```
static void     arr_print(key_t* arr, key_t size);

/* ===== BINARY TREE ===== */
extern BinTree  tree_binary_create(uint32_t initial_capacity); // Creates binary tree with inicialized
elements
extern void     tree_binary_destroy(BinTree btree); // Frees binary tree
extern void     tree_binary_resize(BinTree *btree);  // Resize binary tree
extern void     tree_binary_insert(BinTree *btree, key_t key); // insert key in binary tree, NO
DUPLICATES
extern void     tree_binary_insert_arr(BinTree *btree, key_t* arr, key_t size); // insert array of keys
extern void     tree_binary_print_inorder(BinTree *btree); // in order print according to tree
extern void     tree_binary_print(BinTree *btree);  // print by levels for visual accuracy
extern idx_t    tree_binary_search_key_inorder(BinTree btree, int32_t key); // search for key in binary
tree by order
extern idx_t    tree_binary_search_key_level(BinTree btree, int32_t key); // faster than inorder
because of this structure
extern void     binary_test_and_log(key_t* arr, FILE *fptr);

/* ===== AVL TREE ===== */
extern AVLTree tree_avl_create(idx_t inicial_capacity);
extern void    tree_avl_destroy(AVLTree* avl);
extern void    tree_avl_resize(AVLTree *avl);
static int      _avl_get_height(AVLTree* avl, idx_t index);
static int      _avl_get_balance(AVLTree* avl, idx_t index);
static idx_t    _avl_rotate_right(AVLTree *avl, idx_t y_index);
static idx_t    _avl_rotate_left(AVLTree *avl, idx_t x_index);
static idx_t    _avl_insert_recursive(AVLTree *avl, idx_t node_index, int key);
extern void     tree_avl_insert(AVLTree *avl, int key);
extern void     tree_avl_insert_arr(AVLTree *avl, key_t* arr, size_t size);
extern AVLNode* tree_avl_search(AVLTree *avl, int key);
extern void     tree_avl_in_order(AVLTree *avl); // in-order print

/* ===== RED BLACK TREE ===== */
extern RBTree  tree_rb_create(uint32_t initial_capacity);
extern void    tree_rb_destroy(RBTree *rb);
extern void    tree_rb_resize(RBTree *rb);
static int     _rb_is_red(RBTree *tree, idx_t i);
static idx_t   _rb_rotate_left(RBTree *tree, idx_t h);
static idx_t   _rb_rotate_right(RBTree *tree, idx_t h);
static void    _rb_flip_colors(RBTree *tree, idx_t h);
static idx_t   _rb_fix_up(RBTree *tree, idx_t h);
static idx_t   _rb_insert_recursive(RBTree *tree, idx_t h, key_t key);
extern void    tree_rb_insert(RBTree *tree, key_t key);
extern int     tree_rb_search(RBTree *rb, int key);

/* ===== TREAP ===== */
extern Treap tree_treap_create(idx_t initial_capacity);
extern void  tree_treap_resize(Treap *treap);
extern void  tree_treap_destroy(Treap *treap);
static idx_t _treap_rotate_right(Treap *treap, idx_t x_idx);
static idx_t _treap_rotate_left(Treap *treap, idx_t x_idx);
static idx_t _treap_insert_recursive(Treap *treap, idx_t idx, key_t key);
extern void  tree_treap_insert(Treap *treap, key_t key);

/* ==== FUNCTION DECLATRATIONS ==== */
static inline int
randint(int a, int b) {
   if (a > b) {
      a ^= b;
      b ^= a ;
      a ^= b;
```