IIC2133 – Estructuras de Datos y Algoritmos

Interrogación 1

Hora inicio: 14:00 del 5 de octubre del 2020

Hora máxima de entrega: 23:59 del 5 de octubre del 2020, en los cuestionarios de SIDING.

0. Responde esta pregunta en papel y lápiz, incluyendo tu firma al final. Nos reservamos el derecho a no corregir tu prueba según tu respuesta a este ítem. Puedes adjuntar esta pregunta a cualquiera de las preguntas que subas a SIDING.

    a. ¿Cuál es tu nombre completo?
    b. ¿Te comprometes a no preguntar ni responder dudas de la prueba a nadie que no sea parte del cuerpo docente del curso, ya sea de manera directa o indirecta?

Responde sólo 3 de las 4 preguntas a continuación. Si respondes las 4, se escogerá arbitrariamente cuál pregunta no corregir y se la considerará como no entregada. Entrega cada pregunta como un archivo separado en formato PDF en el cuestionario correspondiente en el SIDING.

Cada letra vale 3pts. Si una pregunta pide que hagas algo en una complejidad específica, entonces debes justificar por qué tu respuesta tiene esa complejidad. Si la complejidad no es la que se pide, entonces el puntaje máximo que puedes tener en esa letra es 1pt.

1. Con respecto al algoritmo de ordenación Quicksort:
    a) Supongamos que al ejecutar Quicksort sobre un arreglo particular, la subrutina *partition* hace siempre el mayor número posible de intercambios; ¿cuánto tiempo toma Quicksort en este caso? ¿Qué fracción del mayor número posible de intercambios se harían en el mejor caso? Justifica.
    b) El algoritmo *quicker-sort* llama a la subrutina *pq-partition*, que utiliza *dos* pivotes $\boldsymbol{p}$ y $\boldsymbol{q}$ $(\boldsymbol{p} < \boldsymbol{q})$ para particionar el arreglo en 5 partes: los elementos menores que $\boldsymbol{p}$, el pivote $\boldsymbol{p}$, los elementos entre $\boldsymbol{p}$ y $\boldsymbol{q}$, el pivote $\boldsymbol{q}$, y los elementos mayores que $\boldsymbol{q}$. Escribe el pseudocódigo de *pq-partition*. ¿Es *quicker-sort* más eficiente que *quick-sort*? Justifica.

2. Con respecto a los Heaps binarios de $\boldsymbol{n}$ elementos:
    a) Escribe un algoritmo $\boldsymbol{O(n)}$ para convertir un ABB en un min-Heap. Demuestra que es correcto.
    b) ¿Por qué encontrar el elemento de menor prioridad en un Heap requiere revisar solo $\frac{\boldsymbol{n}}{\boldsymbol{2}}$ elementos?

3. Con respecto a los Árboles Binarios de Búsqueda:
    a) Sea $\boldsymbol{T}$ un ABB de altura $\boldsymbol{h}$. Escribe un algoritmo que posicione un elemento arbitrario de $\boldsymbol{T}$ como raíz de $\boldsymbol{T}$ en $\boldsymbol{O(h)}$ pasos.
    b) Sean $\boldsymbol{T_1}$ y $\boldsymbol{T_2}$ dos ABB de $\boldsymbol{n}$ y $\boldsymbol{m}$ nodos respectivamente. Explica, de manera clara y precisa, cómo realizar un *merge* entre ambos árboles en $\boldsymbol{O(n+m)}$ pasos para dejarlos como un solo ABB **balanceado** $\boldsymbol{T}$ con los nodos de ambos árboles.

4. Con respecto a los ABB balanceados AVL y Rojo-Negro:
    a) Cualquier árbol AVL es también un árbol Rojo-Negro; solo hace falta pintar *adecuadamente* los nodos del AVL de color **rojo** o **negro**. Explica cómo decidir de qué color pintar cada nodo de un árbol AVL para que el árbol resultante sea un árbol Rojo-Negro (válido).
    b) Calcula la profundidad a la que se encuentra la hoja menos profunda en un AVL de altura $\boldsymbol{h}$ lo más desbalanceado posible. ¿Cómo se relaciona con la profundidad a la que se encuentra la hoja menos profunda en un árbol Rojo-Negro de altura $\boldsymbol{h}$, también lo más desbalanceado posible? ¿Para qué valores de $\boldsymbol{h}$ estas dos profundidades son iguales?