Ejercicios Recursión

Ejercicio 1. Ordenación de una pila. Implementa una función que reciba como argumento una pila de enteros y devuelva otra pila con los elementos ordenados en orden ascendente, sin modificar la primera. Se dispone de la interfaz del TAD pila y de las funciones derivadas `Status stack_split (Stack *s0, Stack *s1, Stack *s1)`, `Stack *stack_merge (Stack *s1, Stack *s2)` y `Stack *stack_copy(Stack *s)`.

```
Stack *stack_order(Stack *s){
  Stack *s1 = NULL, *s2 = NULL, *order = NULL

  if (!s) return NULL;

  /*base case*/
  if (stack_size(s) == 1) return stack_copy(s);

  /*general case*/
  if (!(s1 = stack_init())) return NULL;
  if (!(s2 = stack_init())){
    stack_free(s1);
    return NULL;
  }

  stack_split(s, s1, s2);
  order = stack_merge(stack_order(s1), stack_order(s2));

  stack_free(s1);
  stack_free(s2);

  return order;
}
```

Ejercicio 2. BST equilibrado. Implementa una función que a partir de un array de elementos ordenados genere un árbol de búsqueda óptimo.

```
void ordered_bstree(int *a, int first, int last, BSTree *tree){
  int mid;

  if (!a || first > last || !tree) return;

  mid = (last-first)/2;
  if (!tree_insert(tree, a[mid]))
    return;
  ordered_bstree(a, first, mid-1, tree);
  ordered_bstree(a, mid+1, last, tree);
}
```

Ejercicio 3. Mínimo de una pila. Implementa una función que reciba una pila como argumento y devuelva el elemento mínimo de la misma. No está permitido el uso de más memoria que la de la propia pila y el elemento a devolver.

```
typedef int (*cmp_elem_f)(const *void, const *void)

void *stack_min(Stack *s, cmp_elem_f cmp){
  int *elem = NULL, *min = NULL;

  if (!s) return NULL;

  /*base case*/
  if (stack_isEmpty(s)) return NULL;
```

```
  /*general case*/
  elem = stack_pop(s);
  min = stack_min(s, cmp);

  if (!min || cmpe(elem, min) < 0)
    min = elem;

  stack_push(s, elem);

  return min;
}
```