Objetivo: suministrar al alumno de un acopio de términos referenciados al tema de corrección y complejidad de algoritmos, que deberán completar y fortalecer con la bibliografía recomendada, junto con el cálculo de la complejidad de algoritmos conocidos en el desarrollo de unidades anteriores.

Definición de especificación formal de un algoritmo: es la expresión del efecto de dicho algoritmo que va a tener sobre determinados datos de entrada e indicar en que consisten dichos datos de entrada. Por ej: la suma de los N primeros números naturales

Para todo N perteneciente al conjunto natural la SUMA(N) que es la sumatoria de los valores que toma una variable I desde 1 hasta N inclusive.

Definición de algoritmo correcto: es aquel que satisface la especificación formal dada para su realización.

Clasificación de algoritmo correcto de acuerdo a su cumplimiento:

a) Parcial: garantiza que si termina el algoritmo cumple la especificación.
b) Total: garantiza que el algoritmo termina sobre cualquier conjunto de datos válidos.

Concepto de Verificación: es el mecanismo por el que se valida si un algoritmo es correcto totalmente.

a) Prueba de escritorio (parcial)
b) Prueba formal (total)

Complejidad de los algoritmos

Para resolver un problema computacional, se desarrollan varios algoritmos correctos y se debe elegir el mejor de ellos. Se analiza la cantidad de recursos de la computadora utilizados por cada algoritmo, ó lo mismo que decir lo eficiente ó complejo que sea. Se busca medir la eficiencia para comparar algoritmos.
Existe un análisis de la complejidad, en el que se observan dos enfoques diferentes:

a) *Enfoque empírico* : se ejecutan en máquina y se comparan resultados
b) *Enfoque teórico* : se determina matemáticamente a priori los recursos utilizados

Para realizar la evaluación de la eficiencia, se determinan:

1- *Eficiencia en memoria*: llamada complejidad espacial, que representas la cantidad de almacenamiento necesario.
2- *Eficiencia en tiempo*: como no se ejecuta no existe una unidad determinada para medir la eficiencia teórica, por ello se utiliza el principio de invarianza, que plantea que dos versiones del mismo algoritmo no difieren en su eficiencia en más que una constante:
T1 (n) <= C T2(n) donde n es grande y C es positiva y dependerá de la computadora.

Notación O( ) :
F (n) es de orden O (g(n)) si y solo si se cumple que para una constante positiva C y N_0, ambos independientes de N tales que :
F (N) <= G (N), para todo N >= N_0

Por ejemplo : F(N) = 3N3 + 2 N2 es O (N3)
Donde G (N) = N3 N= 0 y C = 5
$3N^3 + 2 N^2 <= 5 N^3$ para todo N >= 0

Funciones de complejidad en tiempo más usuales:

O (N log(N)): aparece en algoritmo con recursión, ej: ordenamiento rápido y se considera una complejidad buena.
O (N^2): complejidad cuadrática, aparece en bucle anidados, por ejemplo ordenamiento directo por selección.
O (log(N)): Complejidad logarítmica, aparece en algoritmos con iteración, por ej: búsqueda binaria)

Cálculo de la complejidad de algunos algoritmos:

Ejemplo 1: *Método de ordenamiento rápido*

El algoritmo de ordenamiento rápido está diseñado con recursión, basado en dividir el rango de valores a ordenar en partes iguales y luego realizar ordenamiento dentro de cada parte.

Por ejemplo si se tiene un vector de N valores, donde $N = 2^K$, para algún K, siendo K el número de divisiones en que se realizará el algoritmo, definiendo que $K = \log_2 (N)$:

En la 1º fase se realizarán N comparaciones
En la 2º fase se habrán dividido en 2 subrangos de tamaño aproximado N/2, siendo la cantidad de comparaciones para estas 2 partes, la suma de comparaciones realizadas en cada una de las partes:
$2 * (N /2) = N$
En la 3º fase se habrán dividido cada una de las partes en dos partes iguales, es decir que la cantidad total de comparaciones será : $4 * (N / 4) = N$
En general de tener K divisiones se totalizarán K * N comparaciones. Si llamamos a m la cantidad e comparaciones:
$m = K * N$
Reemplazando a K por su función en N:
$m = \log_2 (N) * N$

Por supuesto que si la división de las partes no se realiza en parte iguales ó similares, la eficiencia no será la misma y se aproximará a la eficiencia de los métodos de ordenamiento simples.

Ejemplo 2: *Método de ordenamiento simple por selección*

El algoritmo se define con una estructura de repetición incondicional para controlar la cantidad de pasadas, que de tener N datos a ordenar, serán N-1 bucles ó pasadas. En cada pasada se realizarán comparaciones entre los datos y como resultado de las mismas se realizarán movimientos de datos ó no.

Análisis por las comparaciones (son cantidades fijas e independientes del N y del orden de datos):

1º pasada se realizan N-1 comparaciones entre datos
2º pasada realizan N-2 comparaciones entre datos
3º pasada realizan N-3 comparaciones entre datos
…
(N-1)º pasada se realiza 1 comparación entre datos

Por lo tanto si se suman las cantidades de comparaciones en total se obtiene la sucesión matemática:
$/N-1) + (N-2) + \ldots + 1 = (N * (N -1)) / 2 = N^2 / 2 - N / 2$

Análisis por los movimientos de datos:

a) Existe una situación deseada, que no se tengan que realizar cambio alguno : 0 movimientos

b) Se supone el peor caso, en el que los datos están totalmente desordenados, por lo tanto en cada comparación se requiere mover datos (3 movimientos por vez), por lo tanto se obtendría:
3 * cantidad de comparaciones = 3 * (N^2 -N) / 2 = 3/2 * (N^2 - N)

c) Se calcula el promedio entre a) y b) como situación esperada: ¾ * (N^2 – N) =
¾ N^2 – ¾ N < ¾ N^2 para N >=0 por lo tanto **O (N) = ¾ N^2**
De compararse con el método de ordenamiento de Burbuja, al tener en lugar de una estructura de control de repetición incondicional para controlar las pasadas, utiliza una estructura de repetición condicional, se puede estimar que sea la mitad de repeticiones, por lo tanto se consideraría una cantidad promedio de N^2 / 4 – N / 4 comparaciones. Por ello es que se elige utilizar Burbuja en lugar de selección, para un N grande.

Ejemplo 3: ***Método de búsqueda binaria***

El algoritmo define dividir el rango en su punto medio , se compara entre el valor buscado y el dato ubicado en esa posición, de ser diferentes se continúa buscado el dato en la parte donde existen posibilidad de localizarlo, entre los valores ordenados. Por lo tanto se puede realizar el siguiente análisis:

Para un conjunto de N elementos, se observa que para ciertos valores de n:

1 elemento	se realiza 1 comparación como máximo
3 elementos	se realizarían 2 comparaciones como máximo
7 elementos	se realizarían 3 comparaciones como máximo
15 elementos	se realizarían 4 comparaciones como máximo

En general para n se realizan (2^m – 1) comparaciones con m >=1 , siendo m la cantidad de comparaciones. Para despejar el valor de m se realizan los siguientes pasos:

$$n + 1 = (2^m - 1) + 1 = 2^m$$
$$\log_2 (n + 1) = \log_2 (2^m) = m$$

Entonces si se estima:

a) El mejor caso de búsqueda, realizando una sola comparación
b) El peor caso suponiendo que se realizan $\log_2 (n + 1)$

c) Se calcula el promedio entre ambos casos : $(1 + \log_2 (n + 1)) / 2$
=> ½ + $\log_2 (n + 1) / 2 < \log_2 (n + 1) / 2$

Asumiendo que siempre se podrá obtener un número n que sea aproximadamente (2^m - 1), para un N grande **O (N) = $\log_2$ (n + 1) / 2**

Si se compara este algoritmo con el algoritmo de búsqueda secuencial, sobre un conjunto de N valores, se promedia una cantidad de **(n + 1) / 2** comparaciones, por lo tanto para un N muy grande, el log2 (N) es mucho menor que el valor N y por ello es que de tener un conjunto de valores ordenados en el que se debe buscar un dato, se debe utilizar por eficiente la búsqueda binaria y no la secuencial.

- Final de la guía -