Algoritmos y Estructuras de Datos (CB100, 95.15) – Curso Buchwald

1.er parcialito (2R) – 30/06/2025

Resolvé los siguientes problemas en forma clara y legible. Podés incluir tantas funciones auxiliares como creas necesarias. **Los algoritmos a implementar deben ser de la mejor complejidad posible dadas las características del problema**.

1. Implementar una primitiva `KUltimos[T any](k int) []T` para el TDA Cola que dada una cola devuelva un arreglo con los últimos k elementos que saldrían de la cola (los del fondo), en el orden en que saldrían de esta. En el caso de tener menos de k elementos encolados, devolver los existentes en la cola. Al finalizar la ejecución de la función la cola debe quedar en el mismo estado que antes de invocar a la primitiva. Indicar y justificar la complejidad del algoritmo.

Ej: La cola es `[1, 2, 3, 4, 5]` (primer elemento el 1) con `k = 3`, deberíamos obtener un arreglo `[3, 4, 5]`.

2. Indicar Verdadero o Falso, justificando de forma concisa en cualquier caso.
   a. Se puede mejorar la complejidad de MergeSort si se cuenta con información extra
   b. Que un algoritmo de ordenamiento sea in-place implica que ordena respetando el orden original en el que aparecen los elementos de mismo valor.
   c. La complejidad de RadixSort depende únicamente del valor de d (cantidad de componentes) y de k (rango de cada componente).
3. Se cuenta con un arreglo de enteros ordenado de manera ascendente que contiene exactamente un número duplicado (es decir, todos los demás elementos son distintos). Implementar una función que encuentre dicho número utilizando División y Conquista. Indicar y justificar la complejidad del algoritmo, utilizando el Teorema Maestro.

Algoritmos y Estructuras de Datos (CB100, 95.15) – Curso Buchwald

1.er parcialito (2R) – 30/06/2025

Resolvé los siguientes problemas en forma clara y legible. Podés incluir tantas funciones auxiliares como creas necesarias. **Los algoritmos a implementar deben ser de la mejor complejidad posible dadas las características del problema**.

1. Implementar una primitiva `KUltimos[T any](k int) []T` para el TDA Cola que dada una cola devuelva un arreglo con los últimos k elementos que saldrían de la cola (los del fondo), en el orden en que saldrían de esta. En el caso de tener menos de k elementos encolados, devolver los existentes en la cola. Al finalizar la ejecución de la función la cola debe quedar en el mismo estado que antes de invocar a la primitiva. Indicar y justificar la complejidad del algoritmo.

Ej: La cola es `[1, 2, 3, 4, 5]` (primer elemento el 1) con `k = 3`, deberíamos obtener un arreglo `[3, 4, 5]`.

2. Indicar Verdadero o Falso, justificando de forma concisa en cualquier caso.
   a. Se puede mejorar la complejidad de MergeSort si se cuenta con información extra
   b. Que un algoritmo de ordenamiento sea in-place implica que ordena respetando el orden original en el que aparecen los elementos de mismo valor.
   c. La complejidad de RadixSort depende únicamente del valor de d (cantidad de componentes) y de k (rango de cada componente).
3. Se cuenta con un arreglo de enteros ordenado de manera ascendente que contiene exactamente un número duplicado (es decir, todos los demás elementos son distintos). Implementar una función que encuentre dicho número utilizando División y Conquista. Indicar y justificar la complejidad del algoritmo, utilizando el Teorema Maestro.

Algoritmos y Estructuras de Datos (CB100, 95.15) – Curso Buchwald

1.er parcialito (2R) – 30/06/2025

Resolvé los siguientes problemas en forma clara y legible. Podés incluir tantas funciones auxiliares como creas necesarias. **Los algoritmos a implementar deben ser de la mejor complejidad posible dadas las características del problema**.

1. Implementar una primitiva `KUltimos[T any](k int) []T` para el TDA Cola que dada una cola devuelva un arreglo con los últimos k elementos que saldrían de la cola (los del fondo), en el orden en que saldrían de esta. En el caso de tener menos de k elementos encolados, devolver los existentes en la cola. Al finalizar la ejecución de la función la cola debe quedar en el mismo estado que antes de invocar a la primitiva. Indicar y justificar la complejidad del algoritmo.

Ej: La cola es `[1, 2, 3, 4, 5]` (primer elemento el 1) con `k = 3`, deberíamos obtener un arreglo `[3, 4, 5]`.

2. Indicar Verdadero o Falso, justificando de forma concisa en cualquier caso.
   a. Se puede mejorar la complejidad de MergeSort si se cuenta con información extra
   b. Que un algoritmo de ordenamiento sea in-place implica que ordena respetando el orden original en el que aparecen los elementos de mismo valor.
   c. La complejidad de RadixSort depende únicamente del valor de d (cantidad de componentes) y de k (rango de cada componente).
3. Se cuenta con un arreglo de enteros ordenado de manera ascendente que contiene exactamente un número duplicado (es decir, todos los demás elementos son distintos). Implementar una función que encuentre dicho número utilizando División y Conquista. Indicar y justificar la complejidad del algoritmo, utilizando el Teorema Maestro.

Algoritmos y Estructuras de Datos (CB100, 95.15) – Curso Buchwald

1.er parcialito (2R) – 30/06/2025

Resolvé los siguientes problemas en forma clara y legible. Podés incluir tantas funciones auxiliares como creas necesarias. **Los algoritmos a implementar deben ser de la mejor complejidad posible dadas las características del problema**.

1. Implementar una primitiva `KUltimos[T any](k int) []T` para el TDA Cola que dada una cola devuelva un arreglo con los últimos k elementos que saldrían de la cola (los del fondo), en el orden en que saldrían de esta. En el caso de tener menos de k elementos encolados, devolver los existentes en la cola. Al finalizar la ejecución de la función la cola debe quedar en el mismo estado que antes de invocar a la primitiva. Indicar y justificar la complejidad del algoritmo.

Ej: La cola es `[1, 2, 3, 4, 5]` (primer elemento el 1) con `k = 3`, deberíamos obtener un arreglo `[3, 4, 5]`.

2. Indicar Verdadero o Falso, justificando de forma concisa en cualquier caso.
   a. Se puede mejorar la complejidad de MergeSort si se cuenta con información extra
   b. Que un algoritmo de ordenamiento sea in-place implica que ordena respetando el orden original en el que aparecen los elementos de mismo valor.
   c. La complejidad de RadixSort depende únicamente del valor de d (cantidad de componentes) y de k (rango de cada componente).
3. Se cuenta con un arreglo de enteros ordenado de manera ascendente que contiene exactamente un número duplicado (es decir, todos los demás elementos son distintos). Implementar una función que encuentre dicho número utilizando División y Conquista. Indicar y justificar la complejidad del algoritmo, utilizando el Teorema Maestro.