# Swinburne University of Technology

Faculty of Science, Engineering and Technology

# ASSIGNMENT COVER SHEET

---

**Subject Code:** COS30008
**Subject Title:** Data Structures and Patterns
**Assignment number and title:** 4, Binary Search Trees & In-Order Traversal
**Due date:** May 26, 2022, 14:30
**Lecturer:** Dr. Markus Lumpe

---

**Your name:** **Nguyen Duy Anh Tu** **Your student id:** 104188405

| Check Tutorial | Mon 10:30 | Mon 14:30 | Tues 08:30 | Tues 10:30 | Tues 12:30 | Tues 14:30 | Tues 16:30 | Wed 08:30 | Wed 10:30 | Wed 12:30 | Wed 14:30 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |

---

Marker's comments:

| Problem | Marks | Obtained |
|---|---|---|
| 1 | 94 | |
| 2 | 42 | |
| 3 | 8+86=94 | |
| Total | 230 | |

---

**Extension certification:**

This assignment has been given an extension and is now due on _____

Signature of Convener:_____

```
#pragma once
#include "BinaryTreeNode.h"
#include <stdexcept>

template<typename T>
class BinarySearchTreeIterator;
template<typename T>
class BinarySearchTree
{
private:
	using BNode = BinaryTreeNode<T>;
	using BTreeNode = BNode*;
	BTreeNode fRoot;

public:
	BinarySearchTree() : fRoot((&BNode::NIL)) {}
	~BinarySearchTree()
	{
		if (!fRoot->empty())
		{
			delete fRoot;
		}
	}
	bool empty() const
	{
		return fRoot->empty();
	}
	size_t height() const
	{
		if (empty())
		{
			throw domain_error("Empty tree has no height.");
		}
		return fRoot->height();
	}

	bool insert(const T& aKey)
	{
		if (empty())
		{
			fRoot = new BNode(aKey);
			return true;
		}
		return fRoot->insert(aKey);
	}
	bool remove(const T& aKey)
	{
		if (empty())
		{
			throw domain_error("Cannot remove in empty tree.");
		}
		if (fRoot->leaf())
		{
			if (fRoot->key != aKey)
			{
				return false;
```

```
                        }
                        fRoot = &BNode::NIL;
                        return true;
                }
                return fRoot->remove(aKey, &BNode::NIL);
        }


        using Iterator = BinarySearchTreeIterator<T>;

        friend class BinarySearchTreeIterator<T>;
        Iterator begin() const
        {
                return Iterator(*this).begin();
        }
        Iterator end() const
        {
                return Iterator(*this).end();
        }
};
```

```
#pragma once
#include <stdexcept>
#include <algorithm>
using namespace std;
template<typename T>
struct BinaryTreeNode
{
	using BNode = BinaryTreeNode<T>;
	using BTreeNode = BNode*;

	T key;
	BTreeNode left;
	BTreeNode right;

	static BNode NIL;
	const T& findMax() const
	{
		if (empty())
		{
			throw domain_error("Empty tree encountered");
		}
		if (right->empty())
		{
			return key;
		}
		return right->findMax();
	}
	const T& findMin() const
	{
		if (empty())
		{
			throw domain_error("Empty tree encountered");
		}
		if (left->empty())
		{
			return key;
		}
		return left->findMin();
	}
	bool remove(const T& aKey, BTreeNode aParent)
	{
	  BTreeNode x = this;
	  BTreeNode y = aParent;
	  while (!x->empty())
	  {
	      if (aKey == x->key)
	      {
	          break;
	      }
	      y = x;
	      x = aKey < x->key ? x->left : x->right;
	  }
	  if (x->empty())
	  {
	      return false;
	  }
```

```
  if (!x->left->empty())
  {
      const T& lKey = x->left->findMax();
      x->key = lKey;
      x->left->remove(lKey, x);
  }
  else
  {
      if (!x->right->empty())
      {
          const T& lKey = x->right->findMin();
          x->key = lKey;
          x->right->remove(lKey, x);
      }
      else
      {
          if (y != &NIL)
          {
              if (y->left == x)
              {
                  y->left = &NIL;
              }
              else
              {
                  y->right = &NIL;
              }
          }
          delete x;
      }
  }
  return true;
}

BinaryTreeNode(): key(T()), left(&NIL), right(&NIL){}
BinaryTreeNode(const T& aKey): key(aKey), left(&NIL), right(&NIL){}
BinaryTreeNode(T&& aKey): key(move(aKey)), left(&NIL), right(&NIL){}
~BinaryTreeNode()
{
       if (!left->empty())
       {
              delete left;
       }
       if (!right->empty())
       {
              delete right;
       }
}

bool empty() const
{
       return this == &NIL;
}
bool leaf() const
{
       return left->empty() && right->empty();
}
```

```
	size_t height() const
	{
		if (empty())
		{
			throw domain_error("Empty tree encountered");
		}
		if (leaf())
		{
			return 0;
		}
		const int left_height = left->empty() ? 1 : left->height() + 1;
		const int right_height = right->empty() ? 1 : right->height() + 1;
		return max(left_height, right_height);
	}
	bool insert(const T& aKey)
	{
		if (empty())
		{
			return false;
		}
		if (aKey > key)
		{
			if (right->empty())
			{
				right = new BNode(aKey);
			}
			else
			{
				return right->insert(aKey);
			}
			return true;
		}
		if (aKey < key)
		{
			if (left->empty())
			{
				left = new BNode(aKey);
			}
			else
			{
				return left->insert(aKey);
			}
			return true;
		}
		return false;
	}
};
template<typename T>
 1	BinaryTreeNode<T> BinaryTreeNode<T>::NIL;
```