```
idx_t
tree_binary_search_key_inorder(BinTree btree, int32_t key) {

   /*Helper function */
   idx_t tree_binary_search(BinTreeNode *root, idx_t idx, int32_t key) {
      if (idx == IDX_INVALID || idx == 0) return IDX_INVALID;

      BinTreeNode node = root[idx];
      if (node.data == key) return idx;

      idx_t left = tree_binary_search(root, node.idx_left, key);
      if (left != IDX_INVALID) return left;

      idx_t right = tree_binary_search(root, node.idx_right, key);
      if (right != IDX_INVALID) return right;

      return IDX_INVALID;
   }

   BinTreeNode* root = btree.root;
   if (root->data == key) return 0;

   idx_t left = tree_binary_search(root, root->idx_left, key);
   if (left != IDX_INVALID) return left;

   idx_t right = tree_binary_search(root, root->idx_right, key);
   if (right != IDX_INVALID) return right;


   return IDX_INVALID;
}

idx_t
tree_binary_search_key_level(BinTree btree, int32_t key) {

   /* Como não existe ordem inerente nesta àrvore binária, os nós estão
    * inseridos no array da esquerda para a direita, logo posso percorrer o array.
    * Vou optimizar porque sim. */

   register BinTreeNode *ptr_front = btree.root;
   register BinTreeNode *ptr_back = btree.root + btree.elements;

   /*printf("Search key = %d\n", key);*/

   register int found = 0;
   while (ptr_front + 7 < ptr_back) {

      /* prefetch */
      __builtin_prefetch(ptr_front + 32, 0, 1);
      __builtin_prefetch(ptr_back - 32, 0, 1);

      /* Unrolling */
      found |= (ptr_front->data == key);
      found |= ((ptr_front + 1)->data == key);
      found |= ((ptr_front + 2)->data == key);
      found |= ((ptr_front + 3)->data == key);

      found |= (ptr_back->data == key);
      found |= ((ptr_back - 1)->data == key);
      found |= ((ptr_back - 2)->data == key);
```