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. 9304 _____ Тиняков С.А.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Изучить различные сортировки данных. Реализовать одну из них на языке программирования C++.

Задание.

Вариант 27.

Релизовать поиск k-ого минимума в массиве алгоритмом на основе разделения по медиане медиан пятёрок.

Выполнение работы.

Алгоритм работает следующим образом: для исходного массива выбирается опорный элемент, по которому массив разделяется на две части: левая, в которой элементы меньше опорного, и правая, в которой элементы больше или равны опроному. Если размер левой части больше или равен искомого индекса, то, значит, искомый элемент находится в левой части, и алгоритм вызывается для левой части. Иначе алгоритм вызывается для правой части. Выбор опорного элемента происходит следующим образом: исходный массив разбивается на группы по пять элементов(в последней может быть меньше пяти элементов). Затем элементы в каждой группе сортируются, и выбирается медиана. Из этих медиан составляется массив, в котором выбирается медиана алгоритмом, описанным выше.

На вход программа получает число*(n)* — размер массива, затем считывает *n* чисел, после чего следует индекс элемента, который надо найти. Индексы начинаются с единицы. Программа выводит ход выполнения программы и ответ. Вход и выход в функции окрашены в стандартный цвет, исходные данные, медианы и опорный элемент — синим, отсортированные данные — фиолетовым, разделённый массив — оранжевым, индекс который нужно найти и найденный элемент — красным.

Шаблонная функция *find_k_min* принимает указатель на первый и последний элемент и номер индекса, который нужно найти. Лямбда-функция *quickselect* производит поиск *k*-ого элемента в массиве. Лямбда-функция *pick_pivot* выбирает опорный элемент. Лямбда-функция *quicksort* осуществляет быструю сортировку.

Разработанный программный код см. в приложении А.

Тестирование.

Тестирование происходит при помощи скрипта. Для заданного размера случайным образом генерируется список чисел в диапазоне от *1* до *1000000000.* Затем происходит поиск по всем элементам, т. е. сначала программа вызывается для поиска первого элемента, затем для второго и так далее до последнего. Для проверки правильности работы была реализованна программа *check,* которая имеет такие же входные данные, как и основная прорамма, однако находит элемент при помощи сортировки библиотечной функцией *std::sort* и обращения по индексу.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	3 42 666 13 6	Error: K > size	Проверка, что если задать индекс, выходящий за пределы, программа выдаст ошибку

Выводы.

Были изучены различные сортировки и их реализации на языке программирования C++.

Была разработанна программа, которая находит *k*-й минимальный элемент в массиве. Алгоритм основан на разделении по медиане медиан пятёрок. Для сортировки малых групп использовалась быстрая сортировка.

Для удобства просмотра хода выполнения различные места выводятся разными цветами. При указании неправильного индекса выбрасывается исключение. В реализации алгоритма использовались такие возможности *17*-ого стандарта C++, как лямбда-функии и умные указатели.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Source/lab4.cpp

```
#include<iostream>
#include<memory>

#define PRINT
#ifdef PRINT
#define NORMAL "\x1b[0m"
#define ORIGIN "\x1b[34m"
#define SORTED "\x1b[35m"
#define SEPARETED "\x1b[33m"
#define FIND "\x1b[1;31m"
#define DEEP(x) (x*(std::string("|   ")))

std::string operator*(int x, std::string str){
    if(!x) return std::string();
    std::string res = str;
    for(int i = 1; i < x; i++)
        res += str;
    return res;
}

#endif

template<typename T>
T find_k_min(T* start, T* end, size_t k){
    if (k > end-start) throw std::logic_error("K > size");
    size_t len = end - start;
    int deep = -1;
      auto quicksort = [](T* start, T* end, auto&& quicksort)-
>void{
        if(start >= end) return;
        size_t left = 0, right = end - start, len = right+1;
        T pivot = start[len/2];
        while(left <= right){
            while((left +1) < len && start[left] < pivot) left++;
            while(right > 0 && start[right] > pivot) right--;
            if(left < right){
                T temp = start[left];
                start[left] = start[right];
                start[right] = temp;
                if(right) right--;
                if((left + 1) < len) left++;
            }else break;
        }
        quicksort(start, start+left-1, quicksort);
        quicksort(start+left, end, quicksort);
    };

      auto pick_pivot = [&deep, &quicksort](T* start, T* end,
auto&& quickselect)->T{
        #ifdef PRINT
```

```
            deep++;
            std::cout << NORMAL << DEEP(deep) <<  "Pick Pivot\n";
            #endif
            if( (end - start) < 6){
                quicksort(start, end, quicksort);
                #ifdef PRINT
                     std::cout << NORMAL << DEEP(deep) << SORTED <<
"Sorted array: [ ";
                for(int i = 0; start + i <= end; i++)
                    std::cout<< start[i] << " ";
                 std::cout << "]\n" << NORMAL<< DEEP(deep) << FIND <<
" Pivot = " << start[(end-start+1)/2] << "\n";
                std::cout << NORMAL << DEEP(deep) << "End Pick Pivot\
n";
                deep--;
                #endif
                return start[(end-start+1)/2];
            }
            size_t len = (end - start);
             std::unique_ptr<T[]> medians = std::make_unique<T[]>((len
%5 == 0 ? len/5 : (len/5 + 1)));
            len = len/5;
            #ifdef PRINT
              std::cout << NORMAL << DEEP(deep) << SORTED << "Sorted
groups of 5:";
            #endif
            for(int i = 0; i < len; i++){
                  quicksort(start + i * 5, start + (i + 1) * 5 - 1,
quicksort);
                medians[i] = start[i * 5 + 2];
                #ifdef PRINT
                std::cout << " [ ";
                for(int j = 0; j < 5; j++)
                    std::cout << start[i * 5 + j] << " ";
                std::cout << "]";
                #endif
            }
            if( (len * 5) < (end - start)){
                quicksort(start + len * 5, end, quicksort);
                 medians[len] = start[(end - start - len * 5)/2 + len
* 5];
                len++;
                #ifdef PRINT
                std::cout << " [ ";
                for(int j = 0; j <= (end - start - (len - 1) * 5); j+
+)
                    std::cout << start[(len - 1) * 5 + j] << " ";
                std::cout << "]";
                #endif
            }
            #ifdef PRINT
              std::cout << "\n" << NORMAL << DEEP(deep) << ORIGIN <<
"Medians: [ ";
            for(int i = 0; i < len; i++)
                std::cout << medians[i] << " ";
            std::cout << "]\n";
```

```
            T ret = quickselect(medians.get(), medians.get()+len-1,
len/2, quickselect);
            std::cout << NORMAL << DEEP(deep) << "End Pick Pivot\n";
            deep--;
            return ret;
            #else
               return quickselect(medians.get(), medians.get()+len-1,
len/2, quickselect);
            #endif
        };

        auto quickselect = [&deep, &quicksort, &pick_pivot](T* start,
T* end, size_t k, auto&& quickselect)->T{
            #ifdef PRINT
            deep++;
             std::cout << NORMAL << DEEP(deep) << "QuickSelect\n" <<
NORMAL << DEEP(deep) << FIND << "Finding element on " << k+1 << "
position\n"<< NORMAL << DEEP(deep) << ORIGIN << "Origin array: ";
            for(int i = 0; start + i <= end; i++)
                std::cout << start[i] << " ";
            std::cout<< "\n";
            #endif
            if((end - start) < 6){
                quicksort(start, end, quicksort);
                #ifdef PRINT
                     std::cout << NORMAL << DEEP(deep) << SORTED <<
"Sorted array: ";
                for(int i = 0; start + i <= end; i++)
                    std::cout << start[i] << " ";
                std::cout<< "\n";
                 std::cout << NORMAL << DEEP(deep) << FIND << "Find: "
<< start[k] << "\n" << NORMAL << DEEP(deep) << "End Quick Select\n";
                deep--;
                #endif
                return start[k];
            }
            T pivot = pick_pivot(start, end, quickselect);
            #ifdef PRINT
             std::cout << NORMAL << DEEP(deep) << ORIGIN << "Pivot: "
<< pivot << "\n";
            #endif
            cycle:
            size_t left = 0, right = end - start, len = right+1;
            while(left < right){
                 while((left +1)  < len && start[left] < pivot) left+
+;
                while(right > 0 && start[right] >= pivot) right--;
                if(left <= right){
                    T temp = start[left];
                    start[left] = start[right];
                    start[right] = temp;
                    if(right) right--;
                    if(left + 1 < len) left++;
                }
            }
            while(left > 0 && start[left] > pivot) left--;
```

```
            if(left == 0 && start[left] > pivot){
                pivot = start[left];
                goto cycle;
            }
            #ifdef PRINT
                std::cout << NORMAL << DEEP(deep) << SEPARETED <<
"Separated array: Left: [ ";
            for(int i = 0; i <= left; i++)
                std::cout << start[i] << " ";
            std::cout << "] Right: [ ";
            for(int i = left+1; i < len; i++)
                std::cout << start[i] << " ";
            std::cout<< "]\n";
              T res = (left >= k ? quickselect(start, start+left, k,
quickselect) : quickselect(start+left+1, end, k-left-1, quickselect));
            std::cout << NORMAL << DEEP(deep) << "End QuickSelect\n";
            deep--;
            return res;
            #else
              return (left >= k ? quickselect(start, start+left, k,
quickselect) : quickselect(start+left+1, end, k-left-1, quickselect));
            #endif
        };


        return quickselect(start, end, k, quickselect);
    }

    int main(){
        size_t count;
        std::cin >> count;
        std::unique_ptr<int[]> arr = std::make_unique<int[]>(count);
        for(int i = 0; i < count; i++)
            std::cin >> arr[i];
        size_t k;
        std::cin >> k;
        try{
            int res = find_k_min(arr.get(), arr.get()+count-1, k-1);
            std::cout << "K min = " << res << "\n";
        }catch(std::exception& e){
            std::cout << "Error: " << e.what() << "\n";
            return 1;
        }
        return 0;
    }
```

Название файла: Source/check_sort.cpp

```
#include<iostream>
#include<algorithm>
#include<memory>

int main(){
    size_t count;
    std::cin >> count;
```

```
    std::unique_ptr<int[]> arr = std::make_unique<int[]>(count);
    for(int i = 0; i < count; i++)
        std::cin >> arr[i];
    size_t k;
    std::cin >> k;
    std::sort(arr.get(), arr.get()+count);
    std::cout <<  "Sorted array: ";
    for(int i = 0; i < count; i++)
        std::cout << arr[i] << " ";
    std::cout << "\n";
    std::cout << "K min = " << arr[k-1] << "\n";
    return 0;
}
```

Название файла: Makefile

```
LAB = lab4

.PHONY: all clean

all: run_tests

$(LAB): Source/$(LAB).cpp
	g++ $< -g -std=c++17 -o $@

check: Source/check_sort.cpp
	g++ $< -g -std=c++17 -o $@

run_tests: $(LAB) check
	python3 test.py

clean:
	rm -rf $(LAB) check
```

Название файла: test.py

```
import unittest
import subprocess
import os
import filecmp
import random

class TestParamAnalyzer(unittest.TestCase):

    cwd = os.getcwd()
    test_dir = './Tests/'
    tests = []

    @classmethod
    def setUpClass(self):
        print('Start Testing...')

    def start_test(self):
        out = 'output.test'
```

```
            check_out = 'check_output.test'
            in_file = 'input.test'
            input_str = ''
            for i in range(self.size):
                input_str += str(random.randint(0,1000000000)) + ' '
            print('List size:', self.size)
            print("List for search: [ ", input_str, ']', sep='')
            input_str = str(self.size) + '\n' + input_str
            for i in range(1, self.size+1):
                print("Search element on position:", i)
                in_str = input_str + '\n' + str(i) + '\n\n'
                with open(out, 'w') as f_out:
                     p = subprocess.run(['./lab4', ], cwd = self.cwd,
stdout = f_out, text = True, input = in_str)
                with open(check_out, 'w') as f_out:
                    p = subprocess.run(['./check', ], cwd = self.cwd,
stdout = f_out, text = True, input = in_str)
                with open(out, 'r') as f_out:
                    str_out = f_out.read()
                    print('Output:', str_out, sep='\n')
                    str_out = str_out[str_out.rfind('K min =')]
                with open(check_out, 'r') as f_out:
                    str_check = f_out.read()
                    print('Check output:', str_check, sep='\n')
                        str_check = str_check[str_check.rfind('K min
=')]
                self.assertTrue(str_out == str_check)

        def test_0(self):
            out = 'output.test'
            in_str = '3\n42 666 13\n6\n\n'
            with open(out, 'w') as f_out:
                   p = subprocess.run(['./lab4', ], cwd = self.cwd,
stdout = f_out, text = True, input = in_str)
            with open(out, 'r') as f_out:
                output = f_out.read()
            self.assertTrue(output == 'Error: K > size\n')

        def test_1(self):
            self.size = 25
            self.start_test()

        def test_2(self):
            self.size = 73
            self.start_test()

        def test_3(self):
            self.size = 549
            self.start_test()

        #@unittest.skip('')
        def test_4(self):
            self.size = 1091
            self.start_test()

        def tearDown(self):
```

```
        files = []
        files.extend([f for f in os.listdir('./')
                      if f.endswith('.test')])
        for f in files:
            if os.path.isfile(f):
                os.remove(f)


if __name__ == "__main__":
    unittest.main()
```