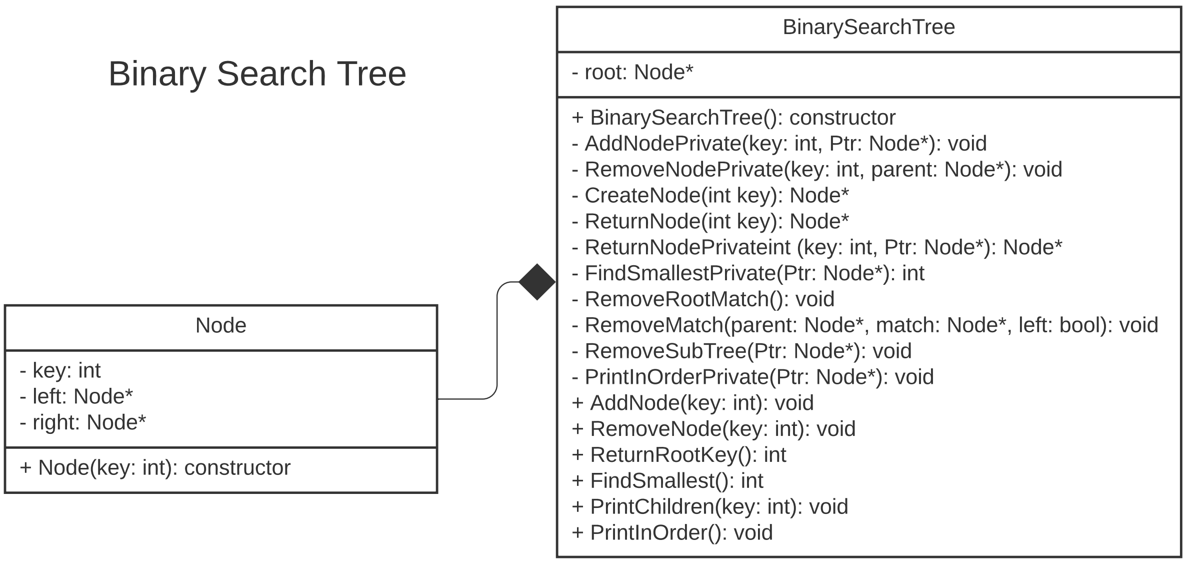
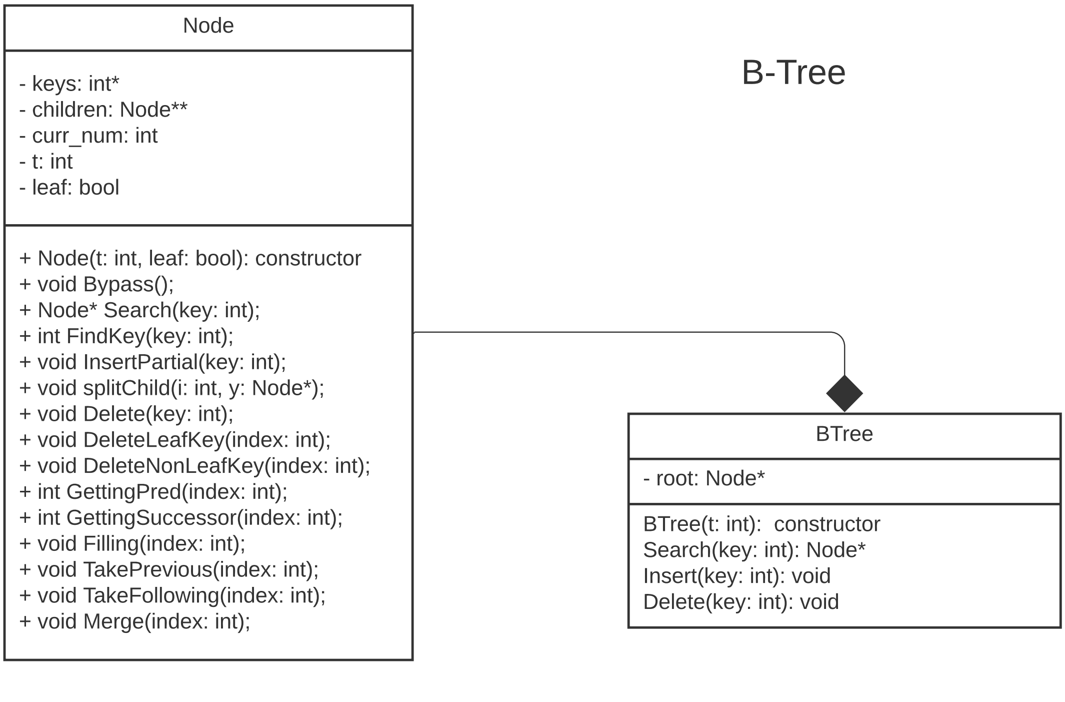
Binary Search Tree


BinarySearchTree
- root: Node*
+ BinarySearchTree(): constructor
- AddNodePrivate(key: int, Ptr: Node*): void
- RemoveNodePrivate(key: int, parent: Node*): void
- CreateNode(int key): Node*
- ReturnNode(int key): Node*
- ReturnNodePrivateint (key: int, Ptr: Node*): Node*
- FindSmallestPrivate(Ptr: Node*): int
- RemoveRootMatch(): void
- RemoveMatch(parent: Node*, match: Node*, left: bool): void
- RemoveSubTree(Ptr: Node*): void
- PrintInOrderPrivate(Ptr: Node*): void
+ AddNode(key: int): void
+ RemoveNode(key: int): void
+ ReturnRootKey(): int
+ FindSmallest(): int
+ PrintChildren(key: int): void
+ PrintInOrder(): void
Node
- key: int
- left: Node*
- right: Node*
+ Node(key: int): constructor

B-Tree


Node
- keys: int*
- children: Node**
- curr_num: int
- t: int
- leaf: bool
+ Node(t: int, leaf: bool): constructor
+ void Bypass();
+ Node* Search(key: int);
+ int FindKey(key: int);
+ void InsertPartial(key: int);
+ void splitChild(i: int, y: Node*);
+ void Delete(key: int);
+ void DeleteLeafKey(index: int);
+ void DeleteNonLeafKey(index: int);
+ int GettingPred(index: int);
+ int GettingSuccessor(index: int);
+ void Filling(index: int);
+ void TakePrevious(index: int);
+ void TakeFollowing(index: int);
+ void Merge(index: int);
BTree
- root: Node*
BTree(t: int): constructor
Search(key: int): Node*
Insert(key: int): void
Delete(key: int): void