

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**КУРСОВАЯ РАБОТА**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Случайные бинарные деревья поиска**

Студент гр. 9304

\_\_\_\_\_

Ковалёв П. Д.

Преподаватель

\_\_\_\_\_

Филатов А. Ю.

Санкт-Петербург

2020

## **ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Студент Ковалёв П. Д.

Группа 9304

Тема работы: Случайные бинарные деревья поиска - исследование

Исходные данные:

Случайное бинарное дерево поиска, построенное по случайно сгенерированному набору данных

Содержание пояснительной записки:

1. Аннотация
2. Содержание
3. Введение
4. Описание алгоритма, структуры данных и кода программы
5. Тестирование
6. Исследование
7. Заключение
8. Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 30 страниц.

Дата выдачи задания: 28.11.2020

Дата сдачи реферата: 28.12.2020

Дата защиты реферата: 28.12.2020

Студент

\_\_\_\_\_

Ковалёв П. Д.

Преподаватель

\_\_\_\_\_

Филатов А. Ю.

## **АННОТАЦИЯ**

В данной работе происходит исследование операции вставки и удаления в случайное бинарное дерево поиска. Исследование происходит для среднего и худшего случаев. Объектом исследования выступает зависимость скорости выполнения вышеперечисленных операций от входных данных.

## **SUMMARY**

In this work we study the operation of insertion and deletion in a random binary search tree. The study takes place for the average and worst cases. The object of research is the dependence of the speed of execution of the above operations on the input data.

## СОДЕРЖАНИЕ

	Аннотация	3
	Введение	5
1.	Формальная постановка задачи	6
2.	Описание алгоритма	7
2.1.	Алгоритм вставки элемента в случайное бинарное дерево поиска	7
2.2.	Алгоритм удаления элемента из случайного бинарного дерева поиска	8
3.	Описание структур данных и функций	9
3.1.	Случайное бинарное дерево поиска	9
3.2.	Описание функций для работы со случайным бинарным деревом поиска	11
4.	Тестирование	15
5.	Исследование	17
5.1.	План исследования	17
5.2.	Генерация входных данных	17
5.3.	Теоретическая оценка сложности алгоритмов	17
5.4.	Исследование вставки и удаления в среднем случае	18
5.5.	Исследование вставки и удаления в худшем случае	20
5.6.	Исследование зависимости высоты дерева от числа узлов	24
5.7.	Итоги исследования	26
	Заключение	27
	Список использованных источников	28
	Приложение А. Результаты тестирования	29
	Приложение Б. Программный код	32

## **ВВЕДЕНИЕ**

Цель работы: реализация и экспериментальное машинное исследование алгоритмов вставки и удаления в случайном бинарном дереве поиска.

Основные задачи: по заданной последовательности элементов построить структуру данных — случайное бинарное дерево поиска, реализовать возможность вставки и удаления элемента. Исследовать скорость работы алгоритмов вставки и удаления элементов.

## 1. ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

Реализовать структуру данных — случайное бинарное дерево поиска и исследовать скорость работы алгоритмов вставки и удаления элемента в дереве. Сравнить практические результаты с теоретической оценкой.

Для 8го вариант требуется:

1) По заданной последовательности элементов Elem построить структуру данных

определённого типа – БДП или хеш-таблицу;

2) Выполнить одно из следующих действий:

б) Для построенной структуры данных проверить, входит ли в неё элемент *e* типа Elem, и если входит, то удалить элемент *e* из структуры данных (первое обнаруженное вхождение). Предусмотреть возможность повторного выполнения с другим элементом.

## **2. ОПИСАНИЕ АЛГОРИТМА**

### **2.1. Алгоритм вставки элемента в случайное бинарное дерево поиска**

Вставка в случайное бинарное дерево поиска происходит следующим образом: вставляемый элемент сравнивается сначала с ключом, содержащимся в узле корня дерева; если элемент больше ключа, то происходит переход к правому поддереву, если же элемент меньше ключа, то происходит переход к левому поддереву. Данный алгоритм применяется для выбранного поддерева. В случае, если значение элемента и ключа в узле корня поддерева совпадает, то в корневом узле увеличивается счетчик попыток вставить данный элемент. Если же значение элемента и ключа узла корня поддерева не совпадают, а у корня нету сыновей, то данный элемент становится правым или левым сыном данного узла, в зависимости от того, больше или меньше он ключа корня соответственно.

## **2.2. Алгоритм удаления элемента из случайного бинарного дерева поиска**

Удаление элемента из случайного бинарного дерева поиска происходит следующим образом: сначала происходит поиск удаляемого элемента.

Алгоритм поиска следующий: начинаем идти по дереву с корня, проходим в правое или в левое поддерева в зависимости от того, искомый элемент больше или меньше ключа, содержащегося в узле текущего корня. Когда элемент станет равным ключу, содержащемуся в узле корня, алгоритм поиска прекращает свою работу. Далее происходит удаление. Если у узла с удаляемым элементом нету потомков, а значит его можно удалить без лишних действий.

В случае, когда у удаляемого элемента есть правый сын — находим минимальный элемент правого поддерева узла с удаляемым элементом, заменяем удаляемый элемент на найденный минимальный, после чего, если у найденного минимального элемента нету потомков, удаляем данный элемент. В противном случае вызываем алгоритм удаления уже для данного элемента.

В случае, если у удаляемого элемента есть левый сын, но нет правого — находим максимальный элемент левого поддерева узла с удаляемым элементом, заменяем удаляемый элемент на найденный максимальный, после чего, если у найденного максимального элемента отсутствуют потомки, удаляем его. В противном случае вызываем алгоритм удаления уже для данного элемента.



### 3. ОПИСАНИЕ СТРУКТУР ДАННЫХ И ФУНКЦИЙ

#### 3.1. Случайное бинарное дерево поиска

Случайное БДП — структура данных, БДП, построенное по входной последовательности следующим образом: первый элемент последовательности — корень, каждый следующий элемент вставляется следующим образом: если у текущего корня нету потомков, то вставляемый элемент становится правым или левым сыном корня, если он больше либо меньше корня соответственно. В случае, если у корня есть потомки и вставляемый элемент больше корня, то происходит переход к правому поддереву, если же меньше — к левому. Если же вставляемый элемент больше текущего корня, то в узле данного корня происходит увеличение счетчика попыток вставки данного элемента.

Само бинарное дерево поиска — бинарное дерево, для которого выполняются следующие условия:

- Оба поддерева — левое и правое — являются двоичными деревьями поиска.
- У всех узлов *левого* поддерева произвольного узла  $X$  значения ключей данных *меньше либо равны*, нежели значение ключа данных самого узла  $X$ .
- У всех узлов *правого* поддерева произвольного узла  $X$  значения ключей данных *больше либо равны*, нежели значение ключа данных самого узла  $X$ .

Очевидно, данные в каждом узле должны обладать ключами, на которых определена операция сравнения *меньше*.

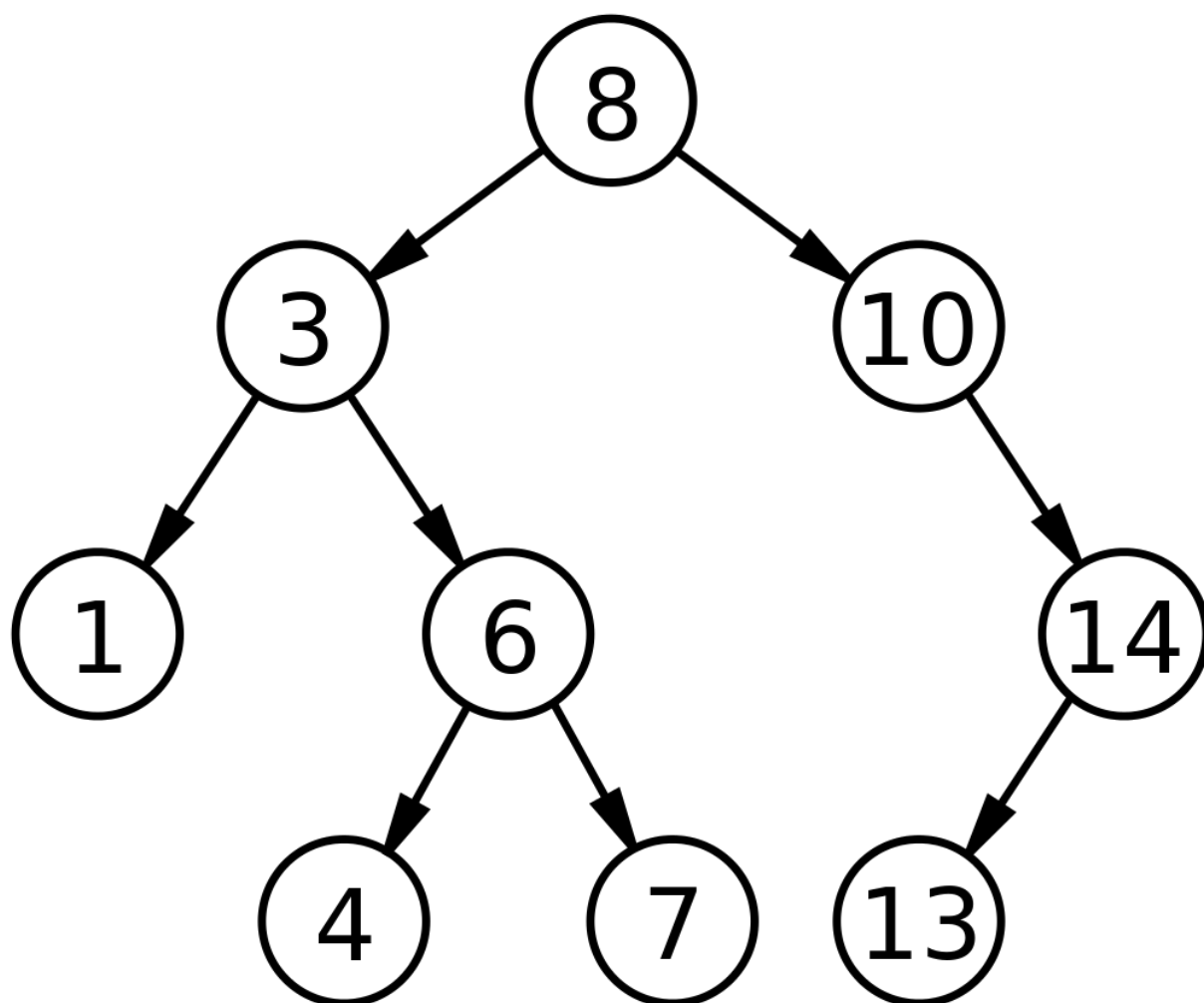


Рисунок 1 — Пример бинарного дерева поиска

## 3.2. Описание функций для работы со случайным бинарным деревом поиска

### 3.2.1. Класс узла БДП

```
class Node{
public:
    int counter;
    T data;
    std::shared_ptr<Node<T>> left {nullptr};
    std::shared_ptr<Node<T>> right {nullptr};
    std::weak_ptr<Node<T>> parent;
    Node() {
        counter = 0;
    }
};
```

Класс *Node* представляет собой узел бинарного дерева поиска. Поле *counter* содержит в себе количество попыток вставки ключа *data*. Также имеются указатели на левого сына (*left*), правого сына (*right*) и родителя (*parent*).

### 3.2.2. Класс случайного БДП

```
BSTree() {}
```

Конструктор класса случайного бинарного дерева поиска.

```
void searchAndInsert(T elem, std::shared_ptr<Node<T>>& ptr){}
```

Метод *searchAndInsert()* осуществляет вставку в случайное БДП. Сначала происходит проверка поданного указателя на узел на пустоту. Если указатель пуст, то создается новый узел дерева и вставляемый элемент помещается в этот узел. Если же указатель не пуст, то вставляемый элемент сравнивается с ключом, лежащим в узле, на который указывает поданный указатель. Если элемент больше ключа — метод запускается с указателем на узел правого

поддерева, если меньше ключа — метод запускается с указателем на узел левого поддерева, иначе увеличивается счетчик попыток вставить данный элемент.

```
T findMin(std::shared_ptr<Node<T>> p, int& ct){}
```

```
T findMax(std::shared_ptr<Node<T>> p, int& ct){}
```

Методы *findMin()* и *findMax()* производят поиск минимального и максимального элементов дерева соответственно.

Метод *findMin()* идет только по левым потомкам каждого узла дерева. Когда он дойдет до конца, то узел, на котором он остановится, будет содержать минимальный ключ.

Метод *findMax()* идет только по правым потомкам каждого узла дерева. Когда он дойдет до конца, то узел, на котором он остановится, будет содержать максимальный ключ.

```
void searchAndDelete(T elem, std::shared_ptr<Node<T>>& ptr){  
}
```

Метод *searchAndDelete()* осуществляет удаление элемента из случайного БДП. Метод сначала ищет элемент, который надо удалить, после того, как он его нашел, смотрит, какие есть потомки у узла с удаляемым элементом. Если есть правый потомок — происходит поиск минимального элемента правого поддерева, с последующей заменой первоначального элемента на этот минимальный элемент и удалением узла с минимальным элементом. Если правого потомка нет, но есть левый потомок — происходит поиск максимального элемента в левом поддереве, замена первоначального элемента на максимальный элемент левого поддерева и последующее удаление максимального элемента левого поддерева. Если же у узла с удаляемым элементом нет потомков — происходит удаление данного узла.

```
void printBST(std::shared_ptr<Node<T>> bt, std::string str="") {}
```

Метод класса BSTree, который печатает случайное БДП.

### **3.2.3. Класс выполняющий исследование операций в БДП**

```
void insertAverageResearch(int size) {}
```

Метод, осуществляющий исследование среднего случая вставки элемента в случайное БДП.

```
void insertWorstResearch(int size) {}
```

Метод, осуществляющий исследование худшего случая вставки элемента в случайное БДП.

```
void deleteAverageResearch(int size) {}
```

Метод, осуществляющий исследование среднего случая удаления элемента в случайном БДП.

```
void deleteWorstResearch(int size) {}
```

Метод, осуществляющий исследование худшего случая удаления элемента в случайном БДП.

```
void shuffle(int count, std::vector<int>& vec) {}
```

Метод, создающий массив элементов от 1 до *count*, и случайным образом перемешивающий их.

```
void worstGenerator(int count, std::vector<int>& vec){}
```

Метод, создающий упорядоченный массив элементов от 1 до *count*.

Разработанный программный код представлен в приложении Б.

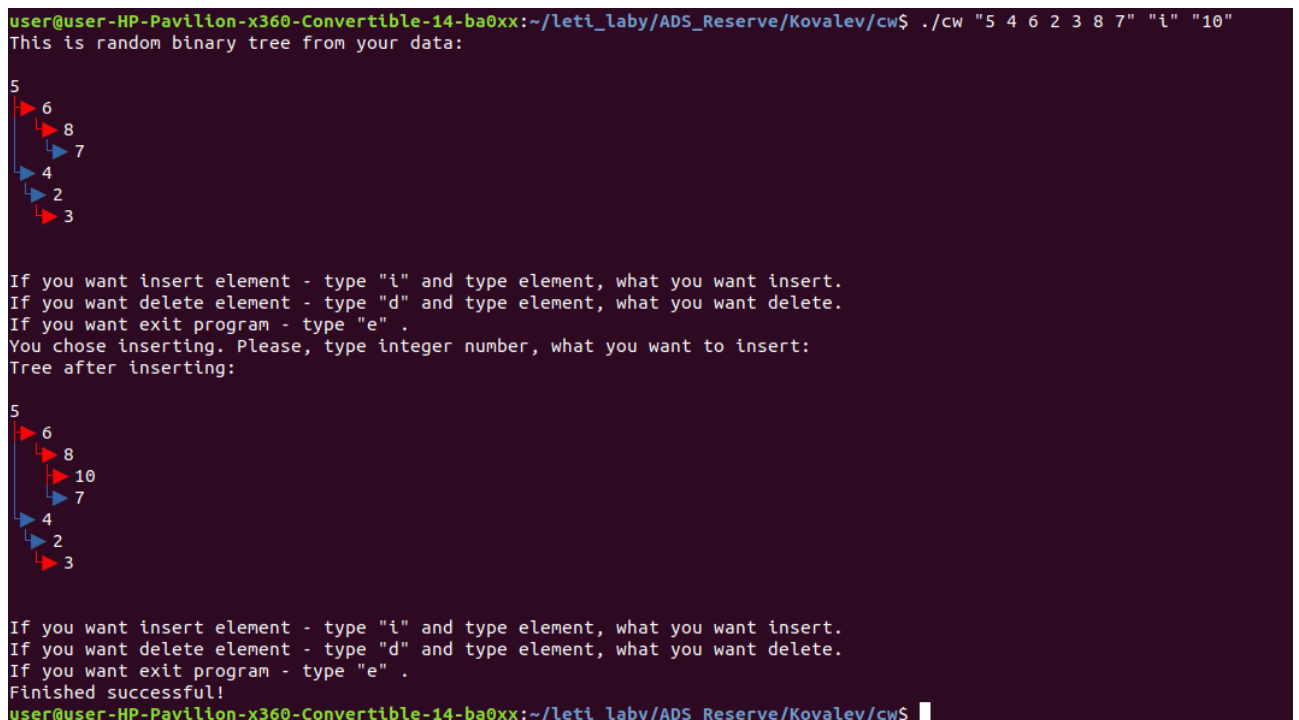
## 4. ТЕСТИРОВАНИЕ

В качестве входных данных программе сначала подается строка, состоящая из элементов типа *char* или *int*, вторым аргументом чего подается один символ-флажок, который означает, какое действие будет производиться, последним аргументом передается ключ, который надо удалить/вставить.

Для запуска необходимо прописать в терминале команду *make*, которая скомпилирует и соберет исполняемый файл, который называется *sw*. Для запуска скриптов необходимо прописать команду *python3 tester.py*. Скрипт запустит 10 тестов и выведет результаты в терминал. Чтобы запустить программу самостоятельно, необходимо передать ей в параметру строку по которой будет построено БДП, после чего один символ — *i* или *d*, а после — элемент, с которым будет производиться действие. В выводе программы красная стрелочка указывает на правого сына, а синяя — на левого.

Результаты тестирования представлены в приложении А.

Примеры запуска программы:



```
user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw$ ./cw "5 4 6 2 3 8 7" "i" "10"
This is random binary tree from your data:

5
├── 6
│   ├── 8
│   └── 7
└── 4
    ├── 2
    └── 3

If you want insert element - type "i" and type element, what you want insert.
If you want delete element - type "d" and type element, what you want delete.
If you want exit program - type "e" .
You chose inserting. Please, type integer number, what you want to insert:
Tree after inserting:

5
├── 6
│   ├── 8
│   └── 10
└── 4
    ├── 2
    └── 3

If you want insert element - type "i" and type element, what you want insert.
If you want delete element - type "d" and type element, what you want delete.
If you want exit program - type "e" .
Finished successful!
user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw$
```

Рисунок 1 — Пример запуска программы

```

user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw$ ./cw "8 3 1 6 4 7 10 14 13" "i" "10"
This is random binary tree from your data:

      8
     /
    10
   /  \
  14   13
 /
3
 /
6
 /  \
7   4
 /
1

If you want insert element - type "i" and type element, what you want insert.
If you want delete element - type "d" and type element, what you want delete.
If you want exit program - type "e" .
You chose inserting. Please, type integer number, what you want to insert:
Tree after inserting:

      8
     /
    10
   /  \
  14   13
 /
3
 /
6
 /  \
7   4
 /
1

If you want insert element - type "i" and type element, what you want insert.
If you want delete element - type "d" and type element, what you want delete.
If you want exit program - type "e" .
Finished successful!
user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw$

```

Рисунок 2 — Пример запуска программы

```

user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw$ ./cw "9 6 17 3 8 16 20 1 4 7 12 19 21 2 5 11 14 18 10 13 15" "i" "10"
This is random binary tree from your data:

      9
     /
    17
   /  \
  20   21
 /  \  /  \
19  18 16  12
 /  \  /  \
14  15 13  11
 /  \  /  \
6   10 8   7
 /  \  /  \
3   4 1   2
 /  \
1   5

      9
     /
    17
   /  \
  20   21
 /  \  /  \
19  18 16  12
 /  \  /  \
14  15 13  11
 /  \  /  \
6   10 8   7
 /  \  /  \
3   4 1   2
 /  \
1   5

```

Рисунок 3 — Пример запуска программы



## 5. ИССЛЕДОВАНИЕ

### 5.1. План исследования

План исследования состоит из нескольких шагов:

1. Генерация входных данных
2. Запуск функций вставки и удаления элемента в случайное БДП, с последующим подсчетом количества операций.
3. Анализ и интерпретация данных

### 5.2. Генерация входных данных

Для генерации входных данных и последующего тестирования был написан класс *researchData*. У данного класса были написаны методы — генераторы *worstGenerator()* и *shuffle()*. Первый метод генерировал массив упорядоченных элементов, второй метод — упорядоченный массив случайных элементов, после чего случайным образом перемешивал их. В дальнейшем, вектора элементов, полученные из этих функций, использовались в исследовании в четырех методах, каждый из которых исследует конкретную операцию в конкретном случае: *insertAverageResearch()*, *insertWorstResearch()*, *deleteAverageResearch()*, *deleteWorstResearch()*.

### 5.3. Теоретическая оценка сложности алгоритмов

В среднем случае операция вставки элемента в случайное БДП имеет сложность  $O(\log^2(n))$ , в худшем случае -  $O(n)$ . Худший случай представляет собой вставку элемента в дерево, которое построено по возрастающей или убывающей последовательности.

В среднем случае, операция удаления элемента из случайного БДП имеет сложность  $O(\log^2(n))$ , в худшем случае -  $O(n)$ . Худший случай представляет собой удаление элемента из дерева, которое построено по возрастающей или убывающей последовательности.

#### 5.4. Исследование вставки и удаления в среднем случае

Для исследования операции вставки в среднем случае генерировались массивы данных, по которым строилось случайное БДП и высчитывалось количество вызовов функции вставки элемента.

Сначала генерировался вектор чисел от 1 до  $n$  ( $n$  задавалось пользователем), после чего вектор случайным образом перемешивался. После чего происходила последовательная вставка элементов вектора в дерево. Высчитывалось количество узлов дерева и количество вызовов функции вставки, соответствующее количеству узлов. Далее, по полученным данным строился график.

Результаты представлены ниже на рисунке 1.

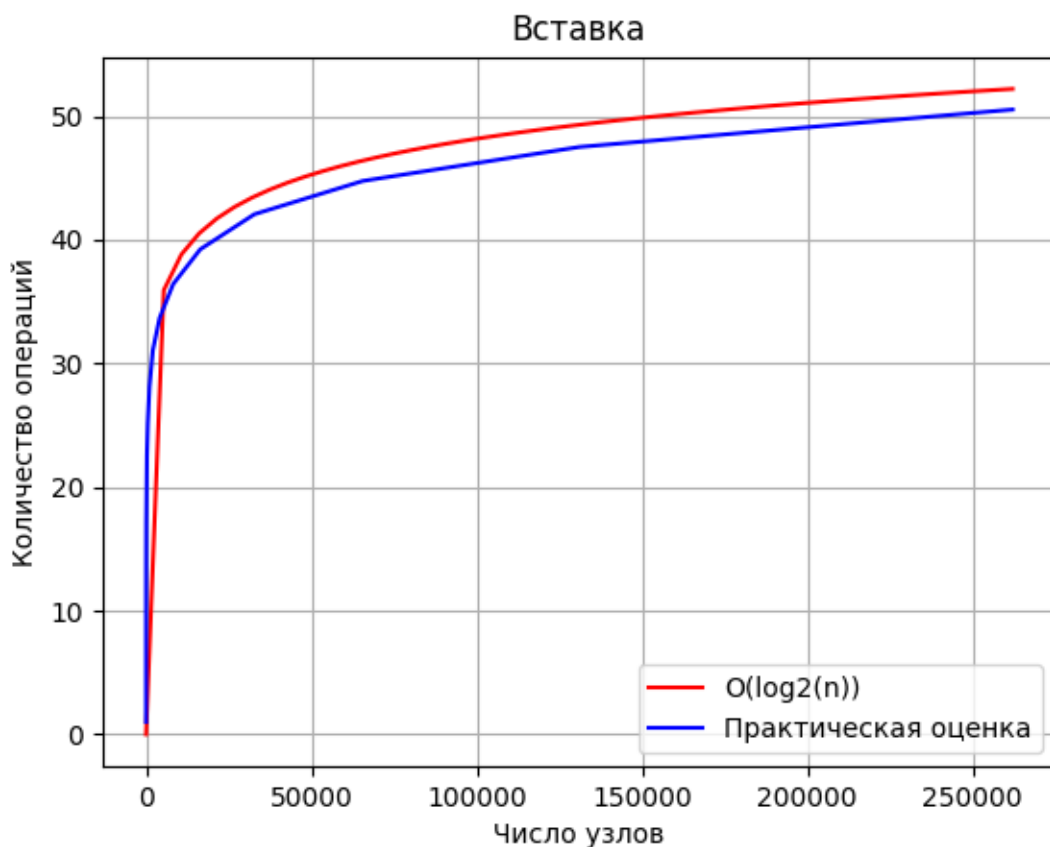


Рисунок 4 — Сложность вставки в среднем случае

Для исследования операции удаления в среднем случае генерировались массивы данных, по которым строилось случайное БДП и высчитывалось количество вызовов функции удаления элемента.

Сначала генерировался вектор чисел от 1 до  $n$  ( $n$  задавалось пользователем), после чего данный массив перемешивался. По данному массиву строилось случайное БДП, после чего происходило последовательное удаление элементов дерева. Высчитывалось количество узлов дерева и количество вызовов функции удаления, соответствующее количеству узлов. Далее, по полученным данным строился график.

Результаты представлены ниже на рисунке 3.

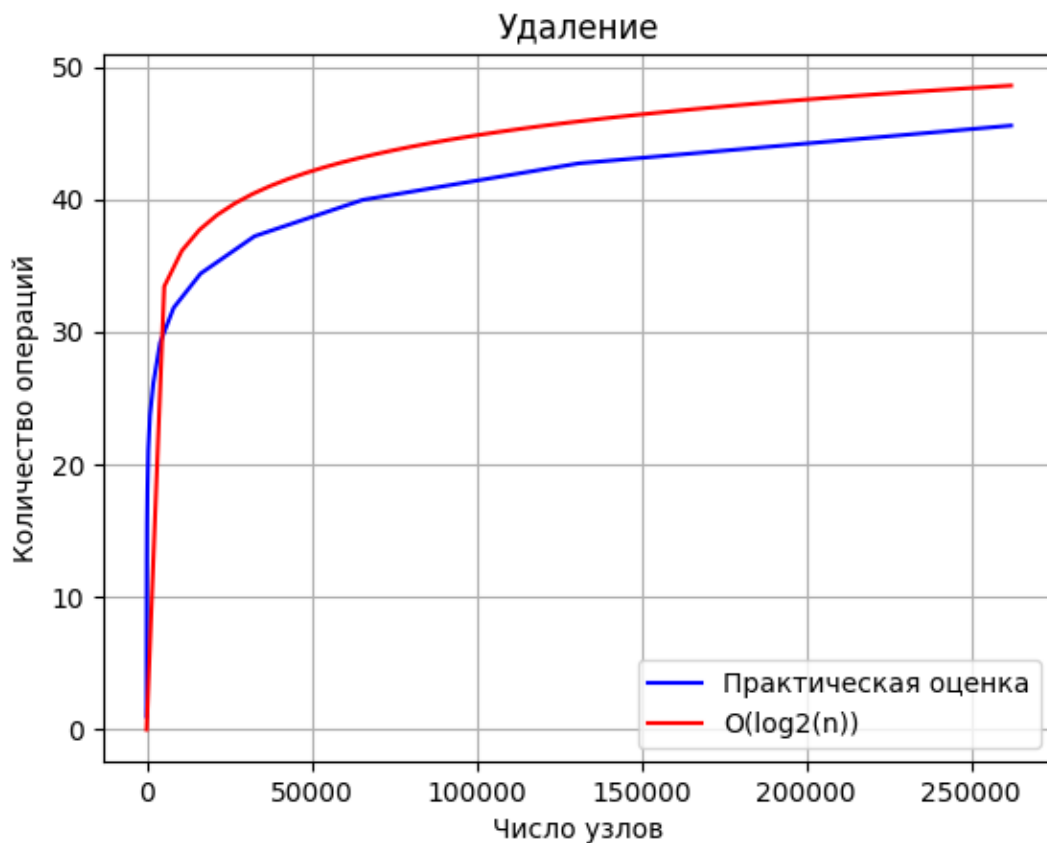


Рисунок 5 — Сложность удаления в среднем случае

### 5.5. Исследование вставки и удаления в худшем случае

Для исследования операции вставки в среднем случае генерировались массивы данных, по которым строилось случайное БДП и высчитывалось количество вызовов функции удаления элемента.

Сначала генерировался вектор чисел от 1 до  $n$  ( $n$  задавалось пользователем), после чего данный массив сортировался. По данному массиву строилось БДП. Из-за того, что последовательность была упорядоченная, сложность работы алгоритма удаления возрастала до  $O(n)$ . Высчитывалось количество узлов дерева и количество вызовов функции удаления, соответствующее количеству узлов. Далее, по полученным данным строился график.

Результаты представлены ниже на рисунке 3.

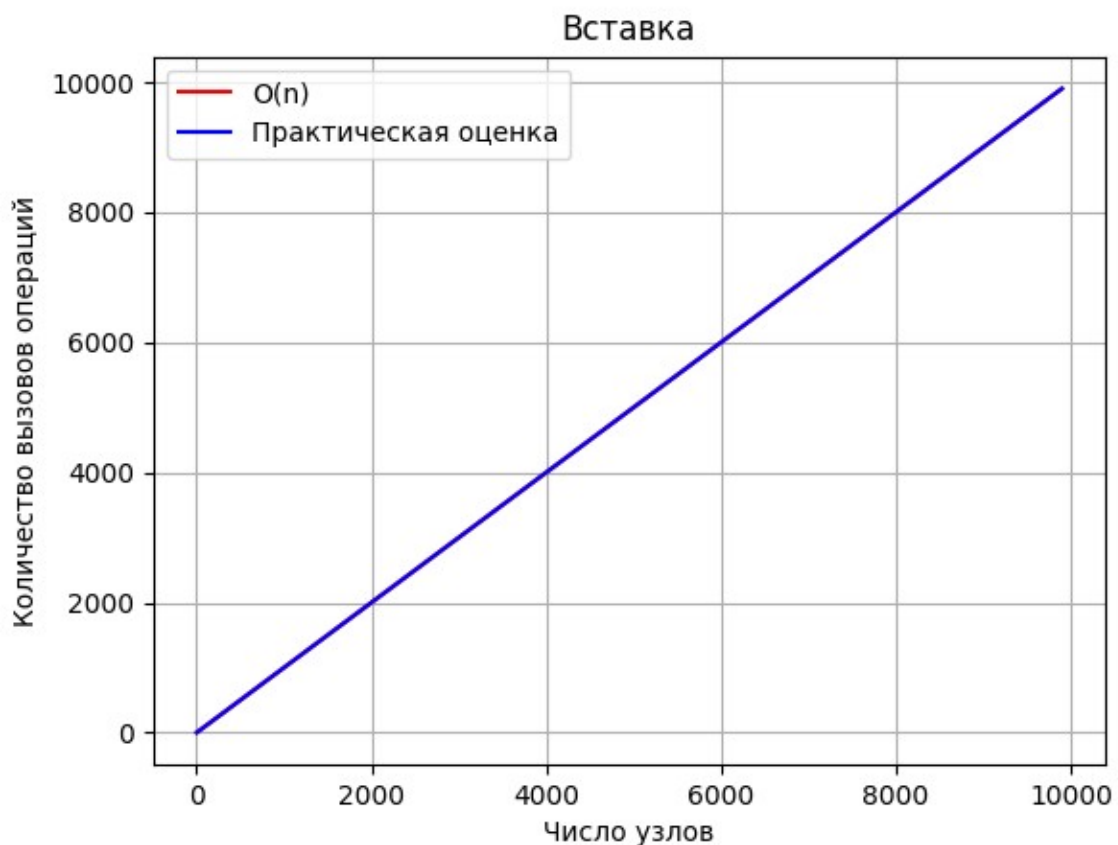


Рисунок 6 — Сложность вставки в худшем случае

Приближение графика выше представлено на рисунке 4 ниже:

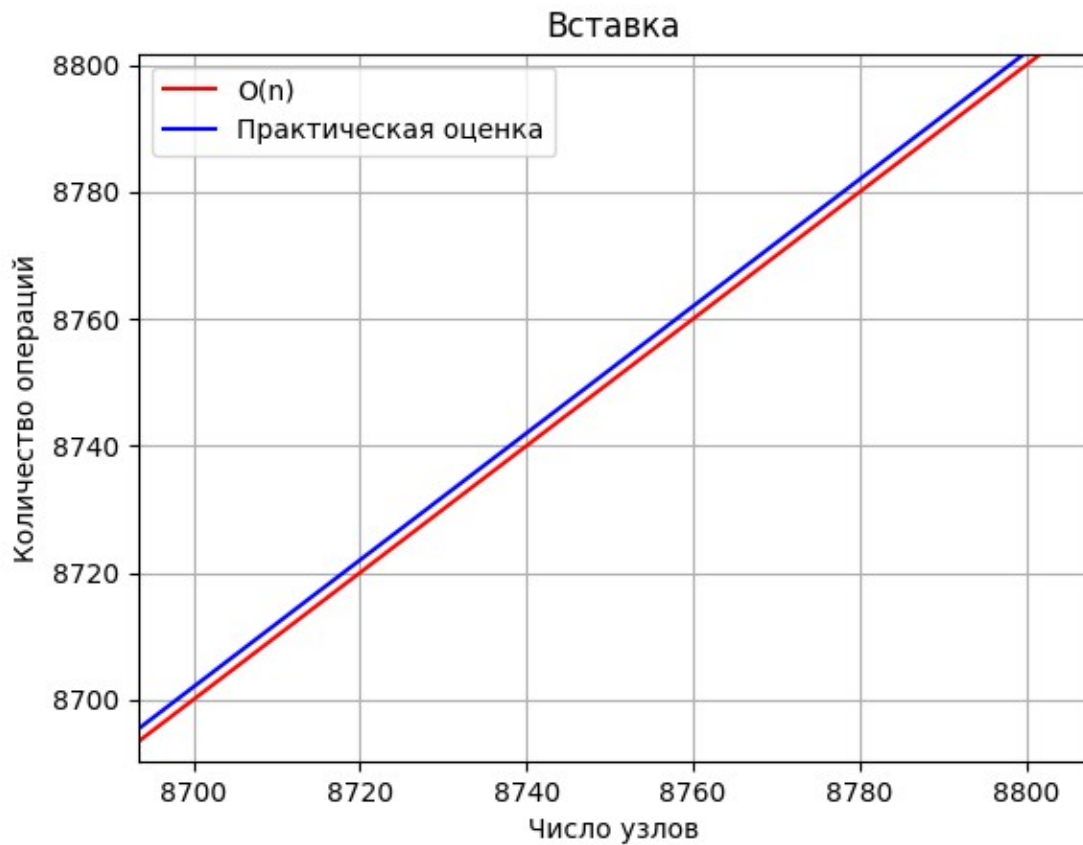


Рисунок 7 — Сложность вставки в худшем случае

Небольшое отклонение практической оценки от графика происходит из-за того, что в случайном БДП изначально нет элементов и в итоговом файле количество вызовов функции вставки оказывается всегда на 1 больше количества узлов в дереве до вставки.

Для исследования операции удаления в худшем случае генерировались отсортированные массивы данных, по которым строилось случайное БДП и высчитывалось количество вызовов функции удаления элемента.

Сначала генерировался вектор чисел от 1 до  $n$  ( $n$  задавалось пользователем), после чего данный массив сортировался. По полученному массиву строилось случайное БДП, после чего, в качестве массива элементов для удаления, брался перевернутый исходный массив. В результате происходило последовательное удаление элементов дерева, начиная с последнего элемента дерева. Таким образом, получалась линейная сложность. Высчитывалось количество узлов дерева и количество вызовов функции удаления, соответствующее количеству узлов. Далее, по полученным данным строился график.

Результаты представлены ниже на рисунке 5.

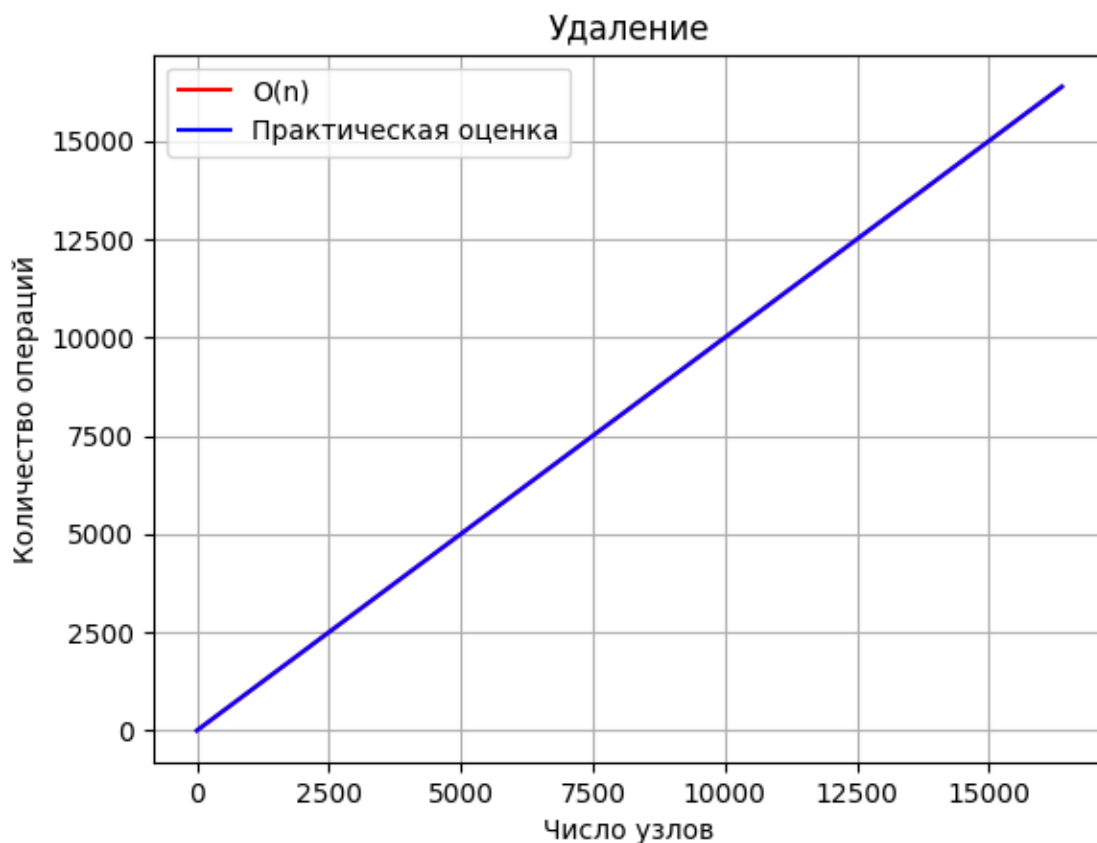


Рисунок 8 — Сложность удаления в худшем случае

Приближение графика выше представлено на рисунке 6 ниже:

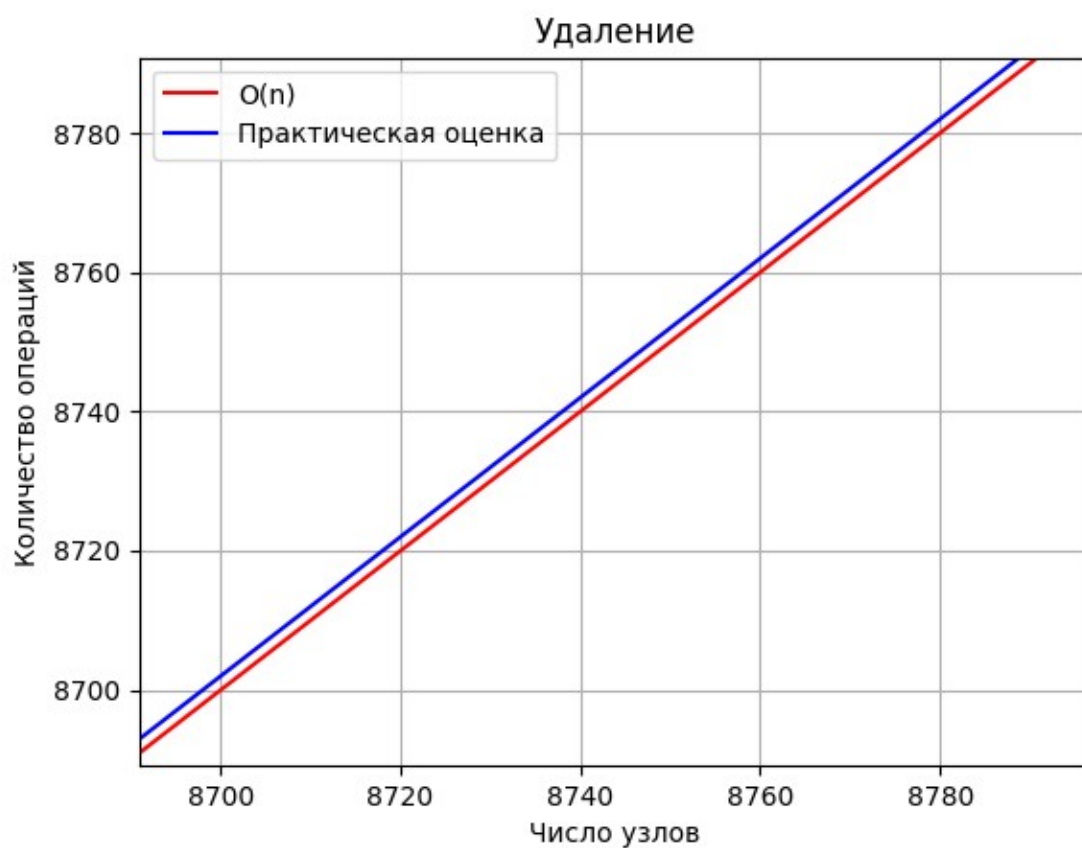


Рисунок 9 — Сложность удаления в худшем случае

Небольшое отклонение практической оценки от графика происходит из-за того, что количество вызовов функции удаления оказывается всегда на 1 больше, чем элементов в дереве.

## 5.6. Исследование зависимости высоты дерева от числа узлов

Для измерения высоты построенного случайного БДП была написан метод *findHeight()*, который измерял высоту дерева. В среднем случае дерево строилось из случайной последовательности, и его высота не превосходит  $\log_2(n)$ . В худшем случае дерево вырождается в линейный список и его высота равна числу элементов. Интерпретация данных представлена на рисунках 10 и 11 ниже.

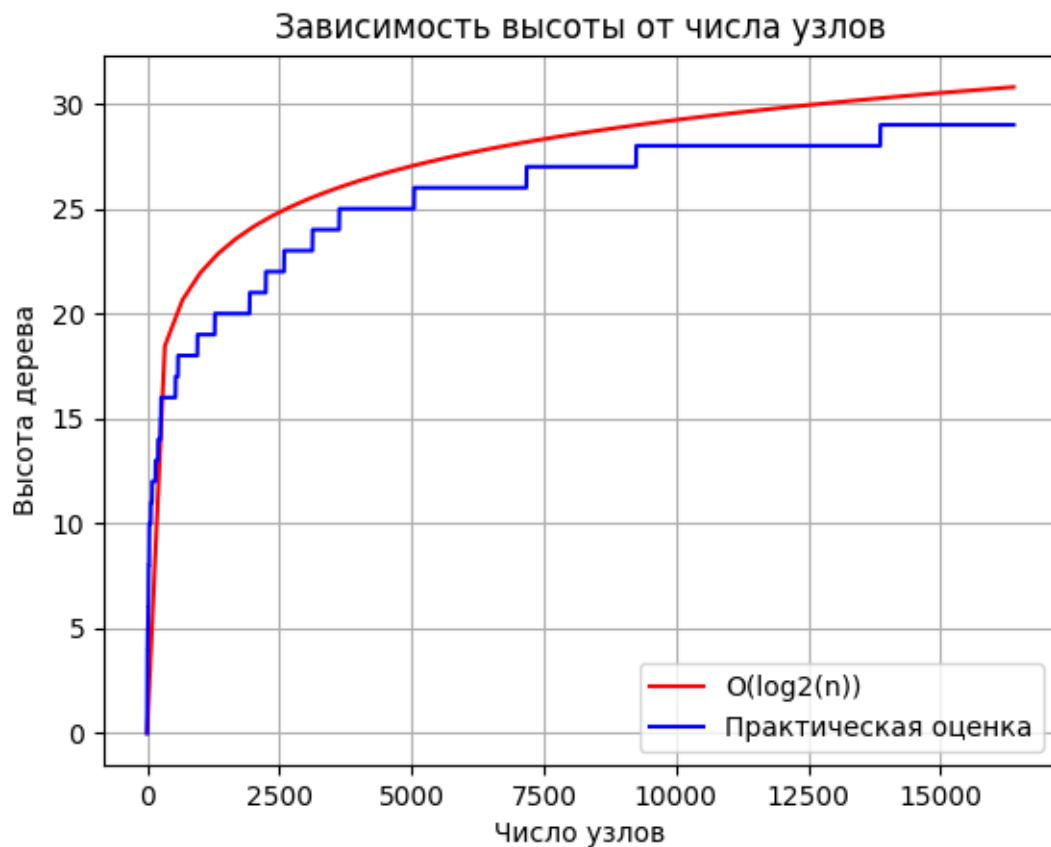


Рисунок 10 — Зависимость высоты от числа узлов в среднем случае



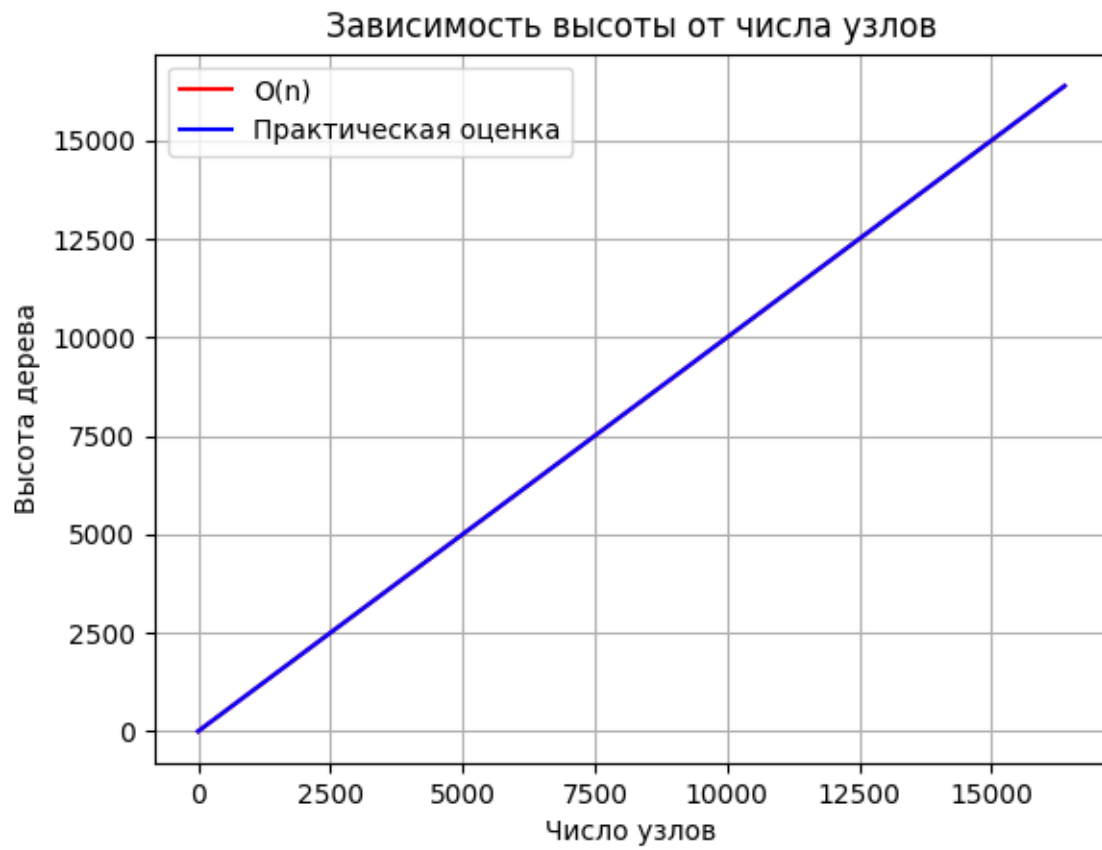


Рисунок 11 — Зависимость высоты от числа узлов в худшем случае

### 5.7. Итоги исследования

В процессе исследования была изучена теория по сложностям алгоритмов в случайном БДП, а также проведены практические исследования, которые подтвердили теоретическую оценку сложности алгоритмов. По графикам отчетливо видно, что операции вставки и удаления в среднем случае действительно имеют сложность  $O(\log^2(n))$ . В худшем случае, оба алгоритма работают уже за другую сложность —  $O(n)$ . Худший случай достигается тогда, когда случайное БДП строится по упорядоченной последовательности. В течение проведения исследования была накоплена большая статистика, которая содержит в себе замеры количества вызовов функций и количество узлов дерева, при котором был сделан данный замер. Все результаты были интерпретированы в виде графиков и сопоставлены с теоретическими оценками.

## **ЗАКЛЮЧЕНИЕ**

В результате исследования была экспериментально подтверждена теоретическая оценка сложности алгоритмов вставки и удаления в случайном БДП. Для проведения тестирования был написан класс исследования, методы которого генерировали наборы данных для тестирования и проводили тесты. Итогом работы данного класса было накопление статистики, благодаря которой, получилось достаточно точно интерпретировать полученные данные.

Была разработана программа на языке программирования C++, которая представляет собой случайное БДП, поддерживающее операции вставки и удаления элементов.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. [http://www.apmath.spbu.ru/ru/staff/pogozhev/files/dvoichnye\\_derevya\\_poiska.pdf](http://www.apmath.spbu.ru/ru/staff/pogozhev/files/dvoichnye_derevya_poiska.pdf)
2. <https://habr.com/ru/post/267855/>
3. [https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5\\_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE\\_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0)

# ПРИЛОЖЕНИЕ А

## РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

№ п/п	Входные данные	Выходные данные	Комментарии
1	a b c d e t i m	<pre> user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw\$ ./cw "a b c d e t" "i" "n" This is random binary tree from your data: a ├── b │   ├── c │   │   ├── d │   │   └── e │   └── t └──  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . You chose inserting. Please, type integer number, what you want to insert: Tree after inserting: a ├── b │   ├── c │   │   ├── d │   │   └── e │   └── t └── n  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw\$ </pre>	
2	a b c d i m	<pre> user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw\$ ./cw "a b c d" "i" "n" This is random binary tree from your data: a ├── b │   ├── c │   └── d └──  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . You chose inserting. Please, type integer number, what you want to insert: Tree after inserting: a ├── b │   ├── c │   └── d └── n  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . Finished successful! user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw\$ </pre>	
3	1 2 3 4 5 8 9 7 456 d 8	<pre> user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw\$ ./cw "1 2 3 4 5 8 9 7 456" "d" "8" This is random binary tree from your data: 1 ├── 2 │   ├── 3 │   │   ├── 4 │   │   └── 5 │   └── 8 │       ├── 9 │       └── 456 └── 7  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . You chose deleting. Please, type integer number, what you want to delete: Tree after deleting: 1 ├── 2 │   ├── 3 │   │   ├── 4 │   │   └── 5 │   └── 9 │       ├── 456 │       └── 7 └──  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . Finished successful! user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw\$ </pre>	

4	1 2 3 4 5 d 1	<pre> user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/letl_laby/ADS_Reserve/Kovalev/cw\$ ./cw "1 2 3 4 5" "d" "1" This is random binary tree from your data: 1 ├── 2 │   ├── 3 │   └── 4 │       └── 5 └── 5  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . You chose deleting. Please, type integer number, what you want to delete: Tree after deleting: 2 ├── 3 │   ├── 4 │   └── 5 └── 5  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . Finished successfully! user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/letl_laby/ADS_Reserve/Kovalev/cw\$ </pre>	
5	1 2 3 4 5 8 d 8	<pre> user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/letl_laby/ADS_Reserve/Kovalev/cw\$ ./cw "1 2 3 4 5 8" "d" "8" This is random binary tree from your data: 1 ├── 2 │   ├── 3 │   └── 4 │       └── 5 │           └── 8 └── 5  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . You chose deleting. Please, type integer number, what you want to delete: Tree after deleting: 1 ├── 2 │   ├── 3 │   └── 4 │       └── 5 └── 5  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . Finished successfully! user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/letl_laby/ADS_Reserve/Kovalev/cw\$ </pre>	
6	1 2 3 4 5 d 5	<pre> user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/letl_laby/ADS_Reserve/Kovalev/cw\$ ./cw "1 2 3 4 5" "d" "5" This is random binary tree from your data: 1 ├── 2 │   ├── 3 │   └── 4 │       └── 5 └── 5  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . You chose deleting. Please, type integer number, what you want to delete: Tree after deleting: 1 ├── 2 │   ├── 3 │   └── 4 └── 4  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . Finished successfully! user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/letl_laby/ADS_Reserve/Kovalev/cw\$ </pre>	
7	1 2 3 4 5 8 9 7 456 d 8	<pre> user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/letl_laby/ADS_Reserve/Kovalev/cw\$ ./cw "1 2 3 4 5 8 9 7 456" "d" "8" This is random binary tree from your data: 1 ├── 2 │   ├── 3 │   └── 4 │       ├── 5 │       ├── 8 │       ├── 9 │       └── 456 └── 7  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . You chose deleting. Please, type integer number, what you want to delete: Tree after deleting: 1 ├── 2 │   ├── 3 │   └── 4 │       ├── 5 │       ├── 9 │       └── 456 └── 7  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . Finished successfully! user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/letl_laby/ADS_Reserve/Kovalev/cw\$ </pre>	

8	1 6 3 4 11 8 9 7 23 i 5	<pre> user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw\$ ./cw "1 6 3 4 11 8 9 7 23" "i" "5" This is random binary tree from your data: 1 ├── 6 │   ├── 11 │   │   ├── 23 │   │   │   ├── 8 │   │   │   └── 9 │   │       └── 7 │   └── 3 │       └── 4 └── 3  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . You chose inserting. Please, type integer number, what you want to insert: Tree after inserting: 1 ├── 6 │   ├── 11 │   │   ├── 23 │   │   │   ├── 8 │   │   │   └── 9 │   │       └── 7 │   └── 3 │       ├── 4 │       └── 5 └── 3  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . Finished successful! user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw\$ </pre>	
9	10 24 7 6 5 67 87 23 i 2	<pre> user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw\$ ./cw "10 24 7 6 5 67 87 23" "i" "2" This is random binary tree from your data: 10 ├── 24 │   ├── 67 │   │   ├── 87 │   │   └── 23 │   └── 7 │       ├── 6 │       └── 5 └── 7  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . You chose inserting. Please, type integer number, what you want to insert: Tree after inserting: 10 ├── 24 │   ├── 67 │   │   ├── 87 │   │   └── 23 │   └── 7 │       ├── 6 │       └── 5 └── 2  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . Finished successful! user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw\$ </pre>	
10	10 24 7 6 54 67 87 2 23 d 67	<pre> user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw\$ ./cw "10 24 7 6 5 4 67 87 2 23" "d" "67" This is random binary tree from your data: 10 ├── 24 │   ├── 54 │   │   ├── 67 │   │   └── 87 │   └── 7 │       ├── 6 │       └── 2 └── 7  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . You chose deleting. Please, type integer number, what you want to delete: Tree after deleting: 10 ├── 24 │   ├── 54 │   │   ├── 87 │   │   └── 23 │   └── 7 │       ├── 6 │       └── 2 └── 7  If you want insert element - type "i" and type element, what you want insert. If you want delete element - type "d" and type element, what you want delete. If you want exit program - type "e" . Finished successful! user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/ADS_Reserve/Kovalev/cw\$ </pre>	

## ПРИЛОЖЕНИЕ Б

### ПРОГРАММНЫЙ КОД

```
#include <iostream>
#include <string>
#include <sstream>
#include <memory>
#include <chrono>
#include <vector>
#include <fstream>
#include <algorithm>

static int counterInsert = 0; //счетчик вызова функции вставки
static int deleterInsert = 0; //счетчик вызова функции удаления
template <typename T>
class Node{
public:
    int counter;
    T data;
    std::shared_ptr<Node<T>> left {nullptr};
    std::shared_ptr<Node<T>> right {nullptr};
    Node(){
        counter = 0;
    }
};

template <typename T>
class BSTree {
public:

    std::shared_ptr<Node<T>> head {nullptr};

    BSTree() {

    }

    BSTree(std::string &input) {
        std::istringstream s(input);
        T elem;
        while (s >> elem) {
            searchAndInsert(elem, head);
        }
    }

    ~BSTree() = default;

    void searchAndInsert(T elem, std::shared_ptr<Node<T>>& ptr)
    { //поиск и вставка элемента
        counterInsert++;
        if (!ptr) {
            ptr = std::make_shared<Node<T>>();
        }
    }
};
```



```

        ptr->data = elem;
        ptr->counter = 1;
    } else if (elem < ptr->data) {
        searchAndInsert(elem, ptr->left);
    } else if (elem > ptr->data) {
        searchAndInsert(elem, ptr->right);
    } else {
        ptr->counter++;
    }
}

void searchAndDelete(T elem, std::shared_ptr<Node<T>>& ptr){//
поиск и удаление элемента
    deleterInsert++;
    if(ptr){
        if(ptr == head && !ptr->left && !ptr->right){
            head = nullptr;
            return;
        }
        if(elem < ptr->data){
            searchAndDelete(elem, ptr->left);
        }else if (elem > ptr->data){
            searchAndDelete(elem, ptr->right);
        }else if (elem == ptr->data){
            if(!ptr->left && !ptr->right){
                ptr = nullptr;//findParent(elem, head);
            }else if (ptr->right){
                T leftSon = findMin(ptr->right, ptr->counter);
                searchAndDelete(leftSon, head);
                ptr->data = leftSon;
            }else if (ptr->left){
                T rightSon = findMax(ptr->left, ptr->counter);
                searchAndDelete(rightSon, head);
                ptr->data = rightSon;
            }
        }
    }else{
        std::cout << "There is no element " << elem << " in
tree.\n";
    }
}

T findMin(std::shared_ptr<Node<T>> p, int& ct){//поиск
минимального элемента
    std::shared_ptr<Node<T>> temp =
std::make_shared<Node<T>>();
    temp = p;
    while(temp->left){
        temp = temp->left;
    }
    ct = temp->counter;
    return temp->data;
}

```

```

    T findMax(std::shared_ptr<Node<T>> p, int& ct){//поиск
максимального элемента
        std::shared_ptr<Node<T>> temp =
std::make_shared<Node<T>>();
        temp = p;
        while(temp->right){
            temp = temp->right;
        }
        ct = temp->counter;
        return temp->data;
    }

    void printBST(std::shared_ptr<Node<T>> bt, std::string str="")
{//печать дерева с псевдографикой
        if(!bt){
            return;
        }
        std::string cpyStr = str;
        std::cout << bt->data << '\n';
        if(bt->right){
            std::cout << str;
        }
        if(!bt->left && bt->right){
            std::cout << "\x1b[31m└─ \x1b[0m";
        }
        if(bt->left && bt->right){
            std::cout << "\x1b[31m└─ \x1b[0m";
        }
        if(bt->left){
            printBST(bt->right, str.append("\x1b[34m└─ \x1b[0m"));
        }else{
            printBST(bt->right, str.append(" "));
        }
        if(bt->left){
            std::cout << cpyStr;
        }
        if(bt->left){
            std::cout << "\x1b[34m└─ \x1b[0m";
        }
        printBST(bt->left, cpyStr.append(" "));
    }

    void print_Tree(std::shared_ptr<Node<T>> bt,int level)//печать
дерева
    {
        if(bt)
        {
            print_Tree(bt->left,level + 1);
            for(int i = 0; i< level; i++){
                std::cout<<"    ";
            }
            std::cout << bt->data << std::endl;
        }
    }

```

```

        print_Tree(bt->right, level + 1);
    }
}

};

bool digitChecker(std::string& s){//чекер для строки с данными
    int len = s.length();
    int counter = 0;
    for(int i = 0; i < len; i++){
        if(isdigit(s[i]) || s[i] == ' '){
            counter++;
        }else{
            break;
        }
    }
    return counter==len;
}

bool charChecker(std::string& s){//чекер для строки с данными
    int len = s.length();
    int counter = 0;
    for(int i = 0; i < len; i++){
        if(isalpha(s[i]) || s[i] == ' '){
            counter++;
        }else{
            break;
        }
    }
    return counter==len;
}

class researchData{//класс исследования
    std::ofstream research1;
    std::ofstream research2;
    std::ofstream research3;
    std::ofstream research4;
    std::string name1 = "research_insertions_average.txt";
    std::string name2 = "research_insertions_worstCase.txt";
    std::string name3 = "research_deletions_average.txt";
    std::string name4 = "research_deletions_worstCase.txt";
public:

    void insertAverageResearch(int size = 262144){// средний
случай вставки
        srand(time(0));
        research1.open(name1);
        std::vector<int> vec;
        auto* bt = new BSTree<int>;
        shuffle(size, vec);
        for(int i = 0; i < size; i++) {
            counterInsert = 0;
            bt->searchAndInsert(vec[i], bt->head);
        }
    }
};

```

```

        research1 << i << " " << counterInsert << "\n";
    }
    research1.close();
    delete bt;
}

void insertWorstResearch(int size = 16384){// худший случай
вставки
    srand(time(0));
    research2.open(name2);
    std::vector<int> vec;
    auto* bt = new BSTree<int>;
    worstGenerator(size, vec);
    for(int i = 0; i < size; i++) {
        counterInsert = 0;
        bt->searchAndInsert(vec[i], bt->head);
        research2 << i << " " << counterInsert << "\n";
    }
    research2.close();
    delete bt;
}

void deleteAverageResearch(int size = 16384){// средней случай
удаления
    srand(time(0));
    research3.open(name3);
    std::vector<int> vec;
    auto* bt = new BSTree<int>;
    shuffle(size, vec);
    for(int j = 0; j < size; j++){
        bt->searchAndInsert(vec[j], bt->head);
    }
    for(int i = 0; i < size; i++) {
        deleterInsert = 0;
        bt->searchAndDelete(vec[vec.size() - 1 - i], bt-
>head);
        research3 << size - 1 - i << " " << deleterInsert <<
"\n";
    }
    research3.close();
    delete bt;
}

void deleteWorstResearch(int size = 16384){// худший случай
удаления
    srand(time(0));
    research4.open(name4);
    std::vector<int> vec;
    auto* bt = new BSTree<int>;
    worstGenerator(size, vec);
    for(int j = 0; j < size; j++){
        bt->searchAndInsert(vec[j], bt->head);
    }

```

```

    }
    for(int i = 0; i < size; i++) {
        deleterInsert = 0;
        bt->searchAndDelete(vec[vec.size() - 1 - i], bt-
>head);
        research4 << size - 1 - i << " " << deleterInsert <<
"\n";
    }
    research4.close();
    delete bt;
}

void shuffle(int count, std::vector<int>& vec) { //массив
случайных элементов
    for(int i = 0; i < count; i++){
        vec.push_back(i);
    }
    int temp;
    for(int i = count - 1; i >= 1; i--) {
        int j = rand()%(count); // обменять значения data[j] и
data[i]
        temp = vec[j];
        vec[j] = vec[i];
        vec[i] = temp;
    }
}

void worstGenerator(int count, std::vector<int>&
vec) { //упорядоченный массив
    for (int i = 0; i < count; i++){
        vec.push_back(i);
    }
}

};

int main() {
    std::string input;
    getline(std::cin, input);
    if(isdigit(input[0])) {
        if (digitChecker(input)) {
            BSTree<int> bts;
            std::istringstream s(input);
            int elem;
            while (s >> elem) {
                bts.searchAndInsert(elem, bts.head);
            }
            std::cout << "This is random binary tree from your
data:\n\n";
            bts.printBST(bts.head);
            std::cout << "\n\n";
            char choose;
            while (true) {

```

```

        std::cout << "If you want insert element -
type \"i\" and type element, what you want insert.\n";
        std::cout << "If you want delete element -
type \"d\" and type element, what you want delete.\n";
        std::cout << "If you want exit program -
type \"e\" .\n";
        std::cin >> choose;
        if (choose == 'i') {
            std::cout << "You chose inserting. Please,
type integer number, what you want to insert:\n";
            int el;
            std::cin >> el;
            bts.searchAndInsert(el, bts.head);
            std::cout << "Tree after inserting:\n\n";
            bts.printBST(bts.head);
            std::cout << "\n\n";
        } else if (choose == 'd') {
            std::cout << "You chose deleting. Please, type
integer number, what you want to delete:\n";
            int el;
            std::cin >> el;
            bts.searchAndDelete(el, bts.head);
            std::cout << "Tree after deleting:\n\n";
            bts.printBST(bts.head);
            std::cout << "\n\n";
        } else if (choose == 'e') {
            break;
        }
    }
} else {
    std::cout << "Incorrect input!\n";
    return 0;
}
} else if (isalpha(input[0])) {
    if (charChecker(input)) {
        BSTree<char> bts;
        std::istringstream s(input);
        char elem;
        while (s >> elem) {
            bts.searchAndInsert(elem, bts.head);
        }
        std::cout << "This is random binary tree from your
data:\n\n";
        bts.printBST(bts.head);
        std::cout << "\n\n";
        char choose;
        while (true) {
            std::cout << "If you want insert element -
type \"i\" and type element, what you want insert.\n";
            std::cout << "If you want delete element -
type \"d\" and type element, what you want delete.\n";
            std::cout << "If you want exit program -
type \"e\" .\n";

```

```

        std::cin >> choose;
        if(choose == 'i'){
            std::cout << "You chose inserting. Please,
type integer number, what you want to insert:\n";
            char el;
            std::cin >> el;
            bts.searchAndInsert(el, bts.head);
            std::cout << "Tree after inserting:\n\n";
            bts.printBST(bts.head);
            std::cout << "\n\n";
        }else if(choose == 'd'){
            std::cout << "You chose deleting. Please, type
integer number, what you want to delete:\n";
            char el;
            std::cin >> el;
            bts.searchAndDelete(el, bts.head);
            std::cout << "Tree after deleting:\n\n";
            bts.printBST(bts.head);
            std::cout << "\n\n";
        }else if (choose == 'e'){
            break;
        }
    }
    }else {
        std::cout << "Incorrect input!\n";
        return 0;
    }
}
std::cout << "Finished successful!\n";
return 0;
}

```