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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Сортировка N-арной кучей

Студентка гр. 9303 _____ Хафаева Н.Л.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студентка Хафаева Н.Л.

Группа 9303

Тема работы: сортировка N-арной кучей

Исходные данные: произвольный набор символов

Предполагаемый объем пояснительной записки:
Не менее 10 страниц.

Дата выдачи задания: 6.11.2020

Дата сдачи реферата: 25.12.2020

Дата защиты реферата: 25.12.2020

Студентка _____ Хафаева Н.Л.

Преподаватель _____ Филатов Ар.Ю.

АННОТАЦИЯ

В данной работе проводилось исследование сортировки n-арной кучи и сравнение просеивания сверху-вниз и снизу-вверх. Была экспериментально установлена сложность алгоритмов на основе подсчета базовых операций. Была реализована генерация входных данных для худшего и среднего случаев.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цели исследования:

-Реализовать алгоритмы сортировки n-арной кучей просеиванием сверху-вниз и снизу-вверх.

-Провести сравнительный анализ сортировки с просеиванием сверху-вниз и снизу-вверх.

Исследование проводилось со случайно сгенерированными наборами данных.

План экспериментального исследования:

1. Реализовать алгоритм сортировки n-арной кучей просеиванием сверху-вниз и снизу-вверх.
2. Реализовать функции для генерации набора данных для среднего и худшего случая.
3. Выбрать базовые операции и реализовать подсчет данных операций.
4. Сделать выводы по полученным данным, оценить сложность алгоритмов, сопоставить с теоретической оценкой.

1. ОПИСАНИЕ СОРТИРОВКИ N-АРНОЙ КУЧЕЙ

1.1. Общие сведения

Куча — это специализированная структура данных типа дерево, которая удовлетворяет свойству кучи: если B является потомком A, то A ≥ B. Из этого следует, что элемент с наибольшим ключом всегда является корневым узлом кучи, поэтому иногда такие кучи называют max-кучами (в качестве альтернативы, если сравнение перевернуть, то наименьший элемент будет всегда корневым узлом, такие кучи называют min-кучами).

1.2. Описание алгоритма

Основной частью сортировки кучей является просейка. Просеивание для элемента состоит в том, что если он меньше по размеру чем потомки, объединённых в неразрывную цепочку, то этот элемент нужно переместить как можно ниже, а бóльших потомков по ветке поднять наверх на 1 уровень.

Алгоритм:

Этап 1. Формируем из всего массива сортирующее дерево. Для этого проходим справа-налево элементы (от последних к первым) и если у элемента есть потомки, то для него делаем просейку.

Этап 2. Максимумы ставим в конец неотсортированной части массива. Так как данные в массиве после первого этапа представляют из себя сортирующее дерево, максимальный элемент находится на первом месте в массиве. Первый элемент (он же максимум) меняем с последним элементом неотсортированной части массива местами. После этого обмена максимум оказался своём окончательном месте, т.е. максимальный элемент отсортирован. Неотсортированная часть массива перестала быть сортирующим деревом, но это исправляется однократной просейкой — в результате чего на первом месте массива оказывается предыдущий по величине максимальный элемент. Действия этого этапа снова повторяются для оставшейся неупорядоченной области, до тех пор, пока максимумы поочерёдно не будут перемещены на свои окончательные позиции.

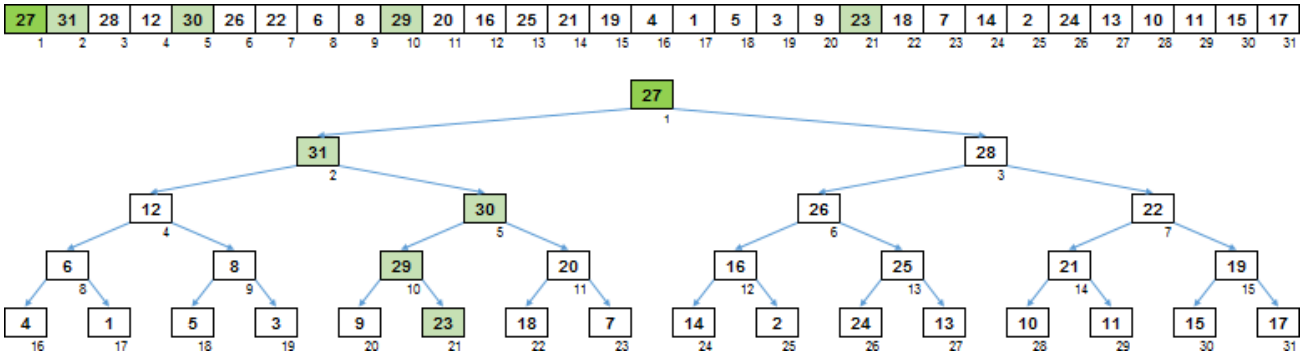
1.3. Восходящая просейка

Стандартная просейка в классической сортировке кучей работает так: элемент из корня поддерева отправляется в буфер обмена, элементы из ветки по результатам сравнения поднимаются наверх. Всё просто, но получается слишком много сравнений.

В восходящей просейке сравнения экономятся за счёт того, что родители почти не сравниваются с потомками, в основном, только потомки сравниваются друг с другом. В обычной heapsort и родитель сравнивается с потомками и потомки сравниваются друг с другом — поэтому сравнений получается почти в полтора раза больше при том же количестве обменов.

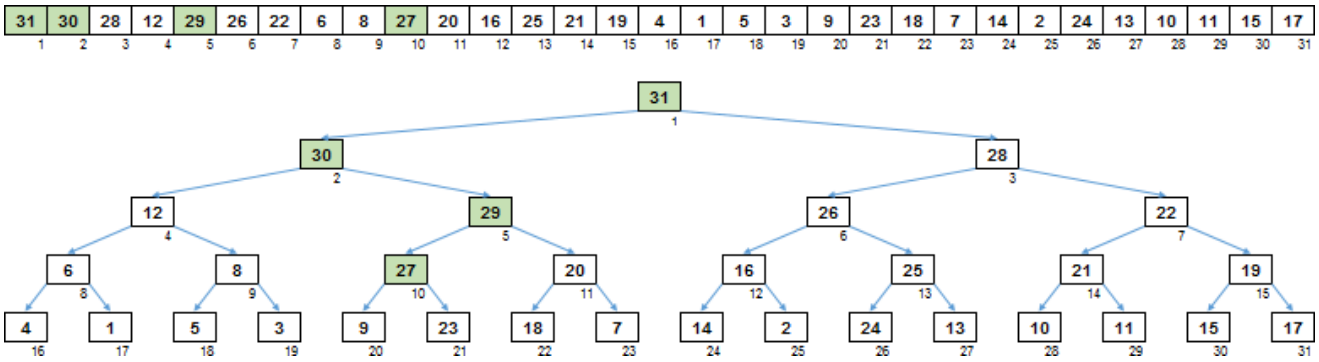
Допустим, у нас массив, в котором уже почти сформирована куча — осталось только просеять корень. Для всех остальных узлов выполнено условие — любой потомок не больше своего родителя.

Прежде всего, от того узла, для которого совершается просейка нужно спуститься вниз, по большим потомкам. Спускаемся в ту ветку, где потомок крупнее. На этом этапе и происходит основное количество сравнений — потомки сравниваются друг с другом.



Достигнув листа на последнем уровне, мы тем самым определились с той веткой, значения в которой нужно сдвинуть вверх. Но сдвинуть нужно не всю ветку, а только ту часть, которая крупнее, чем корень с которого начали.

Поэтому поднимаемся по ветке вверх до ближайшего узла, который больше, чем корень. Последний шаг — используя буферную переменную, сдвигаем значения узлов вверх по ветке.



Восходящая просейка сформировала из массива сортирующее дерево, в котором любой родитель больше, чем его потомки.

2. СРАВНИТЕЛЬНЫЙ АНАЛИЗ СОРТИРОВКИ С ПРОСЕИВАНИЕМ СВЕРХУ-ВНИЗ И СНИЗУ-ВВЕРХ

2.1. Теоретическая оценка сложности алгоритмов

Сложности по времени зависит от просейки.

Однократная просейка обходится в O(log n). Сначала мы для n элементов делаем просейку, чтобы из массива построить первоначальную кучу — этот этап занимает O(n log n). На втором этапе мы при вынесении n текущих максимумов из кучи делаем однократную просейку для оставшейся неотсортированной части, т.е. этот этап также стоит нам O(n log n).

Итоговая сложность по времени: O(n log n) + O(n log n) = O(n log n).

Сортировка снизу вверх - это вариант, который значительно сокращает количество требуемых сравнений. В то время как обычная heapsort требует 2 n log 2 n + O (n) сравнений в худшем случае и в среднем, восходящий вариант требует 1,5 n log 2 n + O (n) сравнений в среднем и в худшем случае. Это достигается за счет улучшения siftDown процедуры. Это изменение несколько улучшает фазу построения кучи в линейном времени, но более существенно во второй фазе. В обычной сортировке кучей каждый шаг просеивания вниз требует двух сравнений, чтобы найти минимум три элемента: новый узел и два его дочерних элемента. Сортировка снизу вверх вместо этого находит путь от самых больших потомков к конечному уровню, используя только одно сравнение на уровне. Другими словами, он находит лист, обладающий тем свойством, что он и все его предки больше или равны своим братьям и сестрам. Затем из этого листа, он ищет вверх (используя одно сравнения за уровень) для правильной позиции в этом пути, чтобы вставить в [конце] . Это то же место, что и при обычном поиске методом heapsort, и для выполнения вставки требуется такое же количество обменов, но для поиска этого местоположения требуется меньше сравнений.

2.2. Практическая оценка сложности алгоритмов

Модно предположить, что худшим случаем для просейки сверху-вниз будет, когда на вход будет подаваться уже отсортированный массив. Потому что будут совершаться бессмысленные телодвижения: просейка сначала будет ставить максимум на первое место в массиве, а потом будет отправлять максимум в конец. Но, если посмотреть на график на рис.1, можно сделать вывод, что у данного алгоритма нет лучшего и худшего случаев, так как графики совпадают.

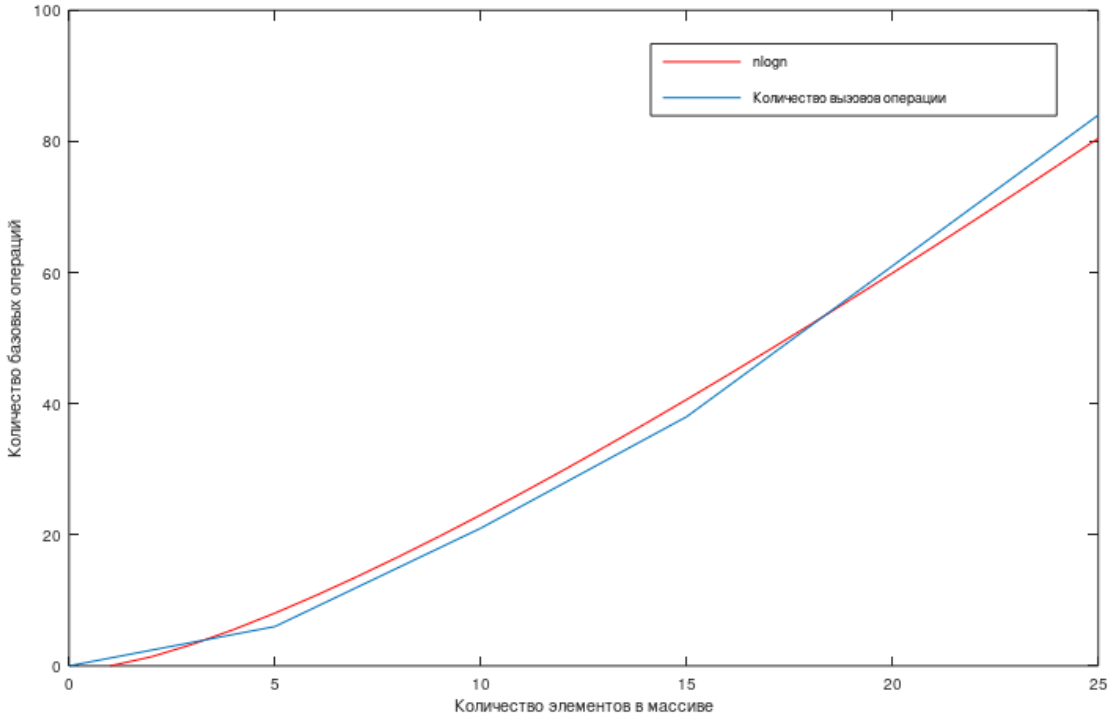


Рисунок 1 - График для отсортированного массива для просейки сверху-вниз

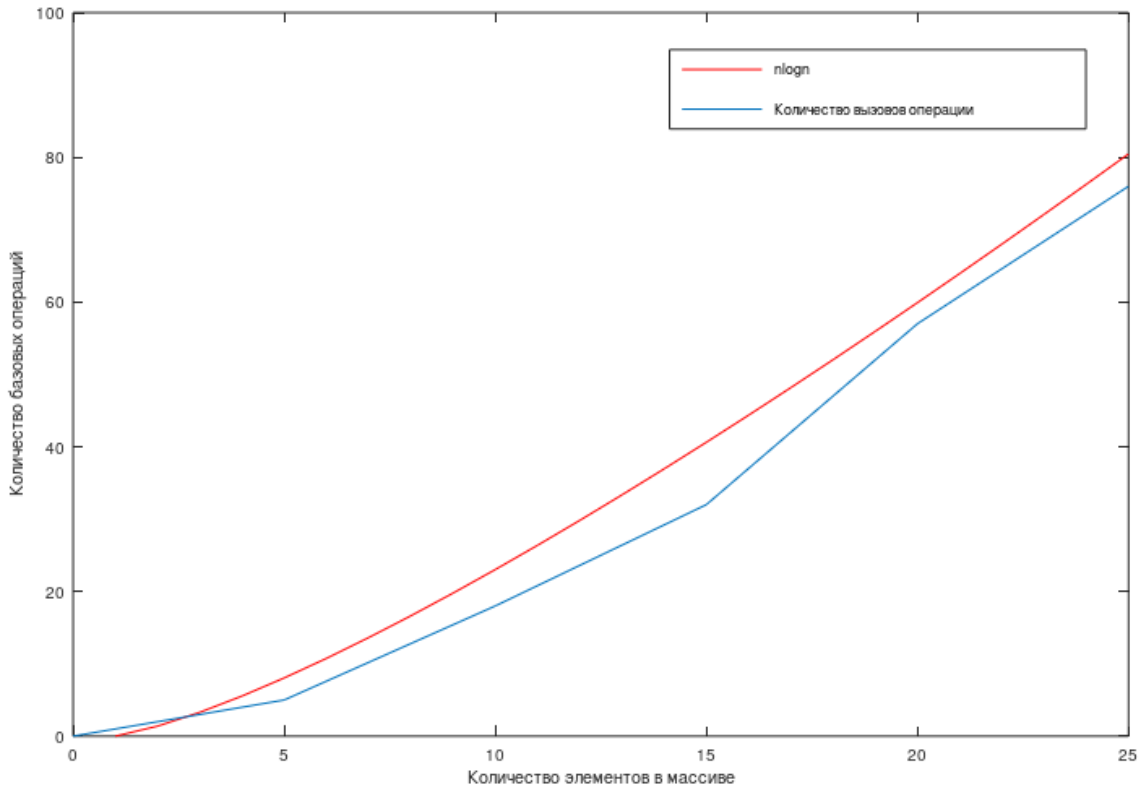


Рисунок 2 - График для массива с рандомными значениями для просейки сверху-вниз

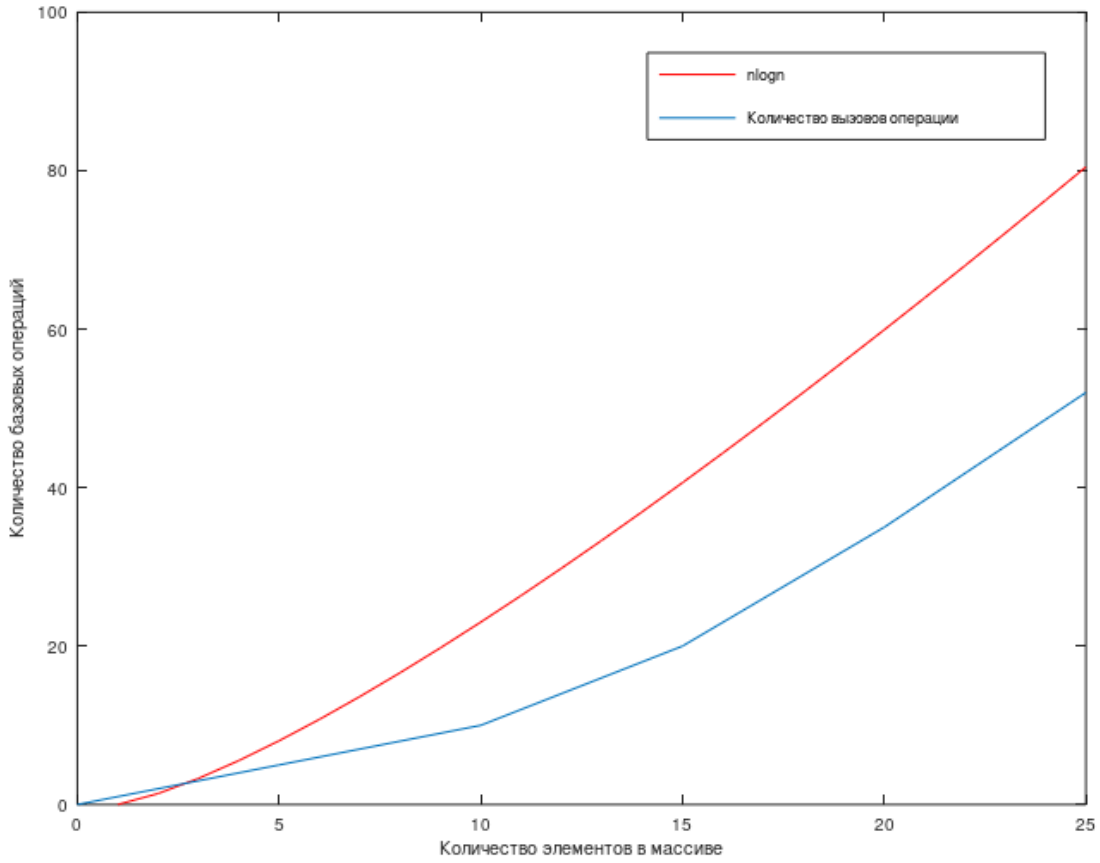


Рисунок 3 - График для просейки снизу-вверх

Можно сделать вывод, что алгоритм с просейкой снизу-вверх работает в 1.5 раза быстрее, чем с просейкой сверху-вниз.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы было проведено сравнительное исследование алгоритма сортировки кучей просейкой снизу-вверх и сверху-вниз. Была написана программа, сортирующая массив n-арной кучей, а также генерация случайных наборов данных. На основе результатов подсчета базовых операций экспериментально оценены сложности данных алгоритмов, которые совпали с теоретическими оценками.

По полученным результатам можно сделать вывод о том, что алгоритм сортировки кучей с просейкой снизу-вверх работает быстрее в 1,5 раза, чем алгоритм сортировки кучей с просейкой сверху-вниз.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

long long CNT = 0;

void sift_down(int* arr, int n, int start, int end) {
	int root = start;

	while (true) {
		int child = root * n + 1;
		if (child > end) {
			break;
		}

		int max = child;
		for (int i = 2; i < n + 1; i++) {
			int current = root * n + i;
			if (current > end) {
				break;
			}
			if (arr[current] > arr[max]) {
				max = current;
			}
		}
		if (arr[root] < arr[max]) {
			int cur = arr[root];
			arr[root] = arr[max];
			arr[max] = cur;
			root = max;
			CNT++;
		}
		else {
			break;
		}
	}
}

int find_max(int* arr,int cur, int n, int len) {
	if (cur * n + 1 >= len) {
		return cur;
	}
```

```
	int max = cur * n + 1;

	for (int i = 2; i <= n; i++) {
		if (cur * n + i > len) {
			break;
		}
		if (arr[cur * n + i] > arr[max]) {
			max = cur * n + i;
		}
	}
	return find_max(arr, max, n, len);
}

void sift_up(int* arr, int root, int n, int len) {
	if (root * n + 1 >= len) {
		return;
	}

	int cur = find_max(arr, root, n,len);

	while (arr[cur] < arr[root]) {
		cur = (cur - 1) / n;
	}
	int tmp = arr[cur];

	arr[cur] = arr[root];

	cur = (cur - 1) / n;

	while (cur > root) {
		int tmp2 = arr[cur];
		arr[cur] = tmp;
		tmp = tmp2;
		cur = (cur - 1) / n;
		CNT++;
	}

	arr[root] = tmp;
}

void sort_down(int* arr, int start, int end, int n) {
	for (int i = end; i >= start; i--) {
		sift_down(arr, n, i, end);
	}

	for (int i = 0; i < end + 1; i++) {
		cout << arr[i] << ' ';
	}
	cout << endl;
```

```
	int temp = end;

	for (int i = 0; i <= end; i++) {
		int cur = arr[0];
		arr[0] = arr[temp];
		arr[temp] = cur;
		for (int i = 0; i < end + 1; i++) {
			cout << arr[i] << ' ';
		}
		cout << endl;
		sift_down(arr, n, 0, temp - 1);
		temp--;
	}
	cout << "result: ";
	for (int i = 0; i < end + 1; i++) {
		cout << arr[i] << ' ';
	}
	cout << endl;

}


int main() {
	int* data;
	int n;
	int size;
	cout << "enter n: " << endl;
	cin >> n;
	cout << "enter array length: " << endl;
	cin >> size;

	data = new int[size];
	char s;
	char c;

	cout << "which case do you want to handle? " << endl;
	cout << "a - average/ w - worst" << endl;
	cin >> s;
	cout << "u - up/ d - down" << endl;
	cin >> c;
	if (s == 'a') {
		for (int i = 0; i < size; i++) {
			data[i] = rand() % 200;
		}
		cout << "input data: ";
		for (int i = 0; i < size; i++) {
			cout << data[i] << ' ';
		}
		cout << endl;
```

```
        if (c == 'd') {
            sort_down(data, 0, size - 1, n);
        }
        else if (c == 'u') {
            for (int i = size - 1; i >= 0; i--) {
                sift_up(data, i, n, size);
            }

            int* ans;
            ans = new int[size];

            for (int i = 0; i < size; i++) {
                ans[size - (i + 1)] = data[0];
                data[0] = data[size - (i + 1)];
                CNT++;
                sift_up(data, 0, n, size - (i + 1));
            }
            cout << "result: ";
            for (int i = 0; i < size; i++) {
                cout << ans[i] << ' ';
            }
            delete[] ans;
        }
        else {
            cout << "incorrect data entered" << endl;
            return 0;
        }
    }
else if (s == 'w') {
        for (int i = 0; i < size; i++) {
            data[i] = i + 1;
        }
        cout << "input data: ";
        for (int i = 0; i < size; i++) {
            cout << data[i] << ' ';
        }
        cout << endl;

        if (c == 'd') {
            sort_down(data, 0, size - 1, n);
        }
        else if (c == 'u') {
            for (int i = size - 1; i >= 0; i--) {
                sift_up(data ,i, n, size);
            }

            int* ans;
            ans = new int[size];

            for (int i = 0; i < size; i++) {
```

```
                    ans[size - (i + 1)] = data[0];
                    data[0] = data[size - (i + 1)];
                    CNT++;
                    sift_up(data, 0,n, size - (i + 1));
               }
               cout << "result: ";
               for (int i = 0; i < size; i++) {
                    cout << ans[i] << ' ';
               }
               delete[] ans;
          }
          else {
               cout << "incorrect data entered" << endl;
               return 0;
          }
     }
     else {
          cout << "incorrect data entered" << endl;
          return 0;
     }

     cout << endl;
     cout << "num " << CNT << endl;

     delete[] data;

     return 0;
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

№	Входные данные	Результат
1	enter n: 3 enter array length: 10 which case do you want to handle? a - average/ w - worst a u - up/ d - down d	input data: 41 67 134 100 169 124 78 158 162 64 169 124 162 100 67 41 78 158 134 64 64 124 162 100 67 41 78 158 134 169 134 124 158 100 67 41 78 64 162 169 64 124 134 100 67 41 78 158 162 169 78 124 64 100 67 41 134 158 162 169 41 78 64 100 67 124 134 158 162 169 67 78 64 41 100 124 134 158 162 169 41 67 64 78 100 124 134 158 162 169 64 41 67 78 100 124 134 158 162 169 41 64 67 78 100 124 134 158 162 169 41 64 67 78 100 124 134 158 162 169 result: 41 64 67 78 100 124 134 158 162 169 num 12 time: 10.255
2	enter n: 3 enter array length: 10 which case do you want to handle? a - average/ w - worst w u - up/ d - down d	input data: 1 2 3 4 5 6 7 8 9 10 10 7 9 4 5 6 2 8 1 3 3 7 9 4 5 6 2 8 1 10 1 7 8 4 5 6 2 3 9 10 1 7 3 4 5 6 2 8 9 10 2 6 3 4 5 1 7 8 9 10 1 5 3 4 2 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 result: 1 2 3 4 5 6 7 8 9 10

		num 17 time: 8.973
3	enter n: 2 enter array length: 10 which case do you want to handle? a - average/ w - worst a u - up/ d - down u	input data: 41 67 134 100 169 124 78 158 162 64 result: 41 64 67 78 100 124 134 158 162 169 num 10
4	enter n: 2 enter array length: 5 which case do you want to handle? a - average/ w - worst a u - up/ d - down u	input data: 41 67 134 100 169 result: 41 67 100 134 169 num 6