Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignaturas: Diseño de Algoritmos - Algoritmos II
Primer Cuatrimestre de 2017

Guía Práctica No. 1: Revisión

La resolución de esta práctica debe finalizar antes de la semana que comienza el **27 de Marzo de 2017**.

Ej. 1. Lea los capítulos 1 y 2 de *Introduction to the Design and Analysis of Algorithms* (Levitin 2003).

Ej. 2. Acceda al sitio `http://codeboard.io`, y cree una cuenta. Suba los datos de su cuenta al foro del moodle de la asignatura, accesible en `http://dc.exa.unrc.edu.ar/moodle`.

Ej. 3. Para cada uno de los siguientes algoritmos, indique qué problema resuelve y cuál es su tiempo de ejecución en el peor caso.

- Mergesort — ordena una secuencia de elementos, y su orden en peor caso es $O(n \log n)$.
- Quicksort
- Algoritmo de Dijkstra
- Algoritmo de Warshall
- Algoritmo de Kruskal
- Búsqueda binaria
- Algoritmo de Euclides
- Búsqueda a lo ancho

Ej. 4. Para cada uno de los siguientes problemas, indique cuán eficientemente pueden resolverse:

- Ordenar una secuencia — Puede resolverse en $O(n \log n)$
- Buscar un elemento en un conjunto
- Problema de la exponenciación (calcular a^n)
- Eliminación Gaussiana
- Encontrar par de elementos más cercanos en un conjunto de puntos del plano
- Composición relacional
- Ordenamiento topológico
- Determinar existencia de ciclos eulerianos en un grafo

Ej. 5. Cuáles de los algoritmos y problemas de los ejercicios 3 y 4 consideraría *numéricos* según la definición del capítulo 1 de *Introduction to the Design and Analysis of Algorithms* (Levitin 2003)?

Ej. 6. Acceda a su cuenta en codeboard y resuelva los acertijos propuestos en los siguientes proyectos:

- **Acertijo1:** https://codeboard.io/projects/15244
- **Acertijo2:** https://codeboard.io/projects/15247

Utilice los test definidos en cada proyecto para comparar su resultado con el de la implementación secreta. Analice las diferencias y modifique su código hasta que los resultados coincidan con los de la solución escondida.
A travs de la opción Submit de codeboard, puede realizar la entrega de las soluciones implementadas y recibirá una devolución por parte de los docentes de la asignatura.

Ej. 7. Una forma de implementar el algoritmo de ordenamiento *heapsort* es mediante el uso de una implementación eficiente de colas de prioridades: los elementos de la secuencia a ordenar se trasladan a la cola de prioridades, y luego se construye incrementalmente una secuencia ordenada mediante la extracción (ordenada) de elementos de la cola de prioridades. Implemente *heapsort* en Java, siguiendo este proceso. Utilice su cuenta en codeboard para acceder al template de este algoritmo donde podrá encontrar también una serie de test para probar su programa (https://codeboard.io/projects/772) .

Ej. 8. Diseñe e implemente en Haskell un algoritmo que tome como parámetro dos palabras y decida si las mismas son anagramas, es decir, si una puede obtenerse a partir de la otra mediante permutación de caracteres. Utilice su cuenta en codeboard para acceder al template de este algoritmo donde podrá encontrar también una serie de test para probar su programa (https://codeboard.io/projects/834).

Ej. 9. Demuestre que $\gcd(m, n) = \gcd(n, m \bmod n)$, para todo par de enteros positivos m y n.

Ej. 10. Considere el algoritmo de ordenamiento *insertion sort*, según se describe a continuación:

```
for i = 1 to length(A) - 1
    x = A[i]
    j = i
    while j > 0 and A[j-1] > x
        A[j] = A[j-1]
        j = j - 1
    A[j] = x
```

Equipe este algoritmo con pre y post condiciones adecuadas, y demuestre que el mismo resuelve correctamente el problema de ordenar una secuencia.

Ej. 11. Implemente un algoritmo que, dado un número entero n, retorne todos los números primos menores o iguales que n. Implemente este algoritmo usando la definición de número primo, es decir, evaluando uno a uno los elementos de $[2..n]$, y comprobando que todos los valores menores (excepto el uno) no los dividen. Realice el análisis de tiempo de ejecución, en peor caso, de este algoritmo.

Implemente luego otra solución a este problema, basada en la *Criba de Eratóstenes* (véase la sección 1.1 de "Introduction to the Design and Analysis of Algorithms" (Levitin 2003)). Corra experimentos para comparar la eficiencia de ambos algoritmos.

Ej. 12. Realice experimentos para comparar los tiempos de corrida de los tres algoritmos para computar el máximo común divisor presentados en la sección 1.1 de "Introduction to the Design and Analysis of Algorithms" (Levitin 2003).