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: Алгоритмы сортировки

Студент гр. 9382 _____ Демин В.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить принципы алгоритмов сортировки массива. Освоить навыки разработки программ, осуществляющих сортировку.

Задание.

Вариант 1s.

Реализовать алгоритм Timsort.

Алгоритм.

1.Формирование подмассивов, вычисление минимального размера подмассива и сортировка каждого подмассива вставками.

2. Добавление размеров подмассивов в стек.

3. Слияние всех подмассивов в один, при этом учитывать что слияение эффективно работает при примерно равных размерах. Поэтому нужен стек, в нем слияния происходит с вершины вниз, при этом сравниваются размеры подмассивов верхних трех элементов стек. В итоге останется два подмассива. Которые остается слить в один.

Функции и структуры данных.

Структура данных стек необходима для алгоритма тимсорт. Элементы стека хранят в себе индекс и размер подмассива.

Функция merge объединяет в один массив два подмассива сортировкой слиянием.

Функции getMinrun – вычисляет оптимальный размер для подмассивов.

Функция subarray – формирует подмассивы и сортирует их вставками.

Функция timsort – основная функция сортировки которая выполняет все необходимые шаги алгоритма.

Выводы.

Был изучен алгоритм timsort. Получены навыки разработки программы, реализующей этот алгоритм.

Тестирование.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	9513 2540 9264 3193 7091 1399 2768 167 1594 5625 2514 2293 7525 9567 3363 5963 6838 332 823 6078 1412 5105 6813 79 8025 6462 2052 8097 2409 9671 4462 4160 1028 554 708 2897 8670 1728 1787 708 6029 1171 4421 3985 2162 1727 9178 7330 6444 4875 9657 9917 4441 3978 1066 9558 6639 333 4694 9378 9066 1723 4889 9902 1814 2484 6294 2065 656 6130 555 559 5608 369 2567 896 2106 1 2698 2352 3709 1194 5068 8 762 1743 6625 3934 8577 7413 8154 1768 8297 8342 4303 614 6791 689 6508 2961	1 8 79 167 332 333 369 554 555 559 614 656 689 708 708 762 823 896 1028 1066 1171 1194 1399 1412 1594 1723 1727 1728 1743 1768 1787 1814 2052 2065 2106 2162 2293 2352 2409 2484 2514 2540 2540 2567 2698 2768 2897 3193 3363 3709 3934 3978 3985 4160 4303 4421 4441 4462 4694 4889 5068 5105 5608 5625 5963 6029 6078 6130 6294 6444 6462 6508 6625 6639 6791 6813 6838 7091 7330 7413 7525 8025 8097 8154 8297 8342 8577 8670 9066 9178 9264 9378 9513 9558 9567 9657 9671 9902 9917 9917	
2.	60 67 12 28 67 76 6 91 31 81 20 35	Array after sorting: 6 12 20 28 31 35 60 67 67 76 81 91	
3.	22 78 151 179 36 79 107 109 139 40 10 72 151 65 93 106 144 25 112 172 66 42 146 16 58 176 194 34 109 39 48 106 194 126 43 8 139 127 159 192 151	0 0 1 4 5 6 7 8 8 8 9 9 10 10 13 13 16 16 17 17 18 19 22 22 22 23 23 24 24 25 25 25 27 27 27 27 27 28 28 28 29 32 33 34 34 35 35 36 36 37 37 37 38 39 40 42 42 43 43 45 45	

	153 106 129 60 63 102 145 28 58 25 35 69 4 35 115 110 7 144 9 23 78 128 61 120 1 27 186 108 55 184 141 79 183 151 160 5 48 158 162 61 90 129 114 23 164 126 28 146 88 24 50 73 139 58 111 22 153 190 80 104 17 54 68 177 60 19 78 171 132 76 32 116 27 138 161 55 89 105 178 13 91 113 110 28 136 22 25 159 141 37 131 94 27 9 108 95 95 164 138 181 163 113 80 96 156 50 106 141 116 165 150 160 168 45 13 103 148 46 27 8 45 128 122 126 135 192 190 8 17 164 69 139 86 189 133 97 77 18 82 169 139 34 16 193 87 138 126 194 150 152 132 51 57 175 6 24 92 85 97 157 0 81 142 197 75 168 197 144 142 146 68 36 191 80 103 50 0 33 90 37 56 70 149 191 185 10 47 29 42 102 38 140 37 164 134 116 43 178 176 155 183 193 109 117 168 135 100 27 139	46 47 48 48 50 50 50 51 54 55 55 56 57 58 58 58 60 60 61 61 63 65 66 68 68 69 69 70 72 73 75 76 77 78 78 78 79 79 80 80 80 81 82 85 86 87 88 89 90 90 91 92 93 94 95 95 96 97 97 100 102 102 103 103 104 105 106 106 106 106 107 108 108 109 109 109 110 110 111 112 113 113 114 115 116 116 116 117 120 120 122 126 126 126 126 127 128 129 129 131 132 132 133 134 135 135 138 138 138 139 139 139 139 139 140 141 141 141 142 142 144 144 144 145 146 146 146 148 149 150 150 151 151 151 151 152 152 153 153 155 156 157 158 159 159 159 160 160 161 162 163 164 164 164 164 165 168 168 168 169 171 172 175 176 176 177 178 178 179 179 181 183 183 184 185 186 189 190 190 191 191 192 192 193 193 194 194 197 197	

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
//
// Created by vikto on 17.11.2020.
//

#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <fstream>

#define NMax 10000

using namespace std;

class logger {
    ofstream file;

public:
    logger() {
        file.open("intermeiate.txt");
        if (!file.is_open())
            throw runtime_error("file open failure");
    }

//закрытие файла
    ~logger() {
        std::cout << "\nIntermeiate values write to file \"intermeiate.txt\"";
        file.close();
    }

    void startTimsort() {
        file << "Start sorting\n";
        file << "Array before sorting:";
    }
```

```
        void endTimsort() {
            file << "End sorting\n";
            file << "Array after sorting:";



        }

        void printArray(int *array, int count) {
            for (int i = 0; i < count; ++i) {
                file << array[i] << " ";
            }
            file << "\n";
            file                                                    <<
"_____________________________________________________________\n";
        }

        void startSortByInsertsForSmallCount() {
            file                                                    <<
"_____________________________________________________________\n";
            file << "N < 64, timsort isn't optimal\n";
            file << "Start simple sort by inserts\n";
        }

        void endSortByInsertsForSmallCount() {
            file << "Sorting end\n";
            file                                                    <<
"_____________________________________________________________\n";
        }

        void startSortByInsertsForSubarray(int index) {
            file << "Sort by insert subarray with index:" << index <<
"\n";
        }

        void setMinrun(int i) {
            this->minrun = i;
        }
```

```
        void mergeBefore(int *array, int countX, int countY, int
indexX, int indexY) {
            file << "Merging two subarrays\n";
            file << "BeforeMerge:\n";
            file << "First:";
            for (int i = 0; i < countX; ++i) {
                file << array[indexX + i] << " ";
            }
            file << "\n";
            file << "Second:";
            for (int i = 0; i < countY; ++i) {
                file << array[indexY + i] << " ";
            }
            file << "\n";
        }

        void mergeAfter(int *array, int count, int index) {
            file << "AfterMerge:\n";
            for (int i = 0; i < count; ++i) {
                file << array[index + i] << " ";
            }
            file << "\n";
        }

        int minrun;
    };


    struct subarray {
        int indexStart; //индекс начала подмасивов
        int countMassiv; //размер подмасссива
    };
    struct stack {
        struct subarray *subarrayes;
        int countSubarray = 0; // кол-во подмассивов
    };
```

```
    void push(struct stack *sub, int indexstart, int countmassiv)
{//Добавление в стек
        if (sub->countSubarray == 0) {
            sub->subarrayes = new struct subarray[1000];
        }
        sub->subarrayes[sub->countSubarray].indexStart = indexstart;
        sub->subarrayes[sub->countSubarray].countMassiv = countmassiv;
        sub->countSubarray++;
    }

    void pop(struct stack *sub) {//удаление вершины
        sub->countSubarray--;
    }

    bool checkXYZ(struct stack *sub) {// проверка на слияние
подмассивов
        if ((sub->subarrayes[sub->countSubarray - 1].countMassiv + sub-
>subarrayes[sub->countSubarray - 2].countMassiv <
             sub->subarrayes[sub->countSubarray - 3].countMassiv &&
             sub->subarrayes[sub->countSubarray - 1].countMassiv < sub-
>subarrayes[sub->countSubarray - 2].countMassiv)) {
            return true; //нужно выполнять слвияние текущего с
предыдущим
        }
        return false;
    }

    void merge(int *left, int *right, int sizeLeft, int sizeRight, bool
typeLeft, bool typeRight) // ф-ия слияния
    {
        int *mLeft = new int[sizeLeft + 1];
        int *mRight = new int[sizeRight + 1];
        for (int i = 0; i < sizeLeft; ++i) {
            mLeft[i] = left[i];
        }
        for (int i = 0; i < sizeRight; ++i) {
            mRight[i] = right[i];
        }
```

```
int i = 0;
int j = 0;
int k = 0;
if (typeLeft && typeRight) { //оба возрастают

    while (i < sizeLeft && j < sizeRight) {
        if (mLeft[i] <= mRight[j]) {
            left[k] = mLeft[i];
            i++;
        } else {
            left[k] = mRight[j];
            j++;
        }
        k++;
    }

    // копирование левый
    while (i < sizeLeft) {
        left[k] = mLeft[i];
        k++;
        i++;
    }


    while (j < sizeRight) {
        left[k] = mRight[j];
        k++;
        j++;
    }

} else if (typeLeft && !typeRight) {//левый возрастает
    while (i < sizeLeft && j < sizeRight) {
        if (mLeft[i] <= mRight[sizeRight - 1 - j]) {
            left[k] = mLeft[i];
            i++;
        } else {
            left[k] = mRight[sizeRight - 1 - j];
            j++;
```

```
            }
            k++;
        }


        while (i < sizeLeft) {
            left[k] = mLeft[i];
            k++;
            i++;
        }


        while (j < sizeRight) {
            left[k] = mRight[sizeRight - 1 - j];
            k++;
            j++;
        }

    } else if (!typeLeft && typeRight) {//правый возрастает
        while (i < sizeLeft && j < sizeRight) {
            if (mLeft[sizeLeft - 1 - i] <= mRight[j]) {
                left[k] = mLeft[sizeLeft - 1 - i];
                i++;
            } else {
                left[k] = mRight[j];
                j++;
            }
            k++;
        }


        while (i < sizeLeft) {
            left[k] = mLeft[sizeLeft - 1 - i];
            k++;
            i++;
        }
```

```
            while (j < sizeRight) {
                left[k] = mRight[j];
                k++;
                j++;
            }
        } else if (!typeLeft && !typeRight) {//никто не  возрастает
            while (i < sizeLeft && j < sizeRight) {
                if (mLeft[sizeLeft - 1 - i] <= mRight[sizeRight - 1 -
j]) {
                    left[k] = mLeft[sizeLeft - 1 - i];
                    i++;
                } else {
                    left[k] = mRight[sizeRight - 1 - j];
                    j++;
                }
                k++;
            }


            while (i < sizeLeft) {
                left[k] = mLeft[sizeLeft - 1 - i];
                k++;
                i++;
            }


            while (j < sizeRight) {
                left[k] = mRight[sizeRight - 1 - j];
                k++;
                j++;
            }
        }
    }

    //вычисление оптимального размера для разбиения на подмассивы
    int getMinrun(
            int n) //   берутся старшие 6 бит из N и добавляется
единица, если в оставшихся младших битах есть хотя бы один ненулевой
```

```
{
    int r = 0;
    while (n >= 64) {
        r |= n & 1;
        n >>= 1;
    }
    return n + r;
}

void subarray(int *array, int N, int minrun, int *position)
//создани подмассивов и их сортировка вставками
{
    if (N - *position < minrun) {
        minrun = abs(N - *position);
    }

    int *curr;//конкретный подмассив
    curr = array + *position;
    int *run = new int[minrun];//временный подмассив

    int runCount = 2;//количество элементов в подмассиые
    run[0] = *curr;
    run[1] = *(curr + 1);

    int j = 1;
    int i = 1;

    bool order = run[0] <= run[1];

    for (i = 1; (i < N - 1) && (j != minrun); i++)//ищем
упорядоченный подмассив
    {
        if (order) {
            if (curr[i] <= curr[i + 1]) {
                run[i + 1] = curr[i + 1];
                ++runCount;
            } else break;
        } else {
```

```
                if (curr[i] >= curr[i + 1]) {
                    run[i + 1] = curr[i + 1];
                    ++runCount;
                } else break;
            }
        }
        if (runCount <
            minrun) //Если размер текущего run'а меньше, чем minrun — выбираются следующие за найденным run-ом элементы.
        {
            while (runCount < minrun) {
                run[runCount] = curr[i];
                ++i;
                ++runCount;
            }
        }
        if (!order)  // сорт по уб.
        {
            int j = 0;
            for (int i = 1; i < minrun; i++) {
                int value = run[i];
                j = i - 1;

                while (j >= 0 && run[j] < value) {
                    run[j + 1] = run[j];
                    run[j] = value;
                    j = j - 1;

                }
            }
        } else {
            for (int i = 1; i < minrun; i++) {
                int value = run[i];
                int j = i - 1;

                while (j >= 0 && run[j] > value) {
                    run[j + 1] = run[j];
                    run[j] = value;
```

```
                    j = j - 1;
                }
            }
        }

        for (int i = 0, j = *position; i < runCount; i++, j++) {
            array[j] = run[i];
            ++(*position);
        }
    }

    void timsort(int *arr, int count) {
        logger *log = new logger();
        log->startTimsort();
        log->printArray(arr, count);
        int minrun;
        if (count < 64) //   простая сортировка вставкой . тк. не
оптимален timsort
        {
            log->startSortByInsertsForSmallCount();
            int minrun = count;
            int j = 0;
            for (int i = 1; i < minrun; i++) {
                log->printArray(arr, count);
                int value = arr[i];
                j = i - 1;

                while (j >= 0 && arr[j] > value) {
                    arr[j + 1] = arr[j];
                    arr[j] = value;
                    j = j - 1;
                }

            }
        } else {
            minrun = getMinrun(count);
            log->setMinrun(minrun);
            cout << "minrun= " << minrun << endl;
```

```
            int position = 0;
            int lastRun = count % minrun;//размер последнего подмассива
            int countRun = count / minrun;//размер подмассивов

            while (position < count) // разделение на подмассивы и их
сортировка вставками
            {
                log->startSortByInsertsForSubarray(position / minrun);
                if (count - position < minrun) {
                    log->printArray(arr + position, lastRun);
                    subarray(arr, count, minrun, &position);
                    log->printArray(arr + position - lastRun, lastRun);
                } else {
                    log->printArray(arr + position, minrun);
                    subarray(arr, count, minrun, &position);
                    log->printArray(arr + position - minrun, minrun);
                }
            }


            struct stack sub;

            for (int i = 0;
                 i < countRun + 1; i++) //заносится пара данных в стек
, индекс начала подмассивов и количество эл-ов в нем.
            {

                if ((i == countRun && lastRun != 0)) {
                    push(&sub, i * minrun, lastRun);
                    break;
                } else if (i == countRun && lastRun == 0) {
                    break;
                }
                push(&sub, i * minrun, minrun);
            }
            //сортировка слиянием
            int index_start_X;
            int index_start_Y;
```

```
            int index_start_Z;
            int count_X;
            int count_Y;
            int count_Z;
            while (sub.countSubarray >= 3) // пока больше 3-ех
происходит сияние ранов
            {

                index_start_X = sub.subarrayes[sub.countSubarray -
1].indexStart;
                index_start_Y = sub.subarrayes[sub.countSubarray -
2].indexStart;
                index_start_Z = sub.subarrayes[sub.countSubarray -
3].indexStart;
                count_X = sub.subarrayes[sub.countSubarray -
1].countMassiv;
                count_Y = sub.subarrayes[sub.countSubarray -
2].countMassiv;
                count_Z = sub.subarrayes[sub.countSubarray -
3].countMassiv;

                if (checkXYZ(&sub)) // проверка на слияния
                {
                    log->mergeBefore(arr, count_X, count_Y,
index_start_X, index_start_Y);
                    merge(&arr[index_start_Y], &arr[index_start_X],
count_Y, count_X,
                          (arr[index_start_Y] < arr[index_start_Y +
count_Y - 1]),
                          (arr[index_start_X] < arr[index_start_X +
count_X - 1]));
                    sub.subarrayes[sub.countSubarray - 2].countMassiv
+= count_X;
                    log->mergeAfter(arr, count_Y + count_X,
index_start_Y);
                    pop(&sub);
                } else {
                    if (count_X <= count_Z) {
```

```
                        log->mergeBefore(arr,     count_X,     count_Y,
index_start_X, index_start_Y);
                        merge(&arr[index_start_Y],  &arr[index_start_X],
count_Y, count_X,
                              (arr[index_start_Y] < arr[index_start_Y +
count_Y - 1]),
                              (arr[index_start_X] < arr[index_start_X +
count_X - 1]));

                        sub.subarrayes[sub.countSubarray              -
2].countMassiv += count_X;
                        log->mergeAfter(arr,    count_Y    +    count_X,
index_start_Y);
                        pop(&sub);
                    } else {
                        log->mergeBefore(arr,     count_Z,     count_Y,
index_start_Z, index_start_Y);
                        merge(&arr[index_start_Z],  &arr[index_start_Y],
count_Z, count_Y,
                              (arr[index_start_Z] < arr[index_start_Z +
count_Z - 1]),
                              (arr[index_start_Y] < arr[index_start_Y +
count_Y - 1]));

                        sub.subarrayes[sub.countSubarray              -
3].countMassiv += count_Y;
                        log->mergeAfter(arr,    count_Z    +    count_Y,
index_start_Z);
                        pop(&sub);
                        sub.subarrayes[sub.countSubarray              -
1].indexStart = index_start_X;
                        sub.subarrayes[sub.countSubarray              -
1].countMassiv = count_X;
                    }
                }
            }
            // оставшеся два
```

```
            index_start_X    =    sub.subarrayes[sub.countSubarray    -
1].indexStart;
            index_start_Y    =    sub.subarrayes[sub.countSubarray    -
2].indexStart;
            count_X      =      sub.subarrayes[sub.countSubarray      -
1].countMassiv;
            count_Y      =      sub.subarrayes[sub.countSubarray      -
2].countMassiv;
            log->mergeBefore(arr,   count_X,   count_Y,   index_start_X,
index_start_Y);
            merge(&arr[index_start_Y],   &arr[index_start_X],   count_Y,
count_X,
                  (arr[index_start_Y]  <  arr[index_start_Y  +  count_Y  -
1]),
                  (arr[index_start_X]  <  arr[index_start_X  +  count_X  -
1]));
            log->mergeAfter(arr, count_Y + count_X, index_start_Y);
        }
        log->endTimsort();
        log->printArray(arr, count);
        delete log;
    }

    void enter(int *array, int *count) {
        int type = 0;//Тип ввода
        cout << "Enter from file -1\n";
        cout << "Enter from console -2\n";
        cin >> type;
        ifstream file;
        string name;//Название файла
        int temp;
        switch (type) {
            case 1:
                cout << "Enter file name\n";
                cin >> name;
                file.open(name);
                if (file.is_open()) {
                    temp = 0;
```

```
                    while (!file.eof()) {//считывание элементов из
файла
                        file >> array[temp];
                        temp++;
                    }
                    *count = temp;
                } else {
                    cout << "File isn't open." << endl;
                }
                break;
            case 2:
                cout << "Enter count elements:\n";
                cin >> *count;
                cout << "Enter elements: \n";
                for (int i = 0; i < *count; ++i) {//считывание
элементов из строки
                    cin >> array[i];
                }
                break;
            default:
                break;
        }
    }

    void print(int *array, int count) {
        int type = 0;
        cout << "\nPrint to console -1\n";
        cout << "Print to file -2\n";
        cin >> type;
        ofstream file;
        string name_f;
        switch (type) {
            case 1: {
                cout << "\n\nArray after sorting:\n";
                for (int i = 0; i < count; i++) {
                    cout << array[i] << ' ';
                }
            }
```

```
            break;
        case 2:
            cout << "Enter file name\n";
            cin >> name_f;
            file.open(name_f);
            if (file.is_open()) {
                file << "\n\nArray after sorting:\n";
                for (int i = 0; i < count; i++) {
                    file << array[i] << ' ';
                }
            } else file << "Unable to open file";
            break;
        default:
            break;
    }


}

int main() {

    int *array = new int[NMax];//Рассматриваемый массив
    int count = 0;//Количество элементов в массиве

    enter(array, &count);//Ввод массива


    timsort(array, count);//Сортировка

    print(array, count);//Вывод массива
}
```