UNIDAD TEMÁTICA 7 – CLASIFICACIÓN PARTE II

Trabajo de Aplicación 2

EJERCICIO 1 (finalización hora 19:20)

A efectos del algoritmo de ordenación HEAPSORT (y otros) resulta conveniente representar un **ARBOL PARCIALMENTE ORDENADO** mediante un vector. Como vimos anteriormente, el método de ordenación en forma abstracta se reduce a

- **Insertar** inicialmente los elementos del conjunto a ordenar en un APO (algoritmo "***insertar***")
- **Extraer** de la raíz del **APO** cada elemento, manteniendo la característica de **APO** del árbol (método "***suprimeMinimo***")

Es apropiado entonces analizar las siguientes cuestiones:

1. ¿Cuáles son las operaciones básicas involucradas en estos dos algoritmos?
2. ¿Cuál es el orden de estas operaciones cuando el árbol APO se representa en la forma estándar (árbol binario)? (analizarlo a alto nivel)
3. ¿Cuál es el orden de estas operaciones cuando el árbol APO se representa mediante un vector con posiciones de 1 a N?

Escribe las respuestas a estas preguntas y súbelas a la tarea correspondiente.

Responde las preguntas presentadas en pantalla

EJERCICIO 2 (finalización hora 20:15)

1. Dado el seudocódigo del método de ordenación ***Heapsort***, analiza detalladamente el orden del tiempo de ejecución del mismo.
2. Utilizando el método de ***Heapsort*** sobre el siguiente conjunto de datos, mostrar en cada paso cómo se van clasificando. Contabilizar cantidad de comparaciones y movimientos (totales). Considerar las dos etapas del algoritmo.

256 - 458 - 365 - 298 - 043 – 648

3. ¿cuál es la relación de las comparaciones y movimientos respecto al algoritmo de Selección Directa visto anteriormente?

Escribe las resoluciones de estas preguntas y súbelas a la tarea correspondiente.

EJERCICIO 3 (finalización hora 20:50)

Utilizando las clases JAVA provistas por la Cátedra "*TClasificador.java*" y el fragmento de código para el algoritmo de *Heapsort*, implementar completamente el mismo.

Observa que el fragmento de código provisto **contiene errores, que deberán ser identificados y reparados.**

Se requiere:

1. Encuentra y repara los errores en el método de Heapsort.
2. Prueba la ejecución del algoritmo de Heapsort con conjuntos de datos de 300 y 10,000 elementos, cada uno en tres órdenes: ascendente, descendente y, ***midiendo el tiempo de ejecución en cada caso*** (PONER LOS TIEMPOS EN UNA PLANILLA DE CALCULO PARA COMPARARLOS)
3. Verifica que el conjunto resultado en cada caso esté efectivamente ordenado.
4. Sincroniza el repositorio