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

Студентка гр. 9382 _____ Круглова В.Д.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Разобраться в понятии и применении сортировки данных. Составить оценку её достоинств и недостатков.

Задание.

Вариант 6.

6. Бинго-сортировка.

Описание основных функций.

void printDepth(int depth)

Назначение: Выводит глубину работы алгоритма.

Описание аргументов: Глубина погружения сортировки.

template<typename type> void printVector(std::vector<type> *vec)

Назначение: Выводит переданный вектор.

Описание аргументов: Указатель на вектор, хранящий элементы произвольного типа.

template<typename type> void bingoSort(std::vector<type> *vec)

Назначение: Производит сортировку элементов вектора.

Описание аргументов: Указатель на вектор, хранящий элементы произвольного типа.

Возвращаемое значение: Отсортированный вектор с элементами произвольного типа.

Описание алгоритма.

После считывания вектора запускается алгоритм сортировки, состоящий из проверки на единственный элемент в векторе, которая мгновенно завершает процесс сортировки, и отлова и помещения в конец найденного максимума с последующей обработкой, игнорирующей данный элемент.

Выглядит это так:

1. Первым делом заводим переменные, отвечающие за номер и значение максимального элемента и инициализируем их индексом последнего элемента и его значением соответственно.
2. В цикле проходим по элементам с конца и сравниваем их с названным максимумом, по ходу присваивая соответствующей переменной большие значения элементов вектора. Пока индекс последнего элемента не ноль и последний элемент является максимумом, уменьшаем индекс на 1.
3. Пока индекс последнего необработанного элемента не ноль, сохраняем максимальное значение в новую переменную, с помощью которой в цикле сравниваем с ней каждый элемент, чтобы найти и положить в конец максимальный элемент, при этом уменьшив индекс последнего необработанного элемента. Параллельно среди остальных элементов ищется новый максимум и снова осуществляется проверка на максимальные элементы итак стоящие в конце.

Такой вид сортировки больше всего подходит для работы с не уникальными элементами, ввиду предусмотренных выше проверок.

Здесь фокус в том, что в неупорядоченной части запоминается не только максимальный элемент, но и определяется максимум для следующей итерации. Это позволяет при повторяющихся максимумах не искать их заново каждый раз, а ставить на своё место сразу как только этот максимум в очередной раз встретили в массиве.

Алгоритмическая сложность осталась та же. Но если массив состоит из повторяющихся чисел, то бинго-сортировка справится в десятки раз быстрее, чем обычная сортировка выбором.

Пример работы программы.

Таблица 1 – Пример работы

Входные данные	Выходные данные
5 1 2 3 4 5	Enter how many symbols will be entered, OR enter 'q' for exit: Enter all your elements, OR enter 'q' for exit: Start sorting. Index of the last unsorted element = 3 Max value for the next iteration = 5

	[] Index of the last unsorted element = 2 [] Max value for the next iteration = 4 [] Array after iteration: [] 1 2 3 4 5 [][] Index of the last unsorted element = 1 [][] Max value for the next iteration = 3 [][] Array after iteration: [][] 1 2 3 4 5 [][][] Index of the last unsorted element = 0 [][][] Max value for the next iteration = 2 [][][] Array after iteration: [][][] 1 2 3 4 5 End of sorting. Array before sorting: 1 2 3 4 5 Array after sorting: 1 2 3 4 5

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 — Результаты тестирования

No	Входные данные	Выходные данные	Комментарии
1	1 1	1	Простейший случай
2	180	Sorry, count of elements can`t be more than 30. Please, try again.	Некорректный ввод
3	-8	Sorry, count of elements can`t be <= 0. Please, try again.	Некорректный ввод
4	4 2 2 2 2	2 2 2 2	Граничные данные
5	3 3 2 1	1 2 3	Ввод через enter
6	4 27 6 12 45	6 12 27 45	Просто ввод
7	q	'q' was entered. Finishing program...	Выход пользователем
8	f	Input is invalid. Please, try again.	Некорректный ввод
9	5	1 2 2 3 4	Граничные

	2 3 1 2 4		данные
10	3 1 2 q	Element №2 is invalid. Please, try again.	Некорректный ввод

Выводы.

Получены знания в области работы с сортировкой Бинго. Написана работающая программа на языке C++, способная сортировать элементы, поданные на вход. Оценена оптимальность работы с этим видом совтировки.

ПРИЛОЖЕНИЕ С КОДОМ

main.cpp :

```
#include "headers.h"

void printDepth(int depth) // выведет "  " указанное кол-во раз
{
    for (int j = 0; j < depth; j++)
        std::cout << "[]";
    return;
}

int main()
{
    using namespace std;

    vector<int> inputElements, sortedVector;

    { // считывание количества элементов и самих элементов

    int count_of_elements = 0;
    int element = 0;
    char ch;
    while(true) // считывание, сколько будет введено элементов
    {
        cout << "Enter how many symbols will be entered, OR enter 'q' for exit:"
<< endl;

        ch = cin.peek();   // записываем следующий знак из cin в переменную, не
изменяя cin
        if (ch == 'q')     // проверка на ввод 'q'
        {
            cout << "\'q\' was entered. Finishing program..." << endl;
            return 0;      // выход из программы
        }

        cin >> count_of_elements; // пытаемся считать значение в int

        if (cin.fail()) // если это не 'q' и не число или же это слишком большое
число
        {
            cin.clear();            // переводим cin в обычный режим работы
            cin.ignore(32767,'\n'); // очищаем cin
            cout << "Input is invalid.  Please, try again.\n" << endl;
        }
        else
        {
            std::cin.ignore(32767,'\n'); // очищаем cin

            if (count_of_elements > 30) // проверяем значение на адекватные
размеры
                cout << "Sorry, count of elements can`t be more than 30. Please,
try again.\n" << endl;
            else if (count_of_elements <= 0)
                cout << "Sorry, count of elements can`t be <= 0. Please, try
again.\n" << endl;
            else // если все хорошо
                break;
        }
    }
```

```
    for (int i = 0; i < count_of_elements; ) // считывание самих элементов
    {
        if (i == 0)
            cout << "Enter all your elements, OR enter 'q' for exit:" << endl;

        ch = cin.peek();   // записываем следующий знак из cin в переменную, не
изменяя cin
        if (ch == 'q')     // проверка на ввод 'q'
        {
            cout << "\'q\' was entered. Finishing program..." << endl;
            return 0;      // выход из программы
        }

        cin >> element;    // пытаемся получить число

        if (cin.fail())    // ошибка при получении числа
        {
            cin.clear();
            cin.ignore(32767,'\n');

            cout << "Element №" << i << " is invalid.  Please, try again.\n" <<
endl;
            if (i > 0)
                cout << "Enter remaining elements, OR enter 'q' for exit:" <<
endl;
            continue;
        }
        else // записываем элементы в вектор
        {
            inputElements.push_back(element);
            i += 1;
        }
    }

    cout << endl;
    }

    sortedVector = inputElements;

    bingoSort(&sortedVector);

    cout << "\nArray before sorting:" << endl;
    printVector(&inputElements);

    cout << "\nArray after sorting:" << endl;
    printVector(&sortedVector);

    return 0;
}
```

headers.h :

```
#include <iostream>
#include <vector>

void printDepth(int depth);

template<typename type> void printVector(std::vector<type> *vec)
{
    using namespace std;

    for (int i = 0; i < vec->size(); i++)
```

```
        cout << vec->at(i) << ' ';
    cout << endl;
}

template<typename type> void bingoSort(std::vector<type> *vec)
{
    using namespace std;
    cout << "Start sorting." << endl;
    if(vec->size() == 1)
    {
        cout << "There is only one element.\nStop sorting." << endl;
        return;
    }

    // начало первого прохода (здесь мы ищем только nextMaxValue)
    int depth = 0;
    int indOfMax = vec->size() - 1; // просто записали индекс последнего
элемента
    // indOfMax хранит индекс, куда записывается максимальный элемент (конец
неотсортированной последовательности)
    int nextMaxValue = vec->at(indOfMax); // и инициализировали максимум для
следующей итерации

    for (int i = indOfMax; i >= 0; i-- ) // идем от конца списка до начала
    {
        if(vec->at(i) > nextMaxValue) // если нынешнее значение > сохраненного
max
        {
            nextMaxValue = vec->at(i); // записываем его в max
        }
    }

    while (indOfMax && vec->at(indOfMax) == nextMaxValue) // пока индекс не 0, и
на этом месте и так лежит максимальное значение
    {
        indOfMax -= 1; // уменьшаем индекс на 1
        // если в конце и так лежат максимальные значения, то нам незачем по ним
ходить ещё раз на следующей итерации
    }
    // конец первого прохода
    // на этот момент у нас есть индекс с которого мы начнем идти и максимальное
значение, с которым будем сравнивать др. элементы

    { // промежуточная информация
    printDepth(depth);
    cout << " Index of the last unsorted element = " << indOfMax << endl;
    printDepth(depth);
    cout << " Max value for the next iteration = " << nextMaxValue << endl;
    depth += 1;
    }

    // последующие проходы
    while(indOfMax) // пока индекс конца неотсортированного массива != 0
    {
        int maxValue = nextMaxValue; // на прошлой итерации уже было найдено max
значение для этой итерации
        nextMaxValue = vec->at(indOfMax); // присвоили значение последнего
неотсортированного элемента

        for (int i = indOfMax; i >= 0; i--) // идем от конца неотсортированного
массива до его начала
        {
            if(vec->at(i) == maxValue) // если значение элемента ==
максимальному на этой итерации
```

```
            {
                type sw = vec->at(i); // то меняем местами этот элемент с
последним неотсортированным
                vec->at(i) = vec->at(indOfMax);
                vec->at(indOfMax) = sw;

                indOfMax -= 1; // и уменьшаем индекс правой границы
неотсортированной последовательности
            }
            else if(vec->at(i) > nextMaxValue) // если этот элемент больше
максимального значения для следующей итерации
            {
                nextMaxValue = vec->at(i); // то сохраняем его значение
            };
        };
        while (indOfMax && vec->at(indOfMax) == nextMaxValue) // пока индекс не
0, и на его позиции лежит максимальное значение следующей итерации
        {
            indOfMax -= 1; // уменьшаем индекс на 1
        };

        { // промежуточная информация
            printDepth(depth);
            cout << " Index of the last unsorted element = " << indOfMax <<
endl;

            printDepth(depth);
            cout << " Max value for the next iteration = " << nextMaxValue <<
endl;

            printDepth(depth);
            cout << " Array after iteration: " << endl;

            printDepth(depth);
            cout << ' ';
            printVector(vec);

            depth += 1;
        }

    }
    cout << "End of sorting." << endl;
    return;
}
```