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

# ОТЧЕТ

**по лабораторной работе №4**
**по дисциплине «Алгоритмы и структуры данных»**
**ТЕМА: СОРТИРОВКИ**

Студент гр. 9382 _____ Кодуков А.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

**Цель работы**:

Познакомиться с одним из часто используемых на практике алгоритмов сортировки данных, оценить его достоинства и недостатки.

**Задание:**

*Индивидуальное задание 8:*

Быстрая сортировка, рекурсивная реализация. Во время сортировки массив должен быть в состоянии:

элементы <x, неотсортированные элементы, элементы >=x.

**Описание алгоритма:**

*Быстрая сортировка:*

Данная сортировка использует технику “разделяй и властвуй”.

Сначала выбирается *опорный элемент*. В данной реализации он выбирается как элемент из середины массива, чтобы ускорить работу на уже отсортированных массивах.

Затем необходимо, чтобы все элементы меньшие опорного элемента оказались слева от него, а большие – справа. Для этого заводятся два указателя: на начало и на конец массива. Затем левый указатель сдвигается вправо по массиву, пока не встретит элемент больше опорного или опорный, а правый аналогичным образом сдвигается влево для обнаружения элемента меньше опорного или опорного. Когда оба указателя установлены на необходимые места, соответствующие элементы меняются местами. Данные действия повторяются пока указатели не зайдут друг за друга на опорном элементе. После этого опорный элемент установлен на свое место в отсортированном массиве.

Далее необходимо отсортировать подмассивы слева и справа от него. Подразбиение будет продолжаться до массива из одного элемента. Когда алгоритм закончит работу, все элементы побывают опорными, а значит будут установлены на свои места и массив будет отсортирован.

*Достоинства:*

- Один из самых быстродействующий в среднем случае алгоритмов сортировки общего назначения
- Использует сравнительно мало памяти
- Довольно короткая и простая в реализации

*Недостатки:*

- Скорость может сильно ухудшится при неудачных входных данных (механизм выбора опорного элемента)
- Рекурсивная реализация может вызвать переполнение стека при неудачных входных данных

**Функции и структуры данных:**

Реализованные функции:

*Быстрая сортировка*

Сигнатура: `void QuickSort(int *A, int size, int lvl)`

Аргументы:

- A –указатель на первый элемент текущего массива
- size – размер текущего массива
- lvl – уровень рекурсии

Алгоритм:

- Выбор опорного элемента как элемента из середины массива
- Пока итерация не завершена
    - Сдвинуть левый указатель до элемента не больше опорного
    - Сдвинуть правый указатель до элемента не меньше опорного
    - Если левый указатель больше правого – закончить итерацию
    - Иначе, поменять элементы под левым и правым указателем местами
- Запустить алгоритм от получившихся левого и правого подмассивов

**Тестирование:**

| № | Входные данные | Результат |
|---|---|---|
| 1 | 10 11 23 2 45 -30 4 99 7 -100 8 51 9 1 -123 5 41 | Sorted: -123 -100 -30 1 2 4 5 7 8 9 10 11 23 41 45 51 99<br>Control: -123 -100 -30 1 2 4 5 7 8 9 10 11 23 41 45 51 99<br>Sorting is correct |
| 2 | 1 1 1 1 1 1 1 1 | Sorted: 1 1 1 1 1 1 1 1<br>Control: 1 1 1 1 1 1 1 1<br>Sorting is correct |
| 3 | 1 1 2 1 2 3 2 2 1 1 3 | Sorted: 1 1 1 1 1 2 2 2 2 3 3<br>Control: 1 1 1 1 1 2 2 2 2 3 3<br>Sorting is correct |
| 4 | 5 4 3 2 1 0 -1 -2 -3 -4 -5 | Sorted: -5 -4 -3 -2 -1 0 1 2 3 4 5<br>Control: -5 -4 -3 -2 -1 0 1 2 3 4 5<br>Sorting is correct |
| 5 | 0 1 0 1 0 1 0 1 0 1 | Sorted: 0 0 0 0 0 1 1 1 1 1<br>Control: 0 0 0 0 0 1 1 1 1 1<br>Sorting is correct |
| 6 | --1 | Wrong input |
| 7 | asd | Wrong input |
| 8 | 1a1 | Wrong input |

**Вывод:**

В результате выполнения работы был изучен и реализован алгоритм быстрой сортировки, а также выявлены его достоинства и недостатки.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

**tree.h**

```
#ifndef __TREE_H
#define __TREE_H

#include <cstdlib>
#include <iostream>
#include <fstream>
#include <queue>

template <class Elem>
class Tree {
 public:
  struct node {
    Elem info;
    Tree *lt;
    Tree *rt;

    node() {
      lt = nullptr;
      rt = nullptr;
    }

    node(const Elem &x, Tree *lst, Tree *rst) {
      info = x;
      lt = lst;
      rt = rst;
    }
  };

 private:
   node *Node = nullptr;

 public:
  Tree() {}

  Tree(node *N) {
    Node = N;
  }

  void Clear() {
    if (Node != nullptr) {
      if (Node->lt != nullptr) Node->lt->Clear();
      if (Node->rt != nullptr) Node->rt->Clear();
      delete Node;
      Node = nullptr;
    }
  }

  ~Tree() { Clear(); }

  Tree *Left() {
    if (Node == nullptr) {
      std::cout << "Error: Left(null) \n";
      exit(1);
    } else
      return Node->lt;
  }
```

```
  Tree *Right() {
    if (Node == nullptr) {
      std::cout << "Error: Right(null) \n";
      exit(1);
    } else
      return Node->rt;
  }
  node *NodePtr() {
    if (Node == nullptr) {
      std::cout << "Error: RootBT(null) \n";
      exit(1);
    } else
      return Node;
  }
  Elem GetNode() {
    if (Node == nullptr) {
      std::cout << "Error: RootBT(null) \n";
      exit(1);
    } else
      return Node->info;
  }
  // Checking if the tree is a binary search tree
  bool CheckSearchTree(float min, float max, int lvl) {
    bool L = true, R = true;
    Elem cur = GetNode();

    std::cout << "\n";
    for (int i = 0; i < lvl; i++) std::cout << "  ";
    std::cout << "ELEMENT: " << cur << "\n";

    for (int i = 0; i < lvl; i++) std::cout << "  ";
    std::cout << min << "(min)" << " <= " << cur << " <= " << max << "(max)" <<
" ? ";
    // Check current node
    if (cur > max || cur < min) {
      std::cout << "false\n";
      return false;
    }
    std::cout << "true\n";
    // Check left subtree
    if (Left() != nullptr) {
      for (int i = 0; i < lvl; i++) std::cout << "  ";
      std::cout << "Check left: (max -> " << cur << ")\n";
      // max -> cur
      L = Left()->CheckSearchTree(min, cur, lvl + 1);
    }
    if (!L)
      return false;
    // Check right subtree
    if (Right() != nullptr) {
      for (int i = 0; i < lvl; i++) std::cout << "  ";
      std::cout << "Check right: (min -> " << cur << ")\n";
      // min -> cur
      R = Right()->CheckSearchTree(cur, max, lvl + 1);
    }
    if (!R)
      return false;
    return true;
  }
  // Checking if the tree is a pyramid tree
  bool CheckPyramidTree() {
    std::queue<Tree<Elem> *> q;
```

```
    // Push root
    q.push(this);
    while (!q.empty()) {
      //Print queue
      std ::cout << "  Queue: ";
      std::queue<Tree<Elem> *> t = q;
      while (!t.empty()) {
        std::cout << t.front()->GetNode() << " ";
        t.pop();
      }
      std ::cout << "\n";

      // Check current node
      Tree<Elem> *cur = q.front();
      q.pop();
      std::cout << "  Element:" << cur->GetNode() << "\n";
      bool L = false, R = false;
      // Check left
      if (cur->Left() != nullptr) {
        std::cout << "  Left: " << cur->GetNode()
                  << " >= " << cur->Left()->GetNode()
                  << " ? ";
        L = cur->GetNode() >= cur->Left()->GetNode();
        std::cout << (L ? "true" : "false") << "\n";
        if (!L)
          return false;
        q.push(cur->Left());
      }
      // Check right
      if (cur->Right() != nullptr) {
        std::cout << "  Right: " << cur->GetNode()
                  << " >= " << cur->Right()->GetNode() << " ? ";
        R = cur->GetNode() >= cur->Right()->GetNode();
        std::cout << (R ? "true" : "false") << "\n";
        if (!R) return false;
        q.push(cur->Right());
      }
      std::cout << "\n";
    }
    return true;
  }
};
#endif // __TREE_H
```

**main.cpp**

```
/* Кодуков Александр 9382, в. 18д
 *
 * Бинарное дерево называется бинарным деревом поиска,
 * если для каждого его узла справедливо
 * : все элементы правого поддерева больше этого узла,
 *   а все элементы левого поддерева – меньше этого узла.Бинарное дерево
 *       называется пирамидой,
 *   если для каждого его узла справедливо
 * : значения всех потомков этого узла не больше,
 *   чем значение узла.Для заданного бинарного дерева с числовым типом
 *       элементов определить,
 *   является ли оно бинарным деревом поиска и является ли оно пирамидой.
 */
#include "tree.h"

template <typename Elem>
```

```
Tree<Elem> *Read(std::ifstream &f) {
  char ch;
  Elem e = 0;
  Tree<Elem> *p, *q;

  f >> ch;
  int d = 0;
  while (ch >= '0' && ch <= '9') {
    e = e * pow(10, d++) + ch - '0';
    f >> ch;
  }
  if (ch == '/')
    return NULL;
  else {
    p = Read<Elem>(f);
    q = Read<Elem>(f);
    typename Tree<Elem>::node *N = new typename Tree<Elem>::node(e, p, q);
    return new Tree<Elem>(N);
  }
}

template <typename Elem>
void Print(Tree<Elem> *q, long n) {
  long i;
  if (q != nullptr) {
    Print<Elem>(q->Right(), n + 5);
    for (i = 0; i < n; i++)
      std::cout << " ";
    std::cout << q->GetNode() << "\n";
    Print<Elem>(q->Left(), n + 5);
  }
}

int main() {
  Tree<int> *t;
  std::ifstream f("input.txt");
  t = Read<int>(f);
  if (t != nullptr) {
    f.close();
    Print(t, 0);
    std::cout << "Check search:\n";
    bool Search = t->CheckSearchTree(-INFINITY, INFINITY, 1);
    std::cout << "Search: " << (Search ? "true" : "false") << "\n\n";
    std::cout << "Check pyramid:\n";
    bool Pyramid = t->CheckPyramidTree();
    std::cout << "Pyramid: " << (Pyramid ? "true" : "false") << "\n";
    t->Clear();
  } else
    std::cout << "Wrong input";
}
```