IIC2133 – Estructuras de Datos y Algoritmos

Interrogación 1

Hora inicio: 14:00 del 8 de abril del 2020

Hora máxima de entrega: 13:00 del 9 de abril del 2020

0. Formalidad. Responde esta pregunta en papel y lápiz, incluyendo tu firma al final.

    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?

1. A la hora de ordenar usando $HeapSort$, es necesario primero que el arreglo esté estructurado como un heap. Considera la siguiente función que recibe un arreglo A de n elementos:

    $preparar\ para\ HeapSort(A, n)$:

        $for\ i = \lfloor \frac{n}{2} \rfloor .. 0$:

            $sift\ down(A, i)$

    a. Demuestra que $preparar\ para\ HeapSort$ deja el arreglo A estructurado como un heap.
    b. Justifica, mediante cálculos, que la complejidad de este algoritmo es $O(n)$.

2. En ocasiones, como vimos con $QuickSort$, los algoritmos pueden tener una componente aleatoria. Esto puede ser muy útil, ya que en el caso de $QuickSort$ significa que no podemos forzar el peor caso intencionalmente. En otros algoritmos, por ejemplo, puede ser un problema. Considera el siguiente algoritmo de ordenación que recibe un conjunto A de n elementos:

    $BogoSort(A, n)$:

        $while(true)$:

            Sea B un arreglo vacío

            $while$ A aún tenga elementos:

                Extraer aleatoriamente un elemento de A y ponerlo al final de B

            if B está ordenado:

                $return$ B

            Devolver todos los elementos de B a A

    a. Demuestra que $BogoSort$ no es correcto según la definición vista en clases. ¿Cuál es la lógica detrás de este algoritmo? ¿Qué habría que modificar en el algoritmo para que este sea correcto, manteniendo esta lógica?

    b. ¿Qué hace que $QuickSort$ sea correcto, pese a que toma decisiones aleatorias?

3. $\boldsymbol{MergeSort}$ utiliza la estrategia "dividir para conquistar" dividiendo los datos en 2 y luego resolviendo el problema recursivamente. Considera una variante de $\boldsymbol{MergeSort}$ que divide los datos en 3 y los ordena recursivamente, para luego combinar todo en un arreglo ordenado usando una variante de $\boldsymbol{Merge}$ que recibe 3 listas.

   a. Escribe la recurrencia $\boldsymbol{T(n)}$ del tiempo que toma este nuevo algoritmo para un arreglo de $\boldsymbol{n}$ datos. ¿Cuál es su complejidad, en notación asintótica?

   b. Generaliza esta recurrencia a $\boldsymbol{T(n,k)}$ para la variante de $\boldsymbol{MergeSort}$ que divida los datos en $\boldsymbol{k}$. ¿Cuál es la complejidad de este algoritmo en función de $\boldsymbol{n}$ y $\boldsymbol{k}$? Considera que la cantidad de pasos que toma $\boldsymbol{Merge}$ para $\boldsymbol{k}$ listas ordenadas, de $\boldsymbol{n}$ elementos en su totalidad, es $\boldsymbol{n \cdot \log_2(k)}$. Por ejemplo, si $\boldsymbol{k = 2}$, $\boldsymbol{Merge}$ toma $\boldsymbol{n}$ pasos, ya que $\boldsymbol{\log_2(2) = 1}$.

      Finalmente, ¿Qué sucede con la complejidad del algoritmo cuando $\boldsymbol{k}$ tiende a $\boldsymbol{n}$?