7541 - 20201C - 1er Recuperatorio

EJERCICIO 1 | EJERCICIO 2 | EJERCICIO 3 | EJERCICIO 4 | EJERCICIO 5

Version 1

a. Escriba un programa (definiendo las variables, estructuras y tipos que crea conveniente) de forma tal que el uso de memoria del mismo sea como el que se muestra a continuación (puede agregar variables o punteros auxiliares si es necesario).

b. Muestre el código que hace que dicho programa libere correctamente toda la memoria reservada.

Version 2

a. Escriba un programa (definiendo las variables, estructuras y tipos que crea conveniente) de forma tal que el uso de memoria del mismo sea como el que se muestra a continuación (puede agregar variables o punteros auxiliares si es necesario).

b. Muestre el código que hace que dicho programa libere correctamente toda la memoria reservada.

Version 3

a. Escriba un programa (definiendo las variables, estructuras y tipos que crea conveniente) de forma tal que el uso de memoria del mismo sea como el que se muestra a continuación (puede agregar variables o punteros auxiliares si es necesario).

b. Muestre el código que hace que dicho programa libere correctamente toda la memoria reservada.

7541 - 20201C - 1er Recuperatorio

Version 1

a. Dadas las siguientes ecuaciones de recurrencia ordenarlas segun su orden de complejidad:

- $T1(n) = 27 * T(n/3) + n^3$
- $T2(n) = 9 * T(n/3) + n$
- $T3(n) = 2 * T(n/2) + n^2$

Explicar detalladamente los motivos.

b. ¿Cuál de todos estos algoritmo de ordenamiento garantizan **O(n * log(n))** en el peor de los casos?. Justifique:

- Quick Sort.
- Insertion Sort.
- Merge Sort.
- Selection Sort.

Version 2

a. Explicar cuáles de las siguientes funciones no pueden ser resueltas con el **Teorema Maestro**, y resolver las que sí:

1. $T1(n) = 27 * T(n/3) + n^3$
2. $T2(n) = 3 * \sin(n)$
3. $T3(n) = 2 * T(n/2) + n$
4. $T4(n) = 9 * T(n/3) + 2^n$

b. Determinar la complejidad de:

- $T(n) = n^2 * \log_2(n) + n * (\log_2(n))^2$

Jusifique.

Version 3

a. ¿Cuál es la complejidad del siguiente algoritmo?

```
int i, j, k = 0;
for (i = n / 2; i <= n; i++) {
    for (j = 2; j <= n; j = j * 2) {
        k = k + n / 2;
    }
}
```

b. Determinar la complejidad de:

- $T(n) = n * \log_3(n) + n * \log_2(n)$

Jusifique.

7541 - 20201C - 1er Recuperatorio

Version 1

a. ¿Qué entiende por TDA?

b. ¿Para qué sirven los TDA?

c. Explique el concepto: **Un qué, mil cómos**.

d. ¿Cuáles primitivas no deberían faltar en un TDA? ¿Por qué?

7541 - 20201C - 1er Recuperatorio

Version 1

a. Dado el siguiente vector **v = {3, 4, 6, 8, 10, 11, 23, 44, 56, 93, 94, 98}** mostrar la diferencia entre ordenarlo descendentemente utilizando **Merge Sort** de 4 particiones y **Merge Sort** de 2 particiones. ¿Existe alguna diferencia respecto al tiempo de ejecución?.

Version 2

a. Dado el siguiente vector **v = {3, 4, 6, 8, 10, 11, 23, 44, 56, 93, 94, 98}** mostrar la diferencia entre ordenarlo descendentemente utilizando **Quick Sort con el pivote seleccionado en la primera posición** y por otro lado seleccionando el **pivote en el último lugar**. ¿Existe alguna diferencia respecto al tiempo de ejecución?.

Version 3

a. Dado el siguiente vector **v = {3, 4, 6, 8, 10, 11, 23, 44, 56, 93, 94, 98}** mostrar la diferencia entre ordenarlo descendentemente utilizando **Quick Sort con el pivote seleccionado en el medio** y **Merge Sort de 3 particiones**. ¿Existe alguna diferencia respecto al tiempo de ejecución?.

7541 - 20201C - 1er Recuperatorio

Version 1

Dado el tda **pila_t** y sus operaciones definidas: **pila_crear**, **pila_tope**, **pila_apilar**, **pila_desapilar**, **pila_cantidad**, **pila_destruir**, y sin usar **while** / **for** / **do**:

a. Complete la función **uno_y_uno** que reciba una pila y muestre por pantalla los elementos de la pila de forma alternada empezando por el elemento tope y luego el del fondo hasta vaciar la pila.

```
void uno_y_uno(pila_t* pila);
```

Por ejemplo, si se insertan en la pila los valores [7,6,5,4,3,2,1] (1 es el ultimo elemento insertado), el resultado esperado es: 1,7,2,6,3,5,4.

Puede asumir que las primitivas de pila no fallan nunca y no deben ser verificadas.

Version 2

Dado el tda **pila_t** y sus operaciones definidas: **pila_crear**, **pila_tope**, **pila_apilar**, **pila_desapilar**, **pila_cantidad**, **pila_destruir**, y sin usar **while** / **for** / **do**:

a. Complete la función **mostrar_pila_ordenada** que reciba una pila y un comparador, y muestre por pantalla los elementos de la pila de forma ordenada.

```
void mostrar_pila_ordenada(pila_t* pila, int (*comparar)(void*, void*));
```

Puede asumir que las primitivas de pila no fallan nunca y no deben ser verificadas.

NOTA: En caso de querer usar algún método de ordenamiento, debe implementarlo bajo las mismas restricciones.

Version 3

Dado el tda **cola_t** y sus operaciones definidas: **cola_crear**, **cola_primero**, **cola_encolar**, **cola_desencolar**, **cola_cantidad**, **cola_destruir**, y sin usar **while** / **for** / **do**:

a. Complete la función **mostrar_cola_ordenada** que reciba una cola y un comparador, y muestre por pantalla los elementos de la cola de forma ordenada.

```
void mostrar_cola_ordenada(cola_t* cola, int (*comparar)(void*, void*));
```

Puede asumir que las primitivas de cola no fallan nunca y no deben ser verificadas.

NOTA: En caso de querer usar algún método de ordenamiento, debe implementarlo bajo las mismas restricciones.
