Examen parcial 1 de Análisis y Diseño de Algoritmos. Fecha: 23 de Septiembre de 2017.

Nombre del alumno: **David Alfonso Velasco Sedano**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

I.- ¿Cuál es el método de ordenamiento que mejor cumple con cada característica citada? Fundamenta tu respuesta de manera breve y clara. *Valor: 30 puntos. (5 puntos por pregunta)*

1. Es de fácil programación y adecuado cuando los movimientos son más costosos que las comparaciones. Funciona con cualquier tipo de dato comparable, no sólo con enteros.

**Nombre del algoritmo.**

Fundamentación.

1. Basa su eficiencia en que la distancia entre los elementos a intercambiar crece de manera exponencial conforme se alejan del elemento 0 de la lista.

**Nombre del algoritmo.**

Fundamentación.

1. Se cumple el predicado *isSorted(array, 0, k)* en su *k-*ésima iteración, donde la función *isSorted(array, left, right)* devuelve verdadero si el sub-arreglo que va de *left* a *right* está ordenado.

**Nombre del algoritmo.**

Fundamentación.

1. Termina de procesar los espacios de búsqueda pequeños antes que los grandes.

**Nombre del algoritmo.**

Fundamentación.

1. Ofrece la máxima eficiencia para ordenar enteros positivos, pero inadecuado para cadenas de texto.

**Nombre del algoritmo.**

Fundamentación.

1. Es una opción eficiente cuando cada elemento a ordenar puede ser partido en M unidades tal que M es mucho menor que el número de elementos.

**Nombre del algoritmo.**

Fundamentación.

II.- Considere el siguiente algoritmo. *Valor: 5 + 10 + 10 puntos*.

**public** **static** **int**[] processingData(**int**[] array) {

**for** (**int** i = 0; i < array.length/2; i++) {

**boolean** swapped = **false**;

**for** (**int** j = i; j < array.length - i - 1; j++) {

**if** (array[j] < array[j+1])

Utils.*swap*(array, j, j+1);

}

**for** (**int** j = array.length - 2 - i; j > i; j--) {

**if** (array[j] > array[j-1]) {

Utils.*swap*(array, j, j-1);

swapped = **true**;

}

}

**if**(!swapped) **break**;

}

**return** array;

}

1. En términos generales, qué hace el algoritmo y cómo lo hace.

Acomoda de mayor a menor los elementos de un arreglo mediante fuerza bruta, ya que hace una comparación con todos los elementos.

1. Mediante un análisis *a priori*, determine el número comparaciones y swaps que ejecuta para un arreglo de tamaño N. Para el mejor y peor caso.

|  |  |  |
| --- | --- | --- |
|  | Comparaciones | Swaps |
| Mejor | 2N - 2 | 0 |
| Peor | (2N-2)^(N/2) | (2N-2)^(N/2) |

1. Demuestre que la complejidad en el mejor caso es mejor que cuadrática mediante una contradicción.
   1. En el mejor escenario, la fórmula del algoritmo es O(2N-2)
   2. Se asume que la fórmula cae en el tipo de O(N^2)
2. El algoritmo….
3. El algoritmo … Por lo tanto, la complejidad es: **Fórmula**
4. Demostrar: Fórmula ∈ …

Desarrollo

Por lo tanto,…

III.- Diseñe un algoritmo del tipo *divide y vencerás* de ordenamiento basado en el siguiente procedimiento:

* + Si el que el inicial intercambiarlos. Valor 10+ 10+ 5
  + Si hay 3 o más elementos valor al final es más pequeño en la lista entonces:
    1. Llamada recursiva con los primeros 2/3 de la lista
    2. Llamada recursiva con los últimos 2/3 de la lista
    3. Otra vez llamada recursiva con los primeros 2/3 de la lista.

1. Muestra la gráfica comparativa de este algoritmo con el merge sort, (en una sola gráfica)
2. ¿Cuál es la complejidad obtenida a posteriori?

**public** **static** **void** prob3ADA(**int**[] array, **int** start, **int** end) {

**if**(array[start] > array[end])

Utils.*swap*(array, start, end);

**if**((end - start) > 1) {

**int** index = ((end-start) \* 2) / 3;

*prob3ADA*(array, start, start + index);

*prob3ADA*(array, end - index, end);

*prob3ADA*(array, start, start + index);

}

}

IV- Codifique un método **boolean** isHeap(**int**[] array, boolean max) que determine si un arreglo es un montículo. El segundo parámetro “max” si es verdadero indica que el padre debe ser mayor que los hijos, si es falso significa que el padre debe ser menor que sus hijos. *Valor: 10 puntos*.

Usar el estilo del código fuente mostrado, o copiar + pegar de Eclipse rescatando los colores originales.

CONSOLAS o COURIER NEW con sangrías. Debe poderse copiar el código no lo pegues en modo imagen.

V.- Implemente un método recursivo int **search(Node234 n, int value)** que busque un número en un árbol 2,3,4. Si encuentra el valor regresar el índice o -1 en caso de no encontrarlo. Basarse en el código visto en clase. *Valor: 15 puntos.*

Usar el estilo del código fuente mostrado, o copiar + pegar de Eclipse rescatando los colores originales.

CONSOLAS o COURIER NEW con sangrías.

VI.- El siguiente método recursivo construye un fractal como el que se muestra en la figura. Diseñe un método iterativo equivalente (con el mismo nombre y declaración). *Valor: 25 puntos.*

**public** **class** Dibujo {

**public** **static** **void** dibujar(Graphics g, **int** x1, **int** y1, **double** angle, **int** depth) {

**if** (depth == 0)

**return**;

**int** x2 = x1 + (**int**) (Math.cos(Math.toRadians(angle)) \* depth \* 5.0);

**int** y2 = y1 + (**int**) (Math.sin(Math.toRadians(angle)) \* depth \* 5.0);

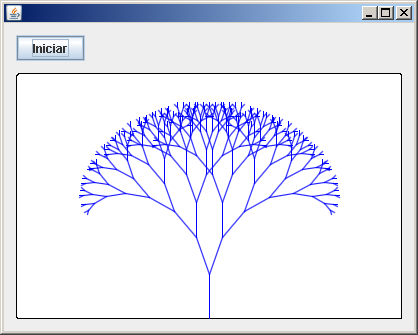
g2.drawLine(x1, y1, x2, y2);

dibujar(g2, x2, y2, angle - 20, depth - 1);

dibujar(g2, x2, y2, angle + 20, depth - 1);

}

}

 Dibujo.dibujar( jPanel1.getGraphics(),800/2, 600, -90, 9);

Usar el estilo del código fuente mostrado, o copiar + pegar de Eclipse rescatando los colores originales.