Name: VAGHAMASHI KISHAN RAJESHBHAI
Student ID: 202312014

Algorithms Assignment 03

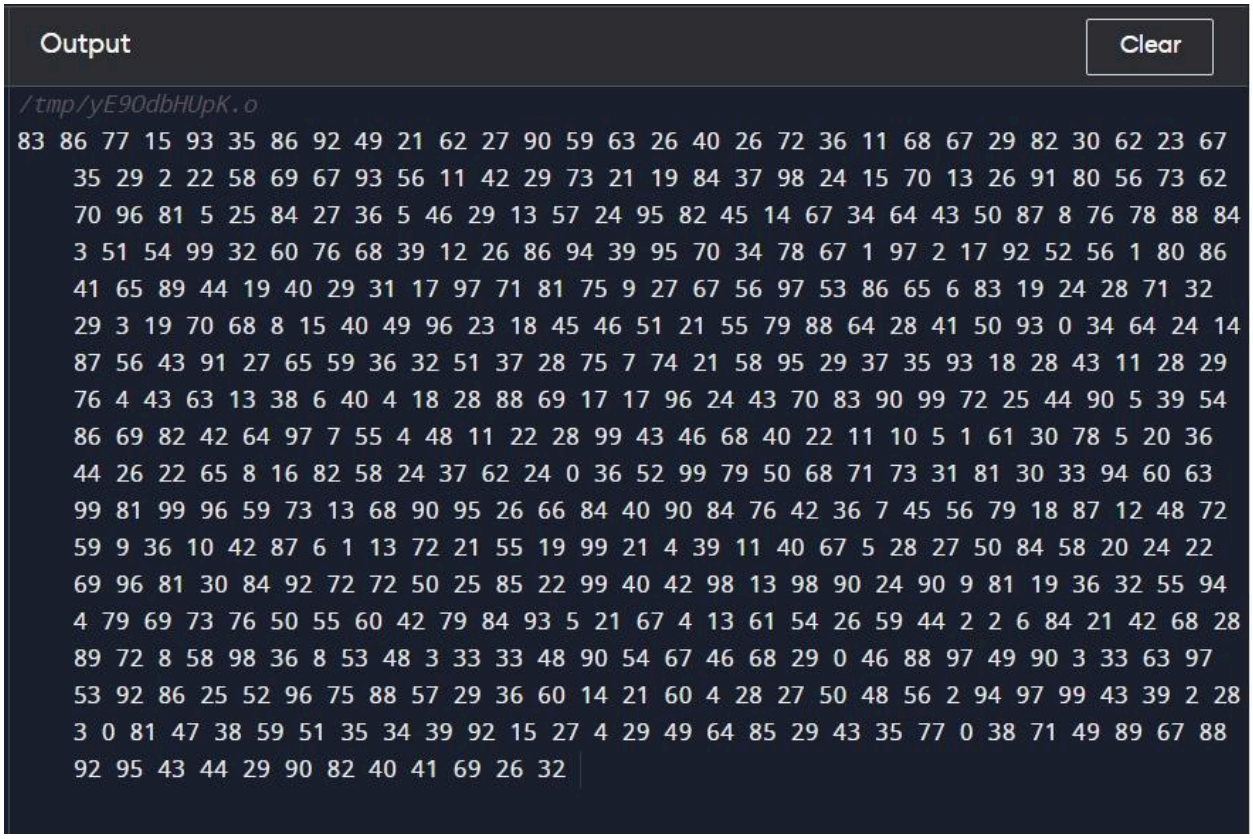
1) Use the random number generator to generate a sequence of 500 random integers

Code:

```cpp
#include <bits/stdc++.h>
using namespace std;
#define NUMBERS_SIZE 500
vector<int> generate_random_n(const int &n)
{
    vector<int> numbers(n);
    generate(begin(numbers), end(numbers),
             []() mutable
             {
                 return rand() % 100;
             });
    return numbers;
}

int main()
{
    auto numbers = generate_random_n(NUMBERS_SIZE);
    for (auto i : numbers)
        cout << i << " ";

    return 0;
}
```

Output:

Name: VAGHAMASHI KISHAN RAJESHBHAI
Student ID: 202312014



2) Create a Binary Search Tree. Insert random numbers generated in Question 1 into the Binary Search Tree. Write a function to find the height of Binary Search Tree. Display the height of Binary Search Tree.

Code:

```
#include <bits/stdc++.h>
using namespace std;
#define NUMBERS_SIZE 500
template <typename T>
class BinaryTree
{
    struct Node
    {
        T data;
        Node *left = nullptr;
        Node *right = nullptr;
        Node() {}
        Node(T arg_data) { data = arg_data; }
```

Name: VAGHAMASHI KISHAN RAJESHBHAI
Student ID: 202312014

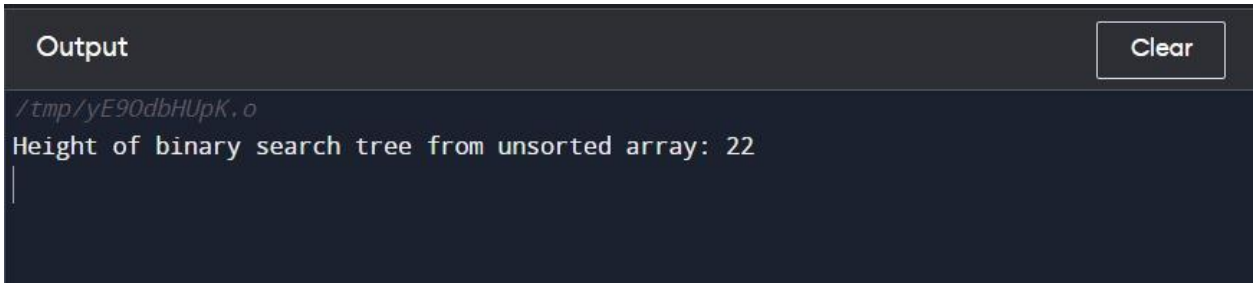
```
    };
    size_t size = 0;
    Node *head = nullptr;
    void inorder_traverse(Node *node)
    {
        if (node == nullptr)
            return;
        inorder_traverse(node->left);
        cout << node->data << endl;
        inorder_traverse(node->right);
    }

public:
    Node *getHead() { return this->head; }
    size_t height(Node *node)
    {
        if (node == nullptr)
            return 0;
        int left_height = height(node->left);
        int right_height = height(node->right);
        return max(left_height, right_height) + 1;
    }
    ssize_t insert(T data)
    {
        Node *new_node;
        new_node = new Node(data);
        size++;
        if (head == nullptr)
        {
            head = new_node;
            return 1;
        }
        auto traverse_node = head;
        while (true)
        {
            if (new_node->data > traverse_node->data)
            {
                if (traverse_node->right == nullptr)
                {
                    traverse_node->right = new_node;
```

Name: VAGHAMASHI KISHAN RAJESHBHAI
Student ID: 202312014

```
                    return 1;
                }
                traverse_node = traverse_node->right;
            }
            if (new_node->data <= traverse_node->data)
            {
                if (traverse_node->left == nullptr)
                {
                    traverse_node->left = new_node;
                    return 1;
                }
                traverse_node = traverse_node->left;
            }
        }
        size--;
        return -1;
    }
    void inorder()
    {
        auto traverse_node = head;
        inorder_traverse(traverse_node);
    }
};
vector<int> generate_random_n(const int &n)
{
    vector<int> numbers(n);
    generate(begin(numbers), end(numbers), [] () mutable
             { return rand() % 100; });
    return numbers;
}
int main()
{
    auto numbers = generate_random_n(NUMBERS_SIZE);
    BinaryTree<int> bt_unsorted;
    for (int x : numbers)
        bt_unsorted.insert(x);
    cout << "Height of binary search tree from unsorted array: ";
    cout << bt_unsorted.height(bt_unsorted.getHead()) << endl;
}
```

Name: VAGHAMASHI KISHAN RAJESHBHAI
Student ID: 202312014

Output:



1) Sort the numbers generated in Question 1 using merge sort. Create a Binary Search Tree. Insert these SORTED key values into the Binary Search Tree. Display the height of Binary Search Tree. Compare the height of trees in Question1 and Question2.

```
#include <bits/stdc++.h>
using namespace std;
#define NUMBERS_SIZE 500
template <typename T>
class BinaryTree
{
    struct Node
    {
        T data;
        Node *left = nullptr;
        Node *right = nullptr;
        Node() {}
        Node(T arg_data) { data = arg_data; }
    };
    size_t size = 0;
    Node *head = nullptr;
    void inorder_traverse(Node *node)
    {
        if (node == nullptr)
            return;
        inorder_traverse(node->left);
        cout << node->data << endl;
        inorder_traverse(node->right);
    }

public:
    Node *getHead()
    {
```

Name: VAGHAMASHI KISHAN RAJESHBHAI
Student ID: 202312014

```
        return this->head;
    }
    size_t height(Node *node)
    {
        if (node == nullptr)
            return 0;
        int left_height = height(node->left);
        int right_height = height(node->right);
        return max(left_height, right_height) + 1;
    }
    ssize_t insert(T data)
    {
        Node *new_node;
        new_node = new Node(data);
        size++;
        if (head == nullptr)
        {
            head = new_node;
            return 1;
        }
        auto traverse_node = head;
        while (true)
        {
            if (new_node->data > traverse_node->data)
            {
                if (traverse_node->right == nullptr)
                {
                    traverse_node->right = new_node;
                    return 1;
                }
                traverse_node = traverse_node->right;
            }
            if (new_node->data <= traverse_node->data)
            {
                if (traverse_node->left == nullptr)
                {
                    traverse_node->left = new_node;
                    return 1;
                }
                traverse_node = traverse_node->left;
```

Name: VAGHAMASHI KISHAN RAJESHBHAI
Student ID: 202312014

```
            }
        }
        size--;
        return -1;
    }
    void inorder()
    {
        auto traverse_node = head;
        inorder_traverse(traverse_node);
    }
};
vector<int> generate_random_n(const int &n)
{
    vector<int> numbers(n);
    generate(begin(numbers), end(numbers), []() mutable
             { return rand() % 1000; });
    return numbers;
}
int main()
{
    auto numbers = generate_random_n(NUMBERS_SIZE);
    BinaryTree<int> bt_unsorted;
    for (int x : numbers)
        bt_unsorted.insert(x);
    cout << "Height of binary search tree from unsorted array: ";
    cout << bt_unsorted.height(bt_unsorted.getHead()) << endl;
    BinaryTree<int> bt_sorted;
    sort(numbers.begin(), numbers.end());
    for (int x : numbers)
        bt_sorted.insert(x);
    cout << "Height of binary search tree from sorted array: ";
    cout << bt_sorted.height(bt_sorted.getHead()) << endl;
}
```

Output:

Name: VAGHAMASHI KISHAN RAJESHBHAI
Student ID: 202312014

```
Output                                                    Clear
/tmp/yE90dbHUpK.o
Height of binary search tree from unsorted array: 19
Height of binary search tree from sorted array: 405
```

2) Your class has 100+ students, whose ID number & name data is provided in ClassList.txt file. Create a binary search tree and Insert the student data into the binary search tree. [NOTE: ID Number is key, and name is satellite data.]

```
#include <bits/stdc++.h>
using namespace std;
#define NUMBERS_SIZE 500
typedef pair<long, string> STUDENT;
template <typename T>
class BinaryTree
{
    struct Node
    {
        T data;
        Node *left = nullptr;
        Node *right = nullptr;
        Node() {}
        Node(T arg_data) { data = arg_data; }
    };
    size_t size = 0;
    Node *head = nullptr;
    void inorder_traverse(Node *node)
    {
        if (node == nullptr)
            return;
        inorder_traverse(node->left);
        cout << node->data << endl;
        inorder_traverse(node->right);
    }

public:
    Node *getHead() { return this->head; }
    size_t height(Node *node)
```

Name: VAGHAMASHI KISHAN RAJESHBHAI
Student ID: 202312014

```
    {
        if (node == nullptr)
            return 0;
        int left_height = height(node->left);
        int right_height = height(node->right);
        return max(left_height, right_height) + 1;
    }
    string find(long id)
    {
        auto traverse_node = head;
        while (true)
        {
            if (id == traverse_node->data.first)
                return traverse_node->data.second;
            if (id > traverse_node->data.first)
                traverse_node = traverse_node->right;
            if (id < traverse_node->data.first)
                traverse_node = traverse_node->left;
        }
        return "";
    }
    ssize_t insert(T data)
    {
        Node *new_node;
        new_node = new Node(data);
        size++;
        if (head == nullptr)
        {
            head = new_node;
            return 1;
        }
        auto traverse_node = head;
        while (true)
        {
            if (new_node->data > traverse_node->data)
            {
                if (traverse_node->right == nullptr)
                {
                    traverse_node->right = new_node;
                    return 1;
```

Name: VAGHAMASHI KISHAN RAJESHBHAI
Student ID: 202312014

```
                }
                traverse_node = traverse_node->right;
            }
            if (new_node->data <= traverse_node->data)
            {
                if (traverse_node->left == nullptr)
                {
                    traverse_node->left = new_node;
                    return 1;
                }
                traverse_node = traverse_node->left;
            }
        }
        size--;
        return -1;
    }
};
vector<int> generate_random_n(const int &n)
{
    vector<int> numbers(n);
    generate(begin(numbers), end(numbers), []() mutable
             { return rand() % 1000; });
    return numbers;
}
auto read_from_file(const string &filename)
{
    vector<pair<long, string>> list;
    ifstream fin(filename);
    if (!fin.is_open())
        exit(-1);
    long id;
    string name;
    getline(fin, name); // first row
    while (fin >> id)
    {
        getline(fin, name);
        list.push_back(make_pair(id, name));
    }
    return list;
}
```

Name: VAGHAMASHI KISHAN RAJESHBHAI
Student ID: 202312014

```
auto BST_from_sorted(BinaryTree<STUDENT> &bt, vector<STUDENT> &list,
int l, int r)
{
    if (l > r)
        return;
    int mid = (l + r) / 2;
    bt.insert(list[mid]);
    BST_from_sorted(bt, list, l, mid - 1);
    BST_from_sorted(bt, list, mid + 1, r);
}
int main()
{
    auto class_list = read_from_file("ClassList.txt");
    BinaryTree<STUDENT> bt;
    for (auto &p : class_list)
        bt.insert(p);
}
```

3) Write a function that takes student ID number as input and outputs the name of the student. Use binary search tree created in Question 4.

```
#include <algorithm>
#include <cstddef>
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <vector>
#define NUMBERS_SIZE 500
typedef std::pair<long, std::string> STUDENT;
template <typename T>
class BinaryTree
{
    struct Node
    {
        T data;
        Node *left = nullptr;
        Node *right = nullptr;
        Node() {}
        Node(T arg_data) { data = arg_data; }
    };
```

Name: VAGHAMASHI KISHAN RAJESHBHAI
Student ID: 202312014

```
    size_t size = 0;
    Node *head = nullptr;
    void inorder_traverse(Node *node)
    {
        if (node == nullptr)
            return;
        inorder_traverse(node->left);
        std::cout << node->data << std::endl;
        inorder_traverse(node->right);
    }

public:
    Node *getHead() { return this->head; }
    size_t height(Node *node)
    {
        if (node == nullptr)
            return 0;
        int left_height = height(node->left);
        int right_height = height(node->right);
        return std::max(left_height, right_height) + 1;
    }
    std::string find(long id)
    {
        auto traverse_node = head;
        while (true)
        {
            if (id == traverse_node->data.first)
                return traverse_node->data.second;
            if (id > traverse_node->data.first)
                traverse_node = traverse_node->right;
            if (id < traverse_node->data.first)
                traverse_node = traverse_node->left;
        }
        return "";
    }
    ssize_t insert(T data)
    {
        Node *new_node;
        new_node = new Node(data);
        size++;
```

Name: VAGHAMASHI KISHAN RAJESHBHAI
Student ID: 202312014

```
        if (head == nullptr)
        {
            head = new_node;
            return 1;
        }
        auto traverse_node = head;
        while (true)
        {
            if (new_node->data > traverse_node->data)
            {
                if (traverse_node->right == nullptr)
                {
                    traverse_node->right = new_node;
                    return 1;
                }
                traverse_node = traverse_node->right;
            }
            if (new_node->data <= traverse_node->data)
            {
                if (traverse_node->left == nullptr)
                {
                    traverse_node->left = new_node;
                    return 1;
                }
                traverse_node = traverse_node->left;
            }
        }
        size--;
        return -1;
    }
};
std::vector<int> generate_random_n(const int &n)
{
    std::vector<int> numbers(n);
    std::generate(std::begin(numbers), std::end(numbers),
                  []() mutable
                  { return rand() % 1000; });
    return numbers;
}
auto read_from_file(const std::string &filename)
```

Name: VAGHAMASHI KISHAN RAJESHBHAI
Student ID: 202312014

```
{
    std::vector<std::pair<long, std::string>> list;
    std::ifstream fin(filename);
    if (!fin.is_open())
        exit(-1);
    long id;
    std::string name;
    std::getline(fin, name); // first row
    while (fin >> id)
    {
        std::getline(fin, name);
        list.push_back(std::make_pair(id, name));
    }
    return list;
}
auto BST_from_sorted(BinaryTree<STUDENT> &bt, std::vector<STUDENT>
&list, int l, int r)
{
    if (l > r)
        return;
    int mid = (l + r) / 2;
    bt.insert(list[mid]);
    BST_from_sorted(bt, list, l, mid - 1);
    BST_from_sorted(bt, list, mid + 1, r);
}

int main()
{
    auto class_list = read_from_file("ClassList.txt");
    BinaryTree<STUDENT> bt;
    for (auto &p : class_list)
        bt.insert(p);
    BST_from_sorted(bt, class_list, 0, class_list.size());
    std::cout << bt.find(202312014) << std::endl;
}
```

Output:

Name: VAGHAMASHI KISHAN RAJESHBHAI
Student ID: 202312014

```
VAGHAMASHI KISHAN RAJESHBHAI
```