1. Complejidad

Explique qué es el **Teorema Maestro**, cómo se aplica y cuándo puede ser aplicado.

Escriba (en C99) un algoritmo del tipo divide y conquista que dado un vector genérico desordenado, devuelva una estructura conteniendo el elemento máximo y mínimo del conjunto. Explique cómo funciona y justifique su complejidad.

2a. Sort

Utilizando heapsort inplace, ordene el siguiente vector de menor a mayor. Muestre el estado del vector para cada paso del algoritmo y explique cómo funciona. Justifique la complejidad del algoritmo.

`V = [5,6,7,8,9,5,4]`

2b. Sort

Utilizando heapsort inplace, ordene el siguiente vector de mayor a menor. Muestre el estado del vector para cada paso del algoritmo y explique cómo funciona. Justifique la complejidad del algoritmo.

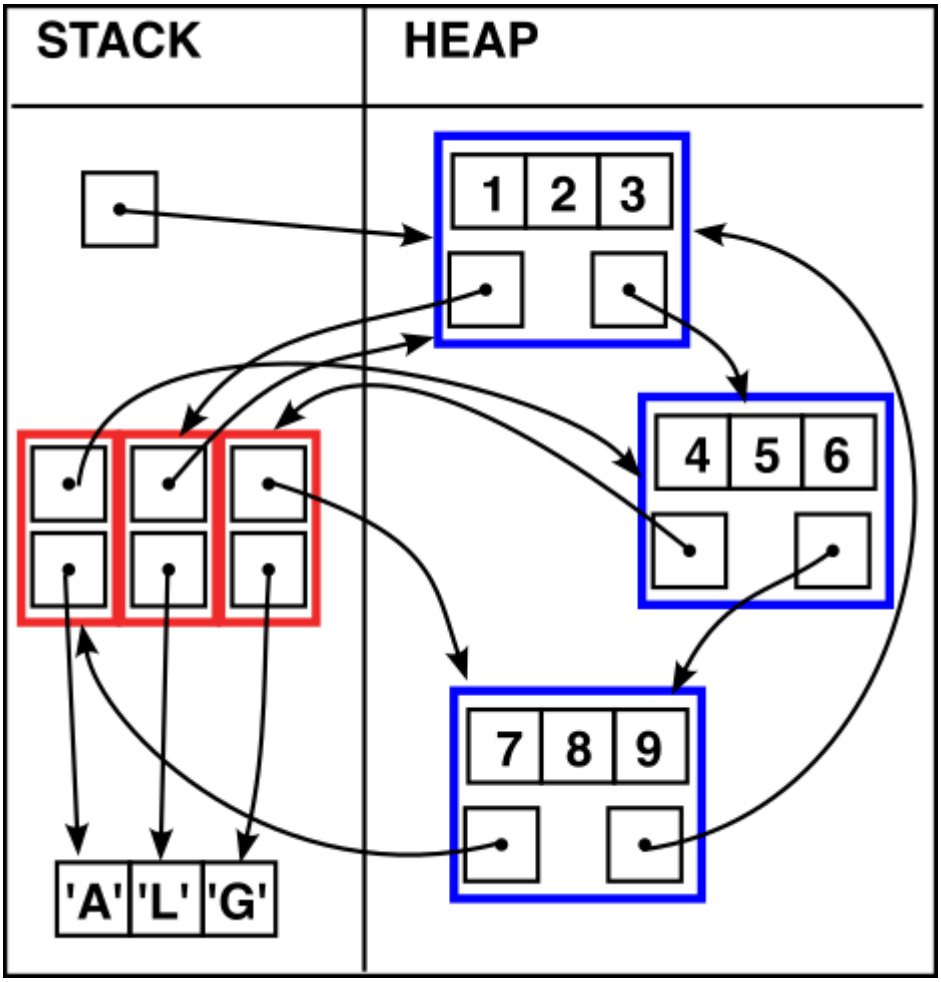
`V = [4,5,9,8,7,6,5]`

3. Punteros (estructuras en el stack)

Escriba un programa en `C` que permita armar en memoria las estructuras presentadas en la imagen. Se pueden utilizar variables auxiliares extra en el stack si es necesario.

Agregue también el código para **liberar toda la memoria de forma correcta**. Notese que en el esquema las estructuras se representan con recuardos de colores y los punteros con cuadrados con un punto en el centro. Por otro lado recuerde que los elementos que se encuentran pegados físicamente (horizontal o vertical) representan datos contiguos en memoria, mientras que los elementos separados no.

ATENCION: Este ejercicio evalúa el uso de punteros y memoria dinámica. **Es fundamental que los accesos a memoria sean correctos para que el ejercicio esté aprobado**.



4. V/F

Indique si cada una de las siguientes afirmaciones son verdaderas o falsas. Justifique claramente cada una de las respuestas dando ejemplos si es necesario.

- Un árbol binario es ordenado de izquierda a derecha
- Un árbol rojo/negro es también AVL
- Counting sort es mas eficiente para ordenar que quicksort
- Los algoritmos del tipo divide y conquista no pueden tener complejidad lineal

5. Recursividad

Escriba un algoritmo recursivo (sin utilizar **for**, **while**, etc) que recibe una lista como la implementada durante el curso y utilizando las funciones de la interfaz (**excepto el iterador interno**), sea capaz de generar una nueva lista con los elementos posicionados en las posiciones multiplo de 3 de la lista original en tiempo lineal. Explique cómo funciona y justifique su complejidad computacional.