Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет «Высшая школа экономики»

ОТЧЕТ

к лабораторной работе №1 «Sorting algorithms»

по дисциплине «Методы программирования»

Выполнила студентка группы СКБ222

Маркова Варвара Владимировна

Москва 2025

Вариант 12

Массив данных о командах, принимающих участие в чемпионате страны по футболу: страна, название клуба, город, год, ФИО главного тренера (сравнение по полям – год, страна количество набранных очков (по убыванию), название клуба)

г) Шейкер-сортировка

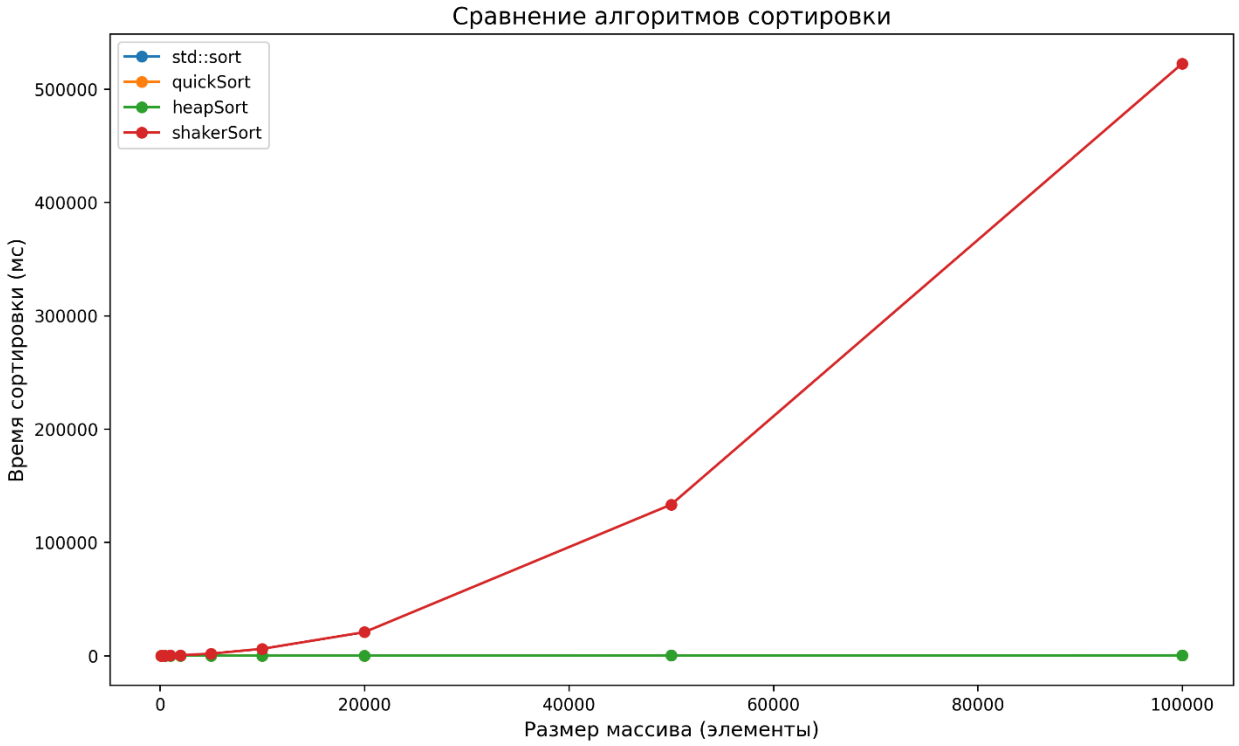
д) Пирамидальная сортировка

е) Быстрая сортировка

Ссылка на исходный код программы в репозитории: https://github.com/varvara0411/pm1_lab/tree/laba1

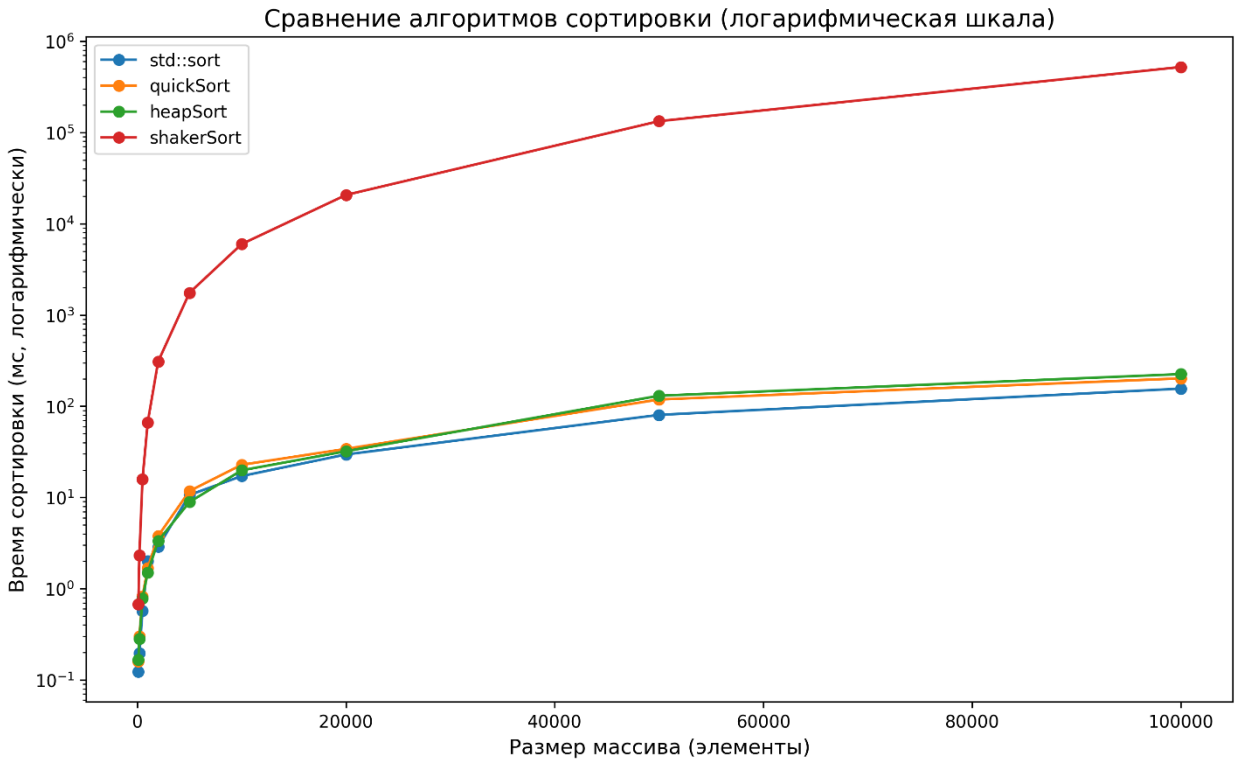
Графики времени сортировок

Сравним время работы 4 алгоритмов сортировки:



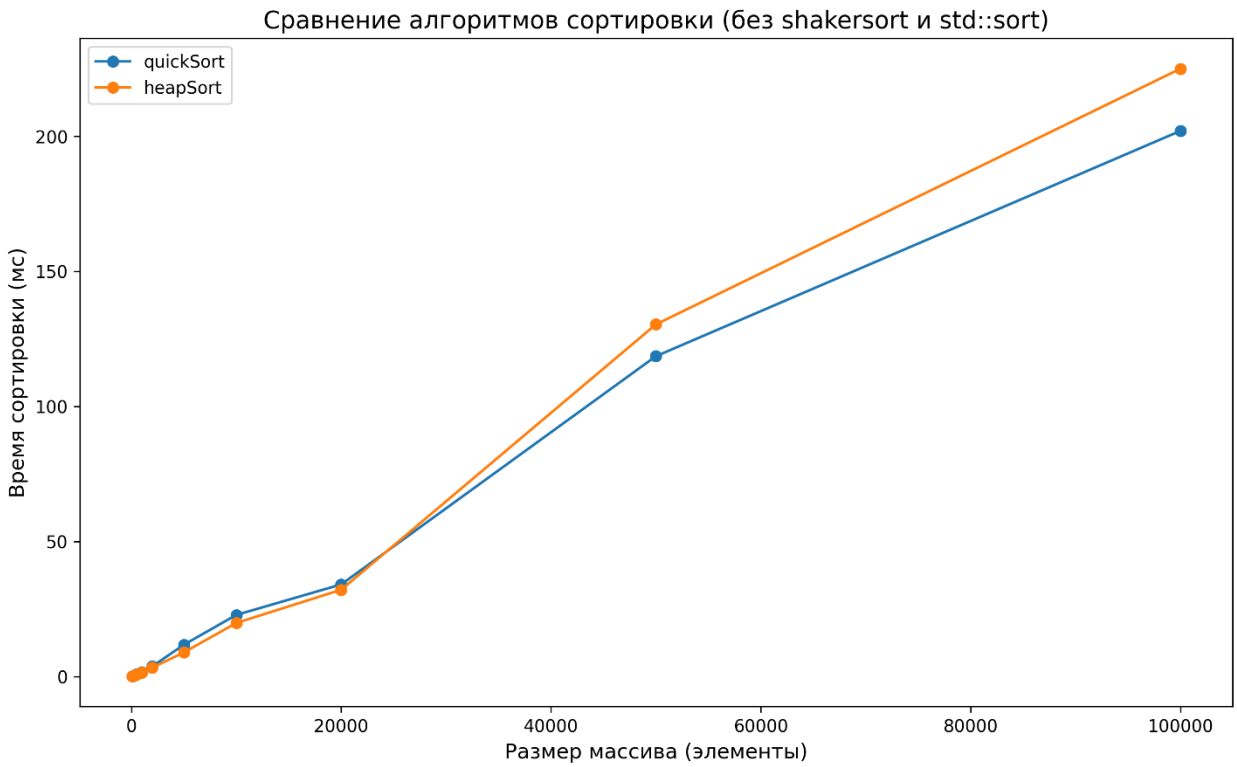
Очевидно, что самый медленно работающий алгоритм – это *шейкер-сортировка*, поскольку его сложность составляет $O(n^2)$, где n – число элементов в массиве. Для более наглядного сравнения трех оставшихся

алгоритмов построим аналогичный график с логарифмической шкалой значений времени работы алгоритмов сортировок.



Видим, что, *пирамидальная сортировка* и *быстрая сортировка* довольно близки по значениям (что видно также и из файла times.csv с результатами измерений времени работы алгоритмов), а *std::sort* представляет собой наиболее быстрый алгоритм.

Теперь построим график без шейкер-сортировки и *std::sort*, чтоб сравнить *пирамидальную сортировку* и *быструю сортировку*.



Заметим, что *быстрая сортировка* начинает понемногу выигрывать во времени у *пирамидальной сортировки* при увеличении размера массива.

Таким образом, наиболее предпочтительным вариантом в плане скорости выполнения сортировок является использование *std::sort*, а наименее предпочтительным – использование *шейкер-сортировки*. *Быстрая сортировка* и *пирамидальная сортировка*, в целом, дают близкие значения по времени, но с увеличением размера массива, *быстрая сортировка* начинает давать небольшой выигрыш во времени, хотя оба алгоритма имеют сложность O(n log(n)), где n – число элементов в массиве.

Lab 1

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 Football Class Reference

The Football class contains information about country, club, city, year, trainer and points of football team.

`#include <football.h>`

Public Member Functions

Constructors and Destructor

Default constructor.

- Football ()
  Default constructor of the Football class.
- Football (std::string country_new, std::string club_new, std::string city_new, int year_new, std::string trainer_new, int points_new)
  Parameterized constructor.
- Football (const Football &other)
  Copy constructor.
- ~Football ()=default
  Default destructor.

Overloading Comparison Operators and Assignment Operator

Comparison based on key fields (country, club, year, points).

- bool operator> (const Football &other) const
  Overloading operator '>' based on key fields.
- bool operator< (const Football &other) const
  Overloading operator '<' based on key fields.
- bool operator<= (const Football &other) const
  Overloading operator '<=' based on key fields.
- bool operator>= (const Football &other) const
  Overloading operator '>=' based on key fields.
- bool operator== (const Football &other) const
  Overloading operator '==' based on key fields.
- Football & operator= (const Football &other)
  Overloading the assignment operator based on key fields.

Friends

Overloading the output operator

Declaring a friend function for overloading

- std::ostream & operator<< (std::ostream &out, const Football &football)
  Overloading the output operator '<<'.

3.1.1 Constructor & Destructor Documentation

3.1.1.1 Football() [1/2]

```
Football::Football (
            std::string country_new,
            std::string club_new,
            std::string city_new,
            int year_new,
            std::string trainer_new,
            int points_new )
```

Parameterized constructor of the Football class.

3.1.1.2 Football() [2/2]

```
Football::Football (
            const Football & other )
```

Copy constructor of the Football class.

The documentation for this class was generated from the following files:

- football.h
- football.cpp

Chapter 4

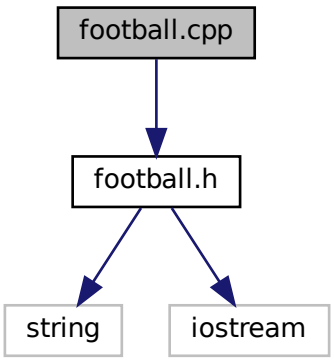
File Documentation

4.1 football.cpp File Reference

Implementation of the Football class methods and operator overloading.

```
#include "football.h"
```

Include dependency graph for football.cpp:



Functions

- std::ostream & operator<< (std::ostream &out, const Football &football)

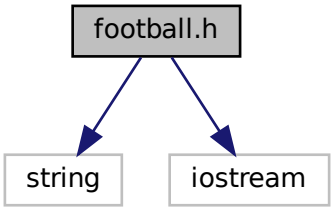
  Overloading the output operator '<<'.

4.2 football.h File Reference

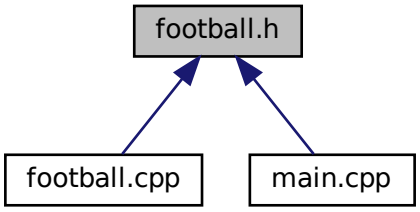
Declaration of the Football class.

```
#include <string>
#include <iostream>
```

Include dependency graph for football.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Football

  The Football class contains information about country, club, city, year, trainer and points of football team.

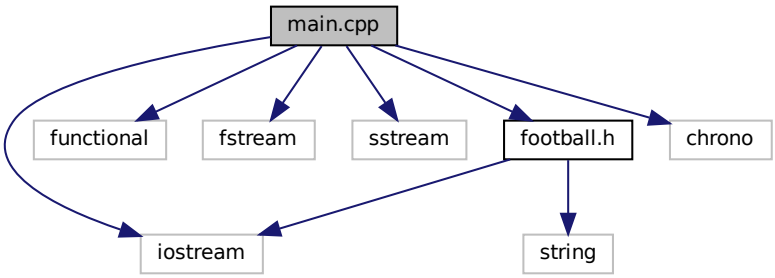
4.3 main.cpp File Reference

Reading datasets, sorting them, recording the time of sorting and saving the results.

```
#include <iostream>
#include <functional>
#include <fstream>
```

```
#include <sstream>
#include "football.h"
#include <chrono>
```

Include dependency graph for main.cpp:



Functions

- template<class T >
  void shakerSort (T a[ ], long size)

  Descending Shaker Sort algorithm.
- template<class T >
  void quickSort (T *a, long N)

  Descending Quick Sort algorithm.
- template<class T >
  void downHeap (T a[ ], long k, long n)

  Helper function for Heap Sort algorithm.
- template<class T >
  void heapSort (T a[ ], long size)

  Descending Heap Sort algorithm.
- template<class T >
  void sort_new (T *arr, long size)

  std::sort algorithm.
- void readCSV (std::string filename, int N, Football *result)

  Reads football data from a CSV file into an array of Football objects.
- int main ()

  Entry point of the program.

4.3.1 Detailed Description

Iterates over multiple CSV files containing arrays of different sizes, as defined in the `sizes` array. For each dataset, it loads the data into an array of `Football` objects using `readCSV`, performs sorting, measures the execution time. Also it saves the sorted arrays into files within the `sorted_datasets` directory.

4.3.2 Function Documentation

4.3.2.1 downHeap()

```
template<class T >
void downHeap (
             T a[],
             long k,
             long n )
```

Template Parameters

T	The type of elements in the array.

Parameters

a	The name of the array.
k	The starting node for the heapifying process.
n	Number of elements.

4.3.2.2 heapSort()

```
template<class T >
void heapSort (
             T a[],
             long size )
```

Template Parameters

T	The type of elements in the array.

Parameters

a	The name of the array.
size	Number of elements.

4.3.2.3 main()

```
int main ( )
```

Iterates through datasets, applies different sorting algorithms to each dataset, measures their execution time, and saves results.

4.3.2.4 quickSort()

```
template<class T >
void quickSort (
             T * a,
             long N )
```

Template Parameters

T	The type of elements in the array.

Parameters

a	The name of the array.
N	Number of elements.

4.3.2.5 readCSV()

```
void readCSV (
             std::string filename,
             int N,
             Football * result )
```

Parameters

filename	The path to the CSV file to read.
N	The number of rows (records) to read from the file.
result	Pointer to the array where the read Football objects will be stored.

4.3.2.6 shakerSort()

```
template<class T >
void shakerSort (
             T a[],
             long size )
```

Template Parameters

T	The type of elements in the array.

Parameters

a	The name of the array.
size	Number of elements.

4.3.2.7 sort_new()

```
template<class T >
void sort_new (
            T * arr,
            long size )
```

Template Parameters

T	The type of elements in the array.

Parameters

arr	The name of the array.
size	Number of elements.

4.4 plt.py File Reference

Parses timing data and plots performance of sorting algorithms.

Variables

- **plt.data** = pd.read_csv('times.csv')
- **plt.figsize**
- **plt.marker**
- **plt.label**
- **plt.fontsize**
- **plt.dpi**
- **plt.bbox_inches**

Index