

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:

```
#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

Output

Clear

/tmp/yE90dbHUpK.o

```
83 86 77 15 93 35 86 92 49 21 62 27 90 59 63 26 40 26 72 36 11 68 67 29 82 30 62 23 67
 35 29 2 22 58 69 67 93 56 11 42 29 73 21 19 84 37 98 24 15 70 13 26 91 80 56 73 62
 70 96 81 5 25 84 27 36 5 46 29 13 57 24 95 82 45 14 67 34 64 43 50 87 8 76 78 88 84
 3 51 54 99 32 60 76 68 39 12 26 86 94 39 95 70 34 78 67 1 97 2 17 92 52 56 1 80 86
 41 65 89 44 19 40 29 31 17 97 71 81 75 9 27 67 56 97 53 86 65 6 83 19 24 28 71 32
 29 3 19 70 68 8 15 40 49 96 23 18 45 46 51 21 55 79 88 64 28 41 50 93 0 34 64 24 14
 87 56 43 91 27 65 59 36 32 51 37 28 75 7 74 21 58 95 29 37 35 93 18 28 43 11 28 29
 76 4 43 63 13 38 6 40 4 18 28 88 69 17 17 96 24 43 70 83 90 99 72 25 44 90 5 39 54
 86 69 82 42 64 97 7 55 4 48 11 22 28 99 43 46 68 40 22 11 10 5 1 61 30 78 5 20 36
 44 26 22 65 8 16 82 58 24 37 62 24 0 36 52 99 79 50 68 71 73 31 81 30 33 94 60 63
 99 81 99 96 59 73 13 68 90 95 26 66 84 40 90 84 76 42 36 7 45 56 79 18 87 12 48 72
 59 9 36 10 42 87 6 1 13 72 21 55 19 99 21 4 39 11 40 67 5 28 27 50 84 58 20 24 22
 69 96 81 30 84 92 72 72 50 25 85 22 99 40 42 98 13 98 90 24 90 9 81 19 36 32 55 94
 4 79 69 73 76 50 55 60 42 79 84 93 5 21 67 4 13 61 54 26 59 44 2 2 6 84 21 42 68 28
 89 72 8 58 98 36 8 53 48 3 33 33 48 90 54 67 46 68 29 0 46 88 97 49 90 3 33 63 97
 53 92 86 25 52 96 75 88 57 29 36 60 14 21 60 4 28 27 50 48 56 2 94 97 99 43 39 2 28
 3 0 81 47 38 59 51 35 34 39 92 15 27 4 29 49 64 85 29 43 35 77 0 38 71 49 89 67 88
 92 95 43 44 29 90 82 40 41 69 26 32
```

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;  
                return 1;  
            }  
            traverse_node = traverse_node->right;  
        }  
    }  
    return 0;  
}
```

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:

```
Output Clear  
/tmp/yE90dbHUpK.o  
Height of binary search tree from unsorted array: 22  
|
```

- 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 <cstdint>
#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