Ejercicio 1

Explique cómo funciona el algoritmo **mergesort** y realice un análisis de complejidad del mismo.

Dado el siguiente vector `V=[1,7,2,9,21,3,11,15,4,0]`

Se sabe que dicho vector estaba siendo ordenado de menor a mayor mediante **mergesort**. Sin embargo mientras se ordenada ocurrió un error y el programa fue cerrado de forma inesperada. Se sabe que la función recursiva mergesort fue invocada en total 6 veces y estaba ejecutando la **6ta invocación** cuando el programa se cerró.

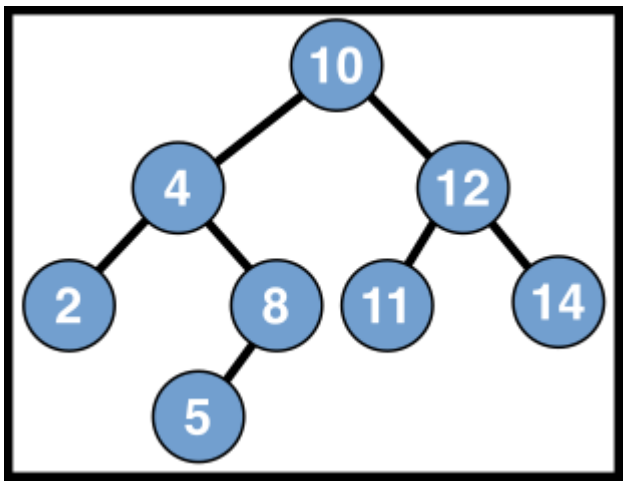
Explique en qué posible estado estaba el algoritmo y justifique si alguna parte del vector ya está (o podría estar) ordenada o no. Complete la corrida del algoritmo para dejar el vector ordenado.

NOTA: El vector muestra el estado del vector al momento de cerrarse el programa. Debe mostrarse cómo se va ordenando el vector desde ese punto paso a paso.

Ejercicio 2

Explique qué es un árbol **AVL** y cómo funciona. Muestre un ejemplo de cada tipo de rotación en un **AVL**.

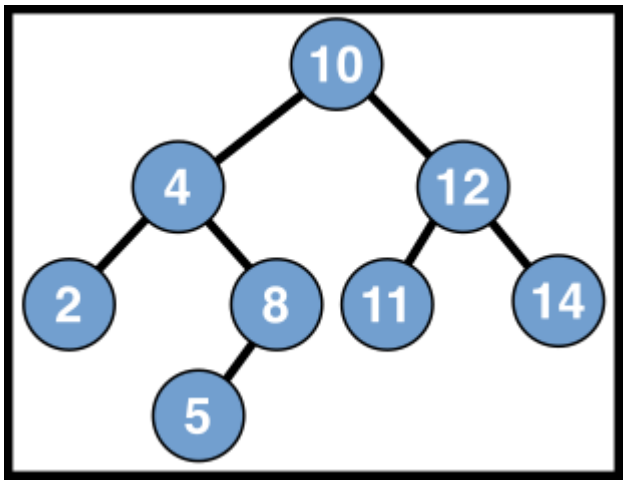
Dado el siguiente **AVL**, indique un orden posible de inserción de los elementos si se sabe que hubo solamente una **rotación doble** (cualquiera) y una **simple izquierda** (en ese orden). Justifique su respuesta.



Ejercicio 2 variante 2

Explique qué es un árbol **AVL** y cómo funciona. Muestre un ejemplo de cada tipo de rotación en un **AVL**.

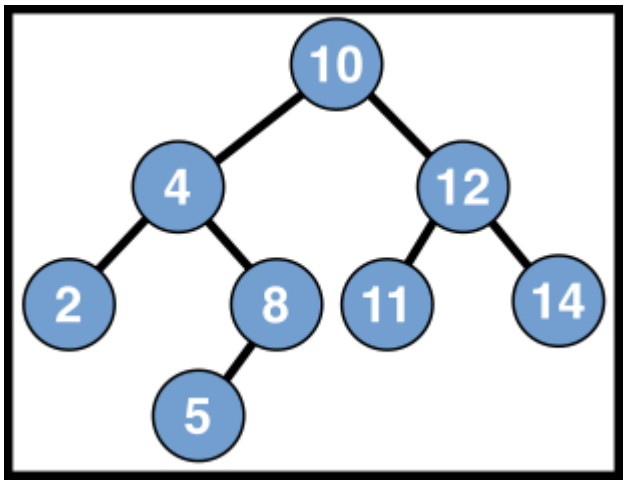
Dado el siguiente **AVL**, indique un orden posible de inserción de los elementos si se sabe que hubo solamente dos **rotaciónes dobles** (cualesquiera). Justifique su respuesta.



Ejercicio 2 variante 3

Explique qué es un árbol **AVL** y cómo funciona. Muestre un ejemplo de cada tipo de rotación en un **AVL**.

Dado el siguiente **AVL**, indique un orden posible de inserción de los elementos si se sabe que hubo solamente una **rotación simple derecha**, y dos **simple izquierda** (en ese orden). Justifique su respuesta.

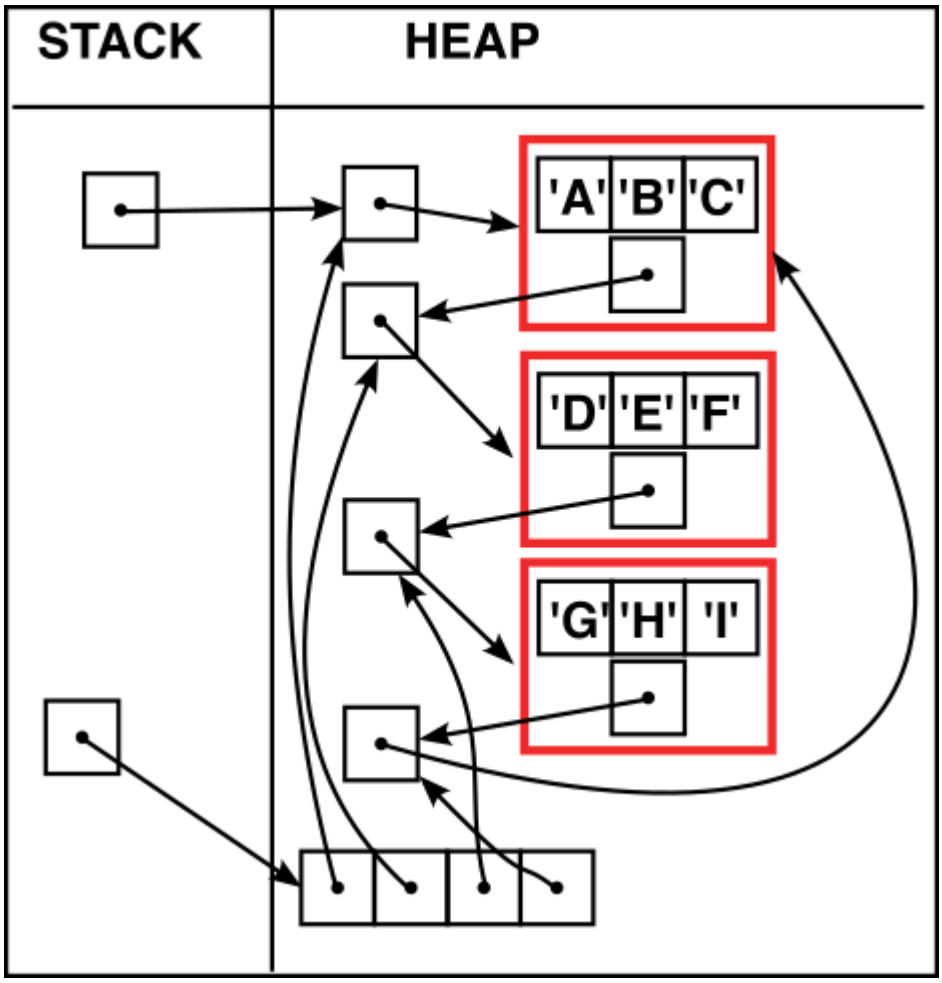


Ejercicio 3

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**.



Ejercicio 4

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

- Una función recursiva escrita como recursiva de cola es mas eficiente que la misma función sin aplicar dicha técnica.
- Un algoritmo que utiliza el mecanismo de backtracking suele ser el mas eficiente para encontrar la respuesta de un problema.
- Un puntero doble ocupa requiere mas espacio en memoria que uno simple.
- Solamente existen 3 recorridos en profundidad aplicables a árboles binarios.

Ejercicio 4 variante 2

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 puntero doble ocupa requiere mas espacio en memoria que uno simple.
- Solamente existen 3 recorridos en profundidad aplicables a árboles binarios.

- Lo algoritmos de ordenamiento estables son mas eficientes que los no estables.
- Un fragmento de código que consiste en 2 for anidados con límites de iteración independientes entre 0 y n y contenido constante resultan en una complejidad computacional $O(n^2)$

Ejercicio 4 variante 3

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

- Una función recursiva escrita como recursiva de cola es mas eficiente que la misma función sin aplicar dicha técnica.
- Un algoritmo que utiliza el mecanismo de backtracking suele ser el mas eficiente para encontrar la respuesta de un problema.
- Lo algoritmos de ordenamiento estables son mas eficientes que los no estables.
- Un fragmento de código que consiste en 2 for anidados con límites de iteración independientes entre 0 y n y contenido constante resultan en una complejidad computacional $O(n^2)$

Ejercicio 5

Escriba una función recursiva (no se permite utilizar for, while, etc) que dado un vector de enteros retorne la cantidad de números no repetidos.

```
size_t cantidad_no_repetidos(int* v, size_t n);
```

Por ejemplo para `[1,3,6,1]` debería retornar 2 (3 y 6 no están repetidos), para `[1,2,3]` debería retornar 3, para `[1,1,2,2,3,3]` debería retornar 0, etc.

Explique cómo funciona y haga un análisis de complejidad.

Se pueden utilizar funciones auxiliares.