```
    key_t data;
    idx_t idx_left;
    idx_t idx_right;
} BinTreeNode; // TOTAL = 12 Bytes per node;

typedef struct BinaryTree {
    uint32_t capacity;
    uint32_t elements;
    BinTreeNode *root;
} BinTree ;

typedef struct AVLNode {
    idx_t left;
    idx_t right;
    int key;
    int height;
} AVLNode;

typedef struct AVLTree {
    AVLNode *nodes;
    idx_t tree_root; // rotations cause the root to change
    idx_t elements;
    idx_t capacity;
} AVLTree; // 20 bytes

typedef struct RBNode {
    idx_t left;     // 4 bytes
    idx_t right;    // 4 bytes
    key_t key;      // 4 bytes
    int8_t color;   // 1 bytes
} RBNode;

typedef struct RBTree {
    RBNode *nodes;
    idx_t tree_root;
    idx_t elements;
    idx_t capacity;
} RBTree;

typedef struct TreapNode {
    key_t key;
    idx_t priority;
    idx_t left;
    idx_t right;
} TreapNode;

typedef struct Treap {
    TreapNode* nodes;
    idx_t tree_root;
    idx_t elements;
    idx_t capacity;
} Treap;

/* === HELPER FUNCTIONS === */
static inline int randint(int a, int b);
static inline idx_t rand_idx(idx_t a, idx_t b);
static inline int max(int a, int b);
static key_t*   arr_gen_conj_a(const key_t size); // ordem crescent, pouca repetição
static key_t*   arr_gen_conj_b(const key_t size); // ordem decrescent, pouca repetição
static key_t*   arr_gen_conj_c(const key_t size); // ordem aleatoria, pouca repetição
static key_t*   arr_gen_conj_d(const key_t size); // ordem aleatoria, 90% repetidos
```