sociales, políticas y actitudinales**

5. **Desempeño en equipos de trabajo.**
6. **Comunicación efectiva.**
7. **Actuación profesional ética y responsable.**
8. **Aprendizaje continuo.**
9. **Desarrollo de una actitud profesional emprendedora.**

Descripción de la Actividad

Tiempo estimado de resolución:

Metodología:
Resolución de problemas de programación con una guía autoexplicativa.

Forma de entrega:
Entrega presencial opcional durante el período de clase destinado a tal fin.

Metodología de Corrección y Retroalimentación:
Se realizará una puesta en común de ejercicios seleccionados. Se utilizará el código provisto por estudiantes voluntarios para analizar distintas posibles resoluciones. Se utilizarán técnicas de pair programming y mob programming para resolver los ejercicios que se seleccionen para hacer en la clase.

Guía de Ejercicios - Complejidad computacional

1) Para cada una de los siguientes conjuntos de funciones, indique su orden de crecimiento con la notación O(f(n)) y ordene por velocidad de crecimiento (en sentido creciente)

a) $6 + n^2$; $4 + n + 2\log(n)$; $n^2 + 3 * 2^n$; $5n^3$
b) $5n$; $n^2 + 5 * 2^n$; 9; $7*n$
c) 2; $\log(n) + 2n$; $5n + 2^n$; n

2) Cuales de las siguientes afirmaciones son verdaderas. Justifique.

a) $n \in O(1)$
b) $500 \in O(n)$
c) $4n \in O(4^n)$
d) $n^2 + 1/5n^3 \in O(n^3)$
e) $n*(n+2)^2/3 \in O(n^2)$
f) $2 \log_2 n \in O(\log_4 n)$
g) $3^n \in O(2^n)$

3) Para cada uno de los siguientes segmentos de algoritmos analice sus complejidades usando notación O(f(n))

a)
```
Leer(a)
n ← a*a
c ← 0
MIENTRAS (a>1) HACER
      a ← a/2
      PARA i=1,n HACER
            c ← c*2
Escribir(c)
```

b)
```
k ← n
PARA i=1,n HACER
      b ← i
      PARA k=k,b (paso=-1) HACER
         SI a(k-1)>a(k) ENTONCES
               Aux ← a(k-1)
               a(k-1) ← a(k)
               a(k) ← Aux
```

4) Suponiendo que se sabe que el tiempo de ejecución de un algoritmo pertenece a O(NlogN) y que el de otro algoritmo pertenece a $O(N^2)$, se pide:

a) ¿Que se puede decir sobre el rendimiento relativo de estos algoritmos ?

b) Suponiendo que ambos algoritmos resuelvan el mismo problema, enumere en qué casos implementaría uno y en cuales implementaría el otro.

c) en el caso de que ambos algoritmos sean de ordenación interna, enumere situaciones donde sería válido usar el algoritmo cuadrático.

5) Sea la siguiente tabla:

T(n) Tiempo de ejecución proporcional a:	Tamaño del problema para 10^3seg N1	Tamaño del problema para 10^4seg N2	Incremento N2/N1
100 n	10	100	10
$5n^2$	14		
$n^3 / 2$	12		
2^n	10		

Tiempo de ejecución T*(n)*	Tamaño máximo de problema para 10^3 seg.	Tamaño máximo de problema para 10^4 seg.	Incremento en el tamaño máximo de problema
100n	10	100	10.0
$5n^2$	14	45	3.2
$n^3/2$	12	27	2.3
2^n	10	13	1.3

En la tercera columna de la tabla se puede apreciar una superioridad evidente del programa *0(n)*, éste permite un aumento del 1000% en el tamaño de problema para un incremento del 1000% en la rapidez del computador. Se observa que los programas $0(n^3)$ y $0(n^2)$ permiten aumentos de 230% y 320%, respectivamente, en el tamaño de problema, para un incremento del 1000% en la rapidez del computador. Estas razones se mantendrán vigentes para incrementos adicionales en la rapidez del computador.

Mientras exista la necesidad de resolver problemas cada vez más grandes, se producirá una situación casi paradójica. A medida que los computadores aumenten su rapidez y disminuyan su precio, como con toda seguridad seguirá sucediendo, también el deseo de resolver problemas más grandes y complejos seguirá creciendo. Así, la importancia del descubrimiento y el empleo de algoritmos eficientes (aquellos cuyas velocidades de crecimiento sean pequeñas) irá en aumento, en lugar de disminuir.

Se pide: a) completar los datos que faltan.

b) sacar alguna conclusión, observando los respectivos incrementos.

6) Dada la siguiente tabla

O(n)	1	100 veces más rápido	1000 veces más rápido
n	N_1	$100N_1$	$1000N_1$
n^2	N_2		
n^3	N_3		
n^5	N_4		
2^n	N_5		
3^n	N_6		

Donde N_i es el tamaño máximo del problema que se puede resolver en una computadora dada. Se pide completar la tabla precedente en función de N_i, suponiendo que se corre el mismo algoritmo en una máquina 100 veces más veloz y la última columna en una máquina 1000 veces más veloz.

7) Dada la siguiente tabla se pide calcular el N_i para un minuto y para una hora.

	N_i / seg	N_i / minuto	N_i / hs.
n	1000	60000	360000
n log n	140		
n^2	31		
n^3	10		
2^n	9		

8) Ídem al ejercicio 5 pero se pide calcular el Ni para una máquina 10 veces más rápida.

	N_i / seg	10 veces más rápido
n	1000	

n log n	140	
n^2	31	
n^3	10	
2^n	9	

9) Para cada uno de los algoritmos de ordenamiento compare el gráfico obtenido con el O(n) correspondiente. (Analice teniendo en cuenta el mejor caso, el caso promedio y el peor caso).

10) Calcule la complejidad del algoritmo “calcular el factorial de n” con recursividad y sin recursividad. Realice mediciones de tiempo.

11) Realice una ficha técnica de su entorno de programación, indicando las características del mismo y cuanto tarda en realizar las siguientes operaciones:

a) 100.000.000 de sumas.
b) 100.000.000 de multiplicaciones
c) 100.000.000 de divisiones
d) 100.000.000 de comparaciones
e) 100.000.000 de asignaciones

12)

	N					
F(n)	10	20	30	40	50	60
N						
N^2						

N^3						
N^5						
2^N						
N!						

Suponga que en la tabla anterior, f(n) equivale a la cantidad de operaciones que realiza un algoritmo. Sabiendo que en una computadora determinada se realizan 100.000.000 de operaciones por segundo complete la tabla dada, indicando en cada celda el tiempo que requeriría cada caso. Analice el resultado. ¿Qué sucede si logramos una máquina un millón de veces más rápida?

13) Si se sabe que el algoritmo de selección tardó 10 segundos en ordenar 10000 elementos. ¿Cuántos elementos podría ordenar en el triple de tiempo? ¿Cuánto tardaría en ordenar el triple de elementos? ¿Cuánto tardaría en ordenar el triple de elementos en una máquina 3 veces más rápida?
14) Suponga que debe ordenar un vector por el método de Selección. Su entrada es de 2000 elementos y en su PC obtuvo un tiempo de 10 segundos. Si debiera ordenar un vector de 5000 entradas, cuál sería el tiempo estimado de ejecución? Cuál sería el tamaño máximo de vector que podría ordenar en 40 segundos?
    Resuelva este problema reemplazando el método de ordenamiento por cada uno de los métodos vistos.
15) Analice la complejidad computacional del algoritmo de búsqueda binaria. Compárelo con el de búsqueda lineal.
16) Se tiene un arreglo desordenado cuyo registro se compone de 10% clave y 90% datos, si se sabe que en una máquina determinada el algoritmo de burbujeo tardó 10 segundos en ordenar 4000 elementos, cuánto tardaría en ordenar 8000 elementos?, Cuánto tardaría el algoritmo de inserción en ordenar ambos vectores en la misma máquina?
17) El algoritmo de selección ordenó un vector de 10000 enteros en 5 seg. ¿Cuál será el

tamaño del vector que espera ordenar en 20 seg. ? Justifique. ¿Qué sucede si ordena ambos vectores en una máquina 5 veces más rápida?

18) Utilizando el algoritmo de inserción se ordenó en un tiempo T1, N1 elementos. En un tiempo de 2xT1 se ordenó un vector de 2xN1 elementos. Explique las condiciones bajo las cuales pudieron hacerse las pruebas.

19) Utilizando el algoritmo de QuickSort se ordenó un vector de 1 millón de elementos en un tiempo T1. ¿Cuántos elementos espera ordenar en un tiempo 2xT1?