ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 18

Дисциплина: Основы алгоритмизации и программирования

Тема: "Вычислительная сложность алгоритмов"

Выполнил: студент группы 211-721
Дерендяев Дмитрий Сергеевич
(Фамилия И.О.)

Дата, подпись 8.12.2021 _____
(Дата) (Подпись)

Проверил: Новичков Иван Константинович _____
(Фамилия И.О., степень, звание) **(Оценка)**

Дата, подпись _____ _____
(Дата) (Подпись)

Замечания:

_____

_____

_____

Москва

2021

Лабораторная работа №18
"Вычислительная сложность алгоритмов"

Цель: Получить практические навыки разработке алгоритмов и их программной реализации.

Справка:

Первоначально понятие алгоритма отождествлялось с понятием метода вычислений. С точки зрения современной практики алгоритм – программа, а критерием алгоритмичности вычислительного процесса является возможность его запрограммировать.

Именно благодаря этой реальности алгоритма, а также благодаря тому, что подход инженера к математическим методам всегда был конструктивным, понятие алгоритма в технике за короткий срок стал необычайно популярным.

Понятие алгоритма, подобно понятиям множества и натурального числа, относится к числу столь фундаментальным понятий, что оно не может быть выражено через другие понятия.

Алгоритм, в конечном счете, выполняется в машинной системе со специфическим набором команд и периферийными устройствами. Для отдельной системы какой-либо алгоритм может быть разработан для полного использования преимуществ данного компьютера и поэтому достигает высокой степени эффективности. Критерий, называемый **системной эффективностью (sys-tem efficiency)**, сравнивает скорость выполнения двух или более алгоритмов, которые разработаны для выполнения одной и той же задачи. Выполняя эти алгоритмы на одном компьютере с одними и теми же наборами данных, мы можем определить относительное время, используя внутренние системные часы. Оценка времени становится мерой системной эффективности для каждого из алгоритмов.

При работе с некоторыми алгоритмами могут стать проблемой ограничения памяти. Процесс может потребовать большого временного хранения, ограничивающего размер первоначального набора данных, или вызвать требующую времени дисковую подкачку. **Эффективность пространства (space efficiency)** — это мера относительного количества внутренней памяти, используемой каким-либо алгоритмом. Она может указать, какого типа компьютер способен выполнять этот алгоритм и полную системную эффективность алгоритма. Вследствие увеличения объема памяти в новых системах, анализ пространственной эффективности становится менее важным.

Третий критерий эффективности рассматривает внутреннюю структуру алгоритма, анализируя его разработку, включая количество тестов сравнения итераций и операторов присваивания, используемых алгоритмом. Эти типы измерений являются независимыми от какой-либо отдельной машинной системы. Критерий измеряет вычислительную сложность алгоритма относительно n, количества элементов данных в коллекции. Мы называем эти критерии **вычислительной эффективностью (computational efficiency)** алгоритма и разрабатываем нотацию **Big-O** для построения измерений, являющихся функциями n.

Сложность алгоритмов обычно оценивают по времени выполнения или по используемой памяти. Алгоритм имеет сложность **O(f(n))**, если при увеличении размера входных данных **n**, время выполнения алгоритма возрастает с той же скоростью, что и функция **f(n)**. Оценивая порядок сложности алгоритма, необходимо использовать только ту часть, которая возрастает быстрее всего.

Задачи:

Необходимо выполнить и оформить описание следующих пунктов:

Необходимо выполнить и оформить описание следующих пунктов:

1. Представить ранее рассмотренные алгоритмы (лабораторные 2-17)
2. Выполнить анализ сложности каждого из алгоритмов.

ПРИМЕЧАНИЕ:

Все алгоритмы, рассмотренные ниже, были изучены в одинаковых условиях:

- Минимальная нагрузка на исполняющий компьютер.
- Использование цикла while в качестве основного цикла.
- Произведение операций обмена с помощью буферной переменной.
- Для вычисления времени выполнения алгоритма была использована библиотека <chrono>, расчет произведен в наносекундах.
- Программы были адаптированы под изучение времени их выполнения:
  - Была использована функция rand() для случайного заполнения массива и изучения различны вариаций состава массива.
  - Была использована структура данных Vector для того, чтобы динамически заполнять массивы и ускорить процесс изучения алгоритмов.

Алгоритм сортировки «Пузырьком»

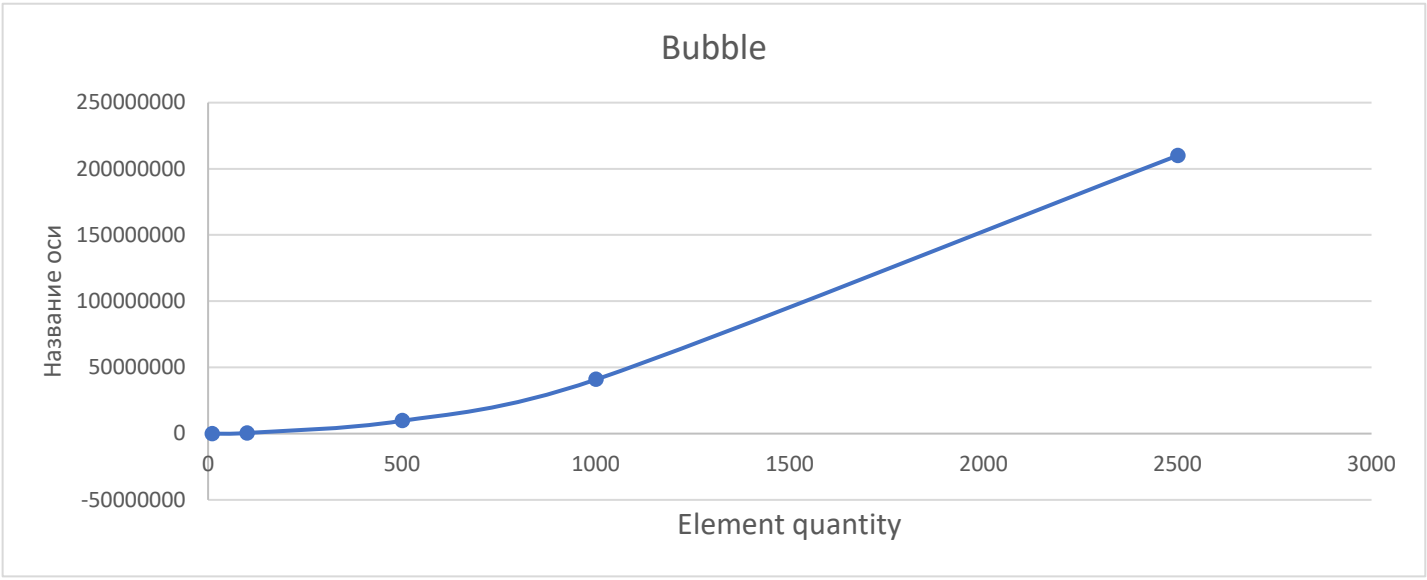
Алгоритм сортировки пузырьком реализован на языке программирования C++(далее листинг программы):

```
#include <iostream>
#include <vector>
#include <ctime>
#include <iomanip>
#include <chrono>

using namespace std;

typedef std::chrono::high_resolution_clock Clock;

int main()
{
vector<int> array2;
int q, peremen;
int i, j, buf1 = 0, buf2 = 0;//îáúÿâëåíèå âñåõ ïåðåìåííûõ, âêëþ÷àÿ ïàðàìåòð ìàññèâîâ, ìàññèâû, áóôåðíûå ÿ÷åéêè ïàìÿòè

clock_t start;
double duration;

cout << "Enter the amount of element:";
cin >> q;
for (int k = 0; k < q; k++)
{
peremen = rand() % q + 1;
array2.push_back(peremen);
}
cout << "Array before sort:" << endl;
for (int i = 0; i < q; i++)
cout << array2[i] << ' ';
cout << endl << endl;

start = clock();

auto t1 = Clock::now();

i = 0;
j = 0;
```

```
while (i < q - 1)
{
while (j < q - 1)
{
if (array2[j] > array2[j + 1])
{
buf2 = array2[j];
array2[j] = array2[j + 1];
array2[j + 1] = buf2;
}
j++;
}
j = 0;
i++;
}
auto t2 = Clock::now();

duration = ((double)clock() - (double)start);
cout << "Array after sort:" << endl;
for (int i = 0; i < q; i++)
cout << array2[i] << ' ';

cout << setprecision(5);
cout << endl << "Duration(miliseconds) is: " << duration << endl;
cout << endl << endl << "Duration(nanoseconds) is: " <<
std::chrono::duration_cast<std::chrono::nanoseconds>(t2 - t1).count() << endl;

return 0;
	}
```

Таблица данных:

Debug version	Algorithm execution time(nanoseconds)			
Element quantity	Bubble			Average
10	4700	4900	4900	4833.333
100	447100	483200	563400	497900
500	9596800	9652600	10088100	9779167
1000	41237100	41932300	39885900	41018433
2500	1.51E+08	2.43E+08	2.37E+08	2.1E+08



Алгоритм сортировки «Расческа»

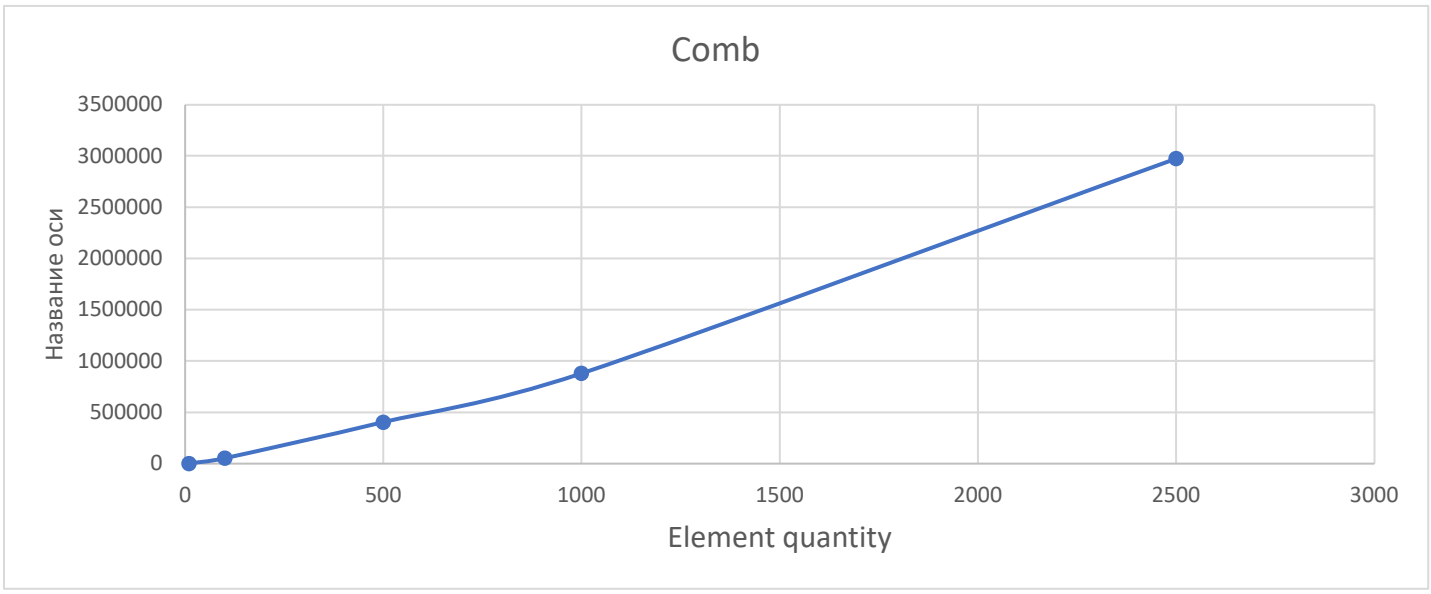
Алгоритм сортировки «Расческой» реализован на языке программирования C++(далее листинг программы):

```
#include <iostream>//îïèñàíèå áèáëèîòåêè
#include <vector>
#include <algorithm>
#include <chrono>

using namespace std;//îáúÿâëåíèå ñòàíäàðòíîãî ïðîñòðàíñòâà èìåí

typedef std::chrono::high_resolution_clock Clock;

vector<int> vec1;//îáúÿâëåíèå ïåðåìåííûõ
int n;
float k = 1.247;
int q, peremen;

int main()//íà÷àëî îñíîâíîé ïðîãðàììû
{

cout << "Enter the amount of element:";

cin >> q;
for (int k = 0; k < q; k++)
{
peremen = rand() % q + 1;
vec1.push_back(peremen);
}

cout << "Array before sort:" << endl;
for (int i = 0; i < q; i++)
cout << vec1[i] << ' ';
cout << endl << endl;

auto t1 = Clock::now();

n = vec1.size();//îïðåäåëåíèå ðàçìåðà ìàññèâà
int step = n - 1;//èíèöèàëèçàöèÿ ïåðâîãî øàãà(êðàéíÿÿ òî÷êà ìàññèâà)
while (step >= 1)//èíèöèàëèçàöèÿ âíåøíåãî öèêëà while äî ìîìåíòà, ïîêà step áîëüøå 1
{
int i = 0;//èíèöèàëèçàöèÿ ñ÷åò÷èêà âíóòðåííåãî öèêëà
```

```
while (i < n - step)//âíóòðåííèé öèêë âûïîëíåò ôóíêöèþ ïðîõîäà ïò íà÷àëà âåêòîðà äî êîíöà(â ïîñëåäíòàèè ðàçìåð ìàññèâà - øàã)

{

if (vec1[i] > vec1[i + step])//åñëè ïåðâûé ýëåìåíò áîëüøå ýëåìåíòà +step ïðîèçâåñòè îáìåí

swap(vec1[i], vec1[i + step]);//âûïîëíåíèå ïåðåìåùåíèÿ ýëåìåíòîâ âåêòîðà ïî èíäåêñàì, çàäàííûì â óñëîâèè

i++;

}

step /= k;//óìåíüøåíèå øàãà

}

auto t2 = Clock::now();

cout << "Array after sort:" << endl;

for (int i = 0; i < q; i++)

cout << vec1[i] << ' ';

cout << endl << endl << "Duration(nanoseconds) is: " <<
std::chrono::duration_cast<std::chrono::nanoseconds>(t2 - t1).count() << endl;

system("pause");

return 0;

}
```

Таблица данных:

Debug version	Algorithm execution time(nanoseconds)			
Element quantity	Comb			Average
10	3100	3000	3600	3233.33
100	61900	46500	55000	54466.7
500	521700	347800	347200	405567
1000	814300	1017300	813800	881800
2500	2382700	3576700	2966900	2975433



Алгоритм сортировки «Вставками»

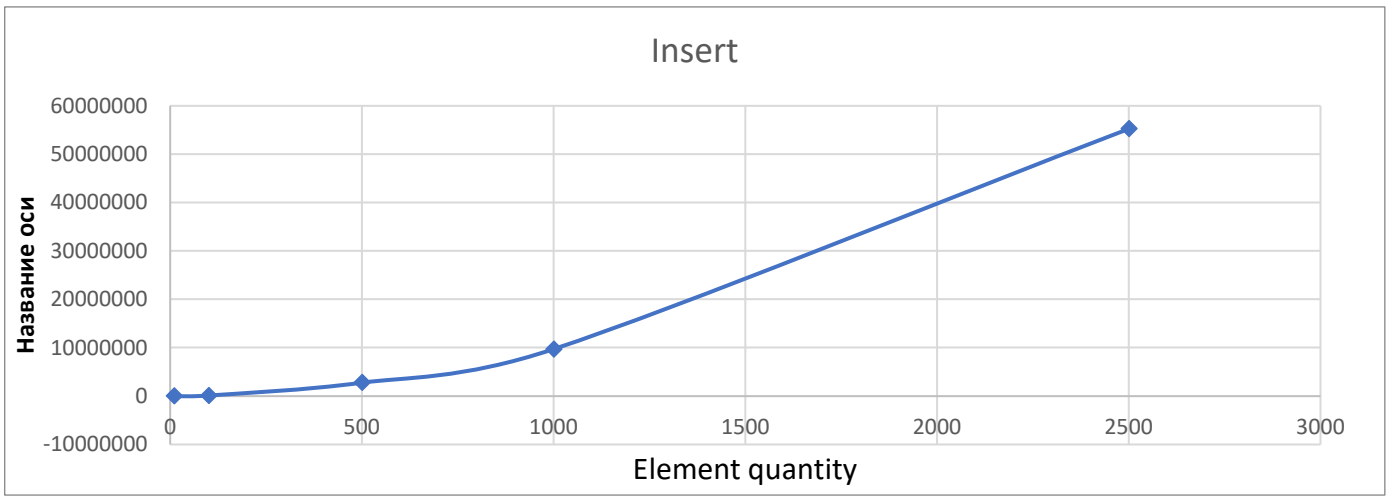
Алгоритм сортировки «Вставками» реализован на языке программирования C++(далее листинг программы):

```
#include <iostream>//подкючение необходимых библиотек
#include <vector>
#include <algorithm>
#include <chrono>

using namespace std;//определение пространства имен

typedef std::chrono::high_resolution_clock Clock;

int el, buf, i, j;//объявление переменных, участвующих в обработке массива
vector<int> vec;//объяление ветора(массива)
int q, peremen;

int main()//начало основной программы
{
setlocale(LC_ALL, "Russian");//установка локали, поодержка Русского языка

cout << "Enter the amount of element:";

cin >> q;
for (int k = 0; k < q; k++)
{
peremen = rand() % q + 1;
vec.push_back(peremen);
}

cout << "Array before sort:" << endl;
for (int i = 0; i < q; i++)
cout << vec[i] << ' ';
cout << endl << endl;

auto t1 = Clock::now();

for (int i = 1; i < vec.size(); i++)//сортировка массива начинается со второго элемента
{
buf = vec[i];//помещаем в буферную ячейку значение числа, которое будем в дальнейшем анализировать
j = i;//внутренний цикл начинается с номера анализируемого элемента
while (j > 0 and buf < vec[j - 1])//цилк выполняется пока элемент не выходит за его пределы
```

```
//и он не больше, чем номер анализируемого элемента j и значение анализируемого элемента меньше, чем предыдущее
{
vec[j] = vec[j - 1];//если условие все-таки выполняется то элементы меняются местами
j--;//уменьшение на единицу номера анализируемого элемента для того, чтобы произвести проверку элемента, расположенного ближе к началу
}
vec[j] = buf;//перчемещение элемента буферной ячейки в ячейку текущего элемента
}
auto t2 = Clock::now();

cout << "Array after sort:" << endl;
for (int i = 0; i < q; i++)
cout << vec[i] << ' ';

cout << endl << endl << "Duration(nanoseconds) is: " <<
std::chrono::duration_cast<std::chrono::nanoseconds>(t2 - t1).count() << endl;

system("pause");

return 0;//завершение программы
}
```

Таблица данных:

Debug version	Algorithm execution time(nanoseconds)			
Element quantity	Inserts			Average
10	2500	2400	2700	2533.33
100	96700	147200	146700	130200
500	2583400	2308900	3496100	2796133
1000	8864800	9237200	1.1E+07	9698767
2500	5.7E+07	5.4E+07	5.5E+07	5.5E+07



Алгоритм сортировки «Шелла»

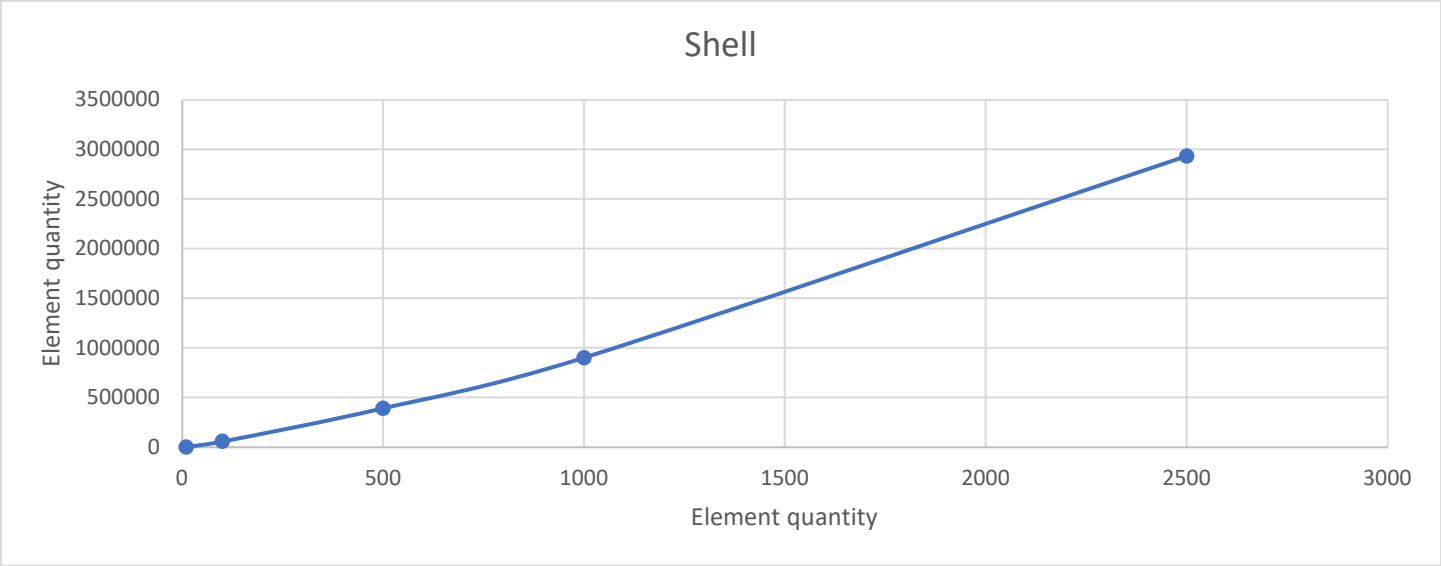
Алгоритм сортировки «Шелла» реализован на языке программирования C++(далее листинг программы):

```
#include <iostream>

#include <vector>

#include <chrono>

using namespace std;

typedef std::chrono::high_resolution_clock Clock;


int main()

{

vector<int> vec;//îáúÿëåíèå âåòîðà(ìàññèâà)

int q, peremen;

int i = 0, j = 0, x = 0, step = 0;//ïîäãîòîâêà ïåðåìåííûõ ê îáðàáîòêå ìàññèâà mas2 ñ ïîìîùüþ öèêëà while

cout << "Enter the amount of element:";

cin >> q;

for (int k = 0; k < q; k++)

{

peremen = rand() % q + 1;

vec.push_back(peremen);

}

cout << "Array before sort:" << endl;

for (int i = 0; i < q; i++)

cout << vec[i] << ' ';

cout << endl << endl;

auto t1 = Clock::now();

step = vec.size() / 2;//âû÷èñëÿåì ðàçìåð øàãà

while (step > 0)//öèêë âûïîëíÿåòñÿ, ïîêà çíà÷åíèå øàãà áîëüøå íóëÿ

{

i = step;// ïåðåõîäèì ê îáðàáîòêå ýëåìåíòà ñ íîìåðîì øàãà

while (i < vec.size())//âûïîëíÿåì öèêë, ïîêà èñïëäóåìûé ýëåìåíò ìåíüøå ðàçìåðà ìàññèâà

{

x = vec[i];//â áóôåðíóþ ïåðåìåííóþ ïîìåùàåì çíà÷åíèå èñïëóäóåìîãî ýëåìåíòà
```

```
j = i;//ïðèñâîåíèå íîìåðó òåêóùåãî ýëåìåíòà íîìåð èñõîäíîãî ýëåìåíòà

while (j >= step)//âûïîëíåíèå öèêëà, ïîêà íîìåð èñõîäíîãî ýëåìåíòà áîëüøå øàãà step

{

if (x < vec[j - step])//ñðàâíèâàåì çíà÷åíèå àíàëèçèðóåìîãî ýëåìåíòà ñ ýëåìåíòîì, îòëè÷àþùèìñÿ íà øàã(ïî èíäåêñó)

{

vec[j] = vec[j - step];////ñòàâèì íà ìåñòî j-ãî ýëåìåíòà ýëåìåíò ñ èíäåêñîì j-øàã

j = j - step;//óìåíüøàåì çíà÷åíèå èíäåêñà j íà âåëè÷èíó øàãà

}

else

break;

}

vec[j] = x;//âîçâðàùàåì çíà÷åíèå áóôåðà

i++;//ïåðåõîäèì ê ñëåäóþùåì ýëåìåíòó

}

step /= 2;//óìåíüøàåì øàã

}

auto t2 = Clock::now();

cout << "Array after sort:" << endl;

for (int i = 0; i < q; i++)

cout << vec[i] << ' ';

cout << endl << endl << "Duration(nanoseconds) is: " <<
std::chrono::duration_cast<std::chrono::nanoseconds>(t2 - t1).count() << endl;

system("pause");

return 0;

}
```

Таблица данных:

Debug version	Algorithm execution time(nanoseconds)			
Element quantity	Shell			Average
10	2900	3500	2900	3100
100	49100	51100	75400	58533.33
500	443300	295000	442000	393433.3
1000	721300	901200	1081900	901466.7
2500	2396900	2694000	3711500	2934133



Алгоритм сортировки «Выбором»

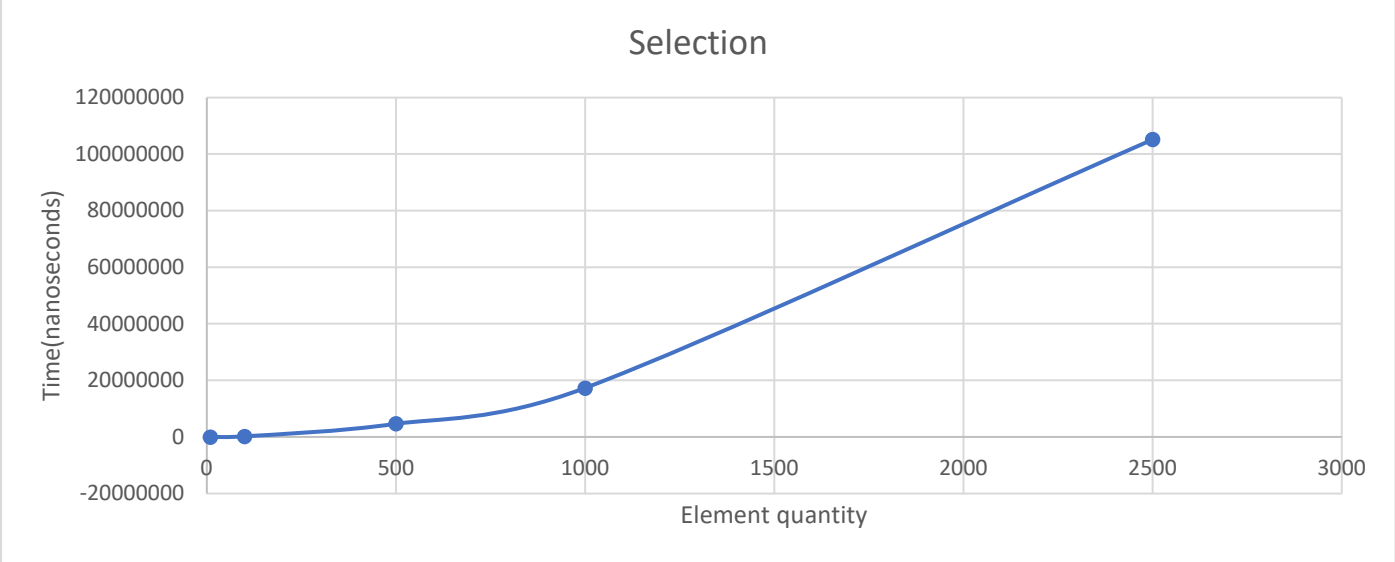
Алгоритм сортировки «Быбором» реализован на языке программирования C++(далее листинг программы):

```
#include <iostream>

#include <vector>

#include <chrono>

using namespace std;

typedef std::chrono::high_resolution_clock Clock;


int main()

{

vector<int> vec;//îáúÿëåíèå âåòîðà(ìàññèâà)

int q, peremen;

int nte, min2, temp, gr = 0;//подготовка необходимых переменных, в том числе размера массива, параметров циклов, буферных переменных и индекса минимального элемента

cout << "Enter the amount of element:";

cin >> q;

for (int k = 0; k < q; k++)

{

peremen = rand() % q + 1;

vec.push_back(peremen);

}

cout << "Array before sort:" << endl;

for (int i = 0; i < q; i++)

cout << vec[i] << ' ';

cout << endl << endl;

auto t1 = Clock::now();


while (gr < vec.size() - 1)//начало цикла прохода по всему массиву от первого элемента до последнего с индексом N-1

{

min2 = gr;//присваивание переменной min индекса первого элемента на данном этапе

nte = gr + 1;//номер текущего элмента равен границе исследуемого массива + 1(для альнейшего анализа)
```

```
while (nte < vec.size())//вложенный цикл будет выполняться, пока номер текущего элемента меньше размера самого массива
{
if (vec[min2] > vec[nte])//сравнение значений текущего элемента и значения элемента неотсортированной части(увеличение до тех пор, пока не закончится массив)
{
min2 = nte;//рписваивание переменной min2 номер аминимального элемента
}
nte++;//(увеличение номера текущего элемента до тех пор, пока не закончится массив)
}
temp = vec[gr];//производение обмена значений переменных, используя буферную переменную buf
vec[gr] = vec[min2];
vec[min2] = temp;
gr++;//продвижение по массиву на шаг вперед
}

auto t2 = Clock::now();

cout << "Array after sort:" << endl;
for (int i = 0; i < q; i++)
cout << vec[i] << ' ';

cout << endl << endl << "Duration(nanoseconds) is: " <<
std::chrono::duration_cast<std::chrono::nanoseconds>(t2 - t1).count() << endl;

system("pause");

return 0;//завершение программы
}
```

Таблица данных:

Debug version	Algorithm execution time(nanoseconds)			
Element quantity	Selection			Average
10	4100	4000	4100	4066.667
100	196400	228300	208100	210933.3
500	4513800	5460700	4197400	4723967
1000	17639900	17358000	16970500	17322800
2500	1.04E+08	1.06E+08	1.06E+08	1.05E+08



Алгоритм сортировки «Гномья»

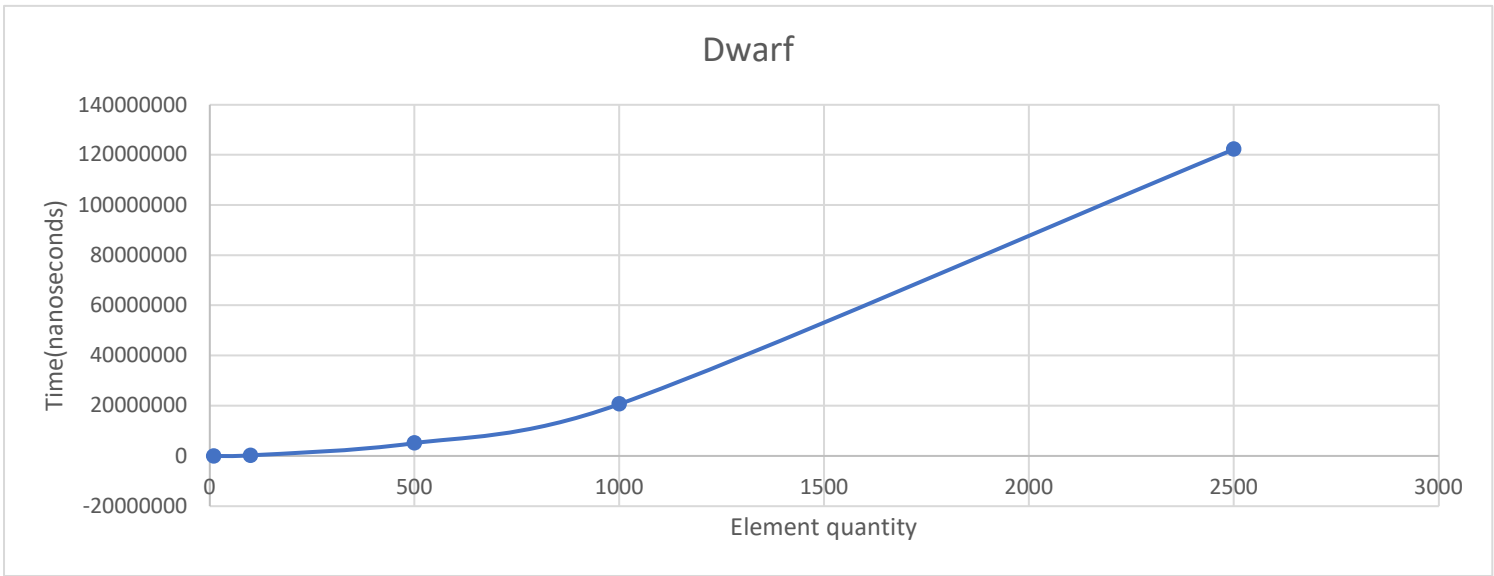
Алгоритм сортировки «Гномья» реализован на языке программирования C++(далее листинг программы):

```
#include <iostream>

#include <vector>

#include <chrono>

using namespace std;

typedef std::chrono::high_resolution_clock Clock;

int main()

{

int j = 2, i = 1, buf;

vector<int> vec;

int q, peremen;

cout << "Enter the amount of element:";

cin >> q;

for (int k = 0; k < q; k++)

{

peremen = rand() % q + 1;

vec.push_back(peremen);

}

cout << "Array before sort:" << endl;

for (int i = 0; i < q; i++)

cout << vec[i] << ' ';

cout << endl << endl;

auto t1 = Clock::now();

while (i < vec.size()) {//íà÷àëî öèêëà ïðîõîäà ïî âñåìó ìàññèâó

if (vec[i - 1] > vec[i]) //ïðîâåðêà óñëîâèÿ, åñëè ïðåäûäóùèé ýëåìåíò áîëüøå òåêóùåãî

{

buf = vec[i];//ïðîèçâîäèì îáìåí çíà÷åíèé ïåðåìåííûõ ïîñðåäñòâîì áóôåðíîé ïåðåìåííîé

vec[i] = vec[i - 1];

vec[i - 1] = buf;

i--;//ò.ê. ìû ïîïàëè â óñëîâèå, çíà÷èò, ó íàñ ïðîèçîøëà ïåðåñòàíîâêà è íàì âàæíî ïîíÿòü, ìåíüøå ëè âñå ïðåäûäóùèå çíà÷åíèÿ è ïðàâèëüíûé ëè òàì ïîðÿäîê, äåëàì øàã íàçàä

if (i > 0) //åñëè i áîëüøå íóëÿ, ïðîäîëæàåì èòåðàöèþ öèêëà
```

```
continue;

}

i = j++;//ïðèðàâíèâàåì íîâûé íîìåð ê çàðàíåå ñîõðàíåííîìó(ýòî ïîçâîëÿåò àëãîðèòìó ïðåðûâàòüñÿ íà òî ìåñòî, îòêóäà îí íà÷àë óõîäèòü â àíàëèç ïðåäûäóùèõ ýëåìåíòîâ))

}

auto t2 = Clock::now();

cout << "Array after sort:" << endl;

for (int i = 0; i < q; i++)

cout << vec[i] << ' ';

cout << endl << endl << "Duration(nanoseconds) is: " <<
std::chrono::duration_cast<std::chrono::nanoseconds>(t2 - t1).count() << endl;

system("pause");

return 0;

}
```

Таблица данных:

Element quantity	Dwarf			Average
10	4000	3200	3200	3466.667
100	230600	206900	256200	231233.3
500	5293800	5112200	5146700	5184233
1000	20725700	19726600	21714300	20722200
2500	1.21E+08	1.24E+08	1.21E+08	1.22E+08



Алгоритм сортировки «Быстрая»

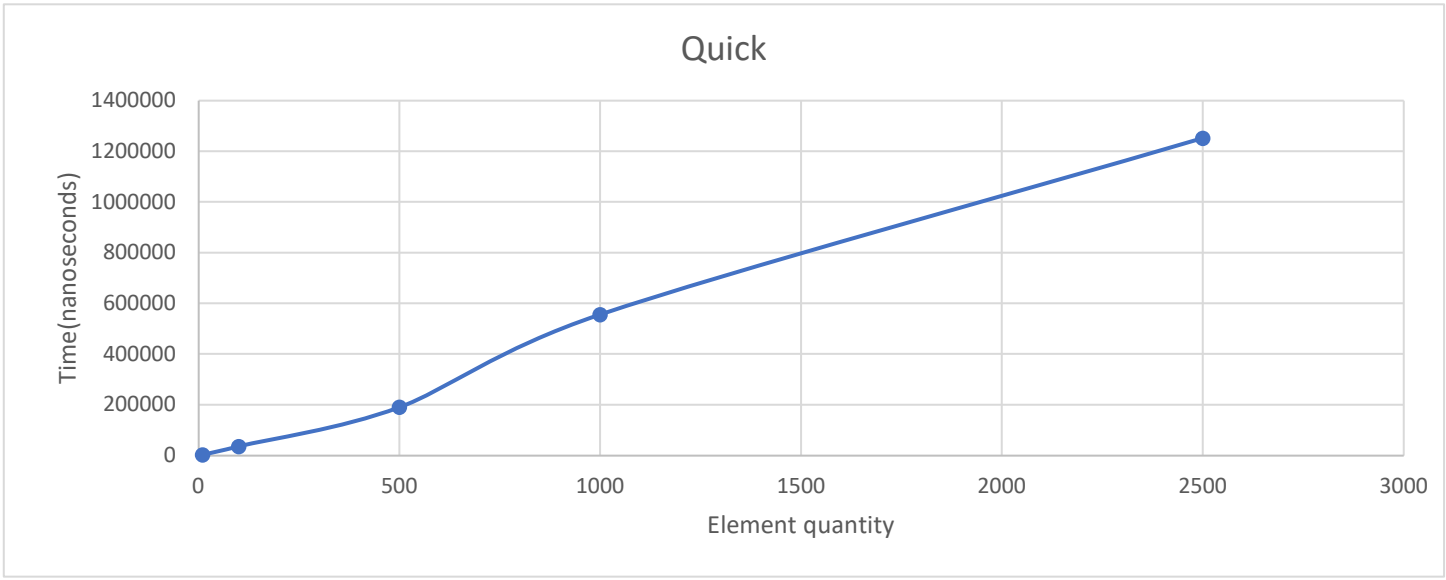
Алгоритм сортировки «Быстрая» реализован на языке программирования C++(далее листинг программы):

```
#include <iostream>//подключение необходимых библиотек

#include <vector>

#include <chrono>

using namespace std;

typedef std::chrono::high_resolution_clock Clock;

vector<int> vec;

int q, peremen;

void qsort(vector<int>& vec, int left, int right)//объявление подфункции сортировки

{

int i = left, j = right;//объявление переменных крайнего левого положения и крайнего праввого положения

int temp, pivot = vec[(left + right) / 2];//поиск центра(поиск опорного элемента)

while (i <= j)//цикл выполняется, пока границы не сомкнутся

{

while (vec[i] < pivot) //если значение анализируемого элемента из левой части меньше значения опорного элемента

i++;//мы сдвигаем левую границу на одну единицу вправо

while (vec[j] > pivot) //если значение анализируемого элемента из правой части больше значения опорного элемента

j--;//мы сдвигаем правую границу на одну единицу влево

//сдвигаем границы к центру - к опорному элементу

if (i <= j)

{

if (vec[i] > vec[j])//выполнение обмена, если значение текущего элемента в левой части больше значения текущего элемента в правой части

{

temp = vec[i];//обмен значений переменных с использованием буферной переменной temp

vec[i] = vec[j];

vec[j] = temp;

}

i++;

j--;//уменьшнеие границ диапазона

}
```

```
}

if (left < j)

qsort(vec, left, j);//если обнаружено нарушение порядка в одной из частей, вызывается функция сортировки определенной части

if (i < right)

qsort(vec, i, right);//если обнаружено нарушение порядка в одной из частей, вызывается функция сортировки определенной части

}

//----------------------------------------------------------------

int main()//начало главной функции

{

cout << "Enter the amount of element:";

cin >> q;

for (int k = 0; k < q; k++)

{

peremen = rand() % q + 1;

vec.push_back(peremen);

}

cout << "Array before sort:" << endl;

for (int i = 0; i < q; i++)

cout << vec[i] << ' ';

cout << endl << endl;

auto t1 = Clock::now();

qsort(vec, 0, vec.size() - 1); // вызов функции сортировки

auto t2 = Clock::now();

cout << "Array after sort:" << endl;

for (int i = 0; i < q; i++)

cout << vec[i] << ' ';

cout << endl << endl << "Duration(nanoseconds) is: " <<
std::chrono::duration_cast<std::chrono::nanoseconds>(t2 - t1).count() << endl;

system("pause");

return 0;}
```

Таблица данных:

Debug version	Algorithm execution time(nanoseconds)			
Element quantity	Quick			Average
10	3200	3000	3400	3200
100	43700	31700	33300	36233.33
500	190000	191000	190300	190433.3
1000	625800	520800	520900	555833.3
2500	1443800	1156200	1154700	1251567

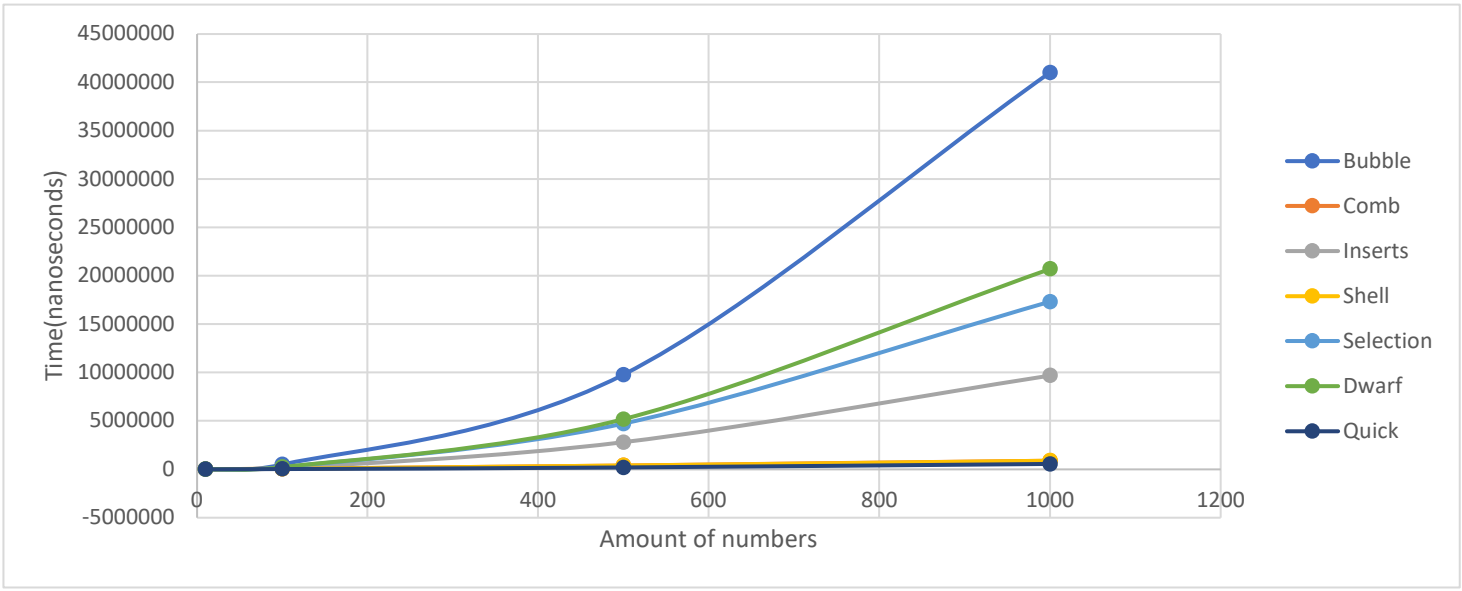


Сопоставление алгоритмов:

Ниже представлена таблица времени работы всех алгоритмов:

Debug version	Algorithm execution time(nanoseconds)				Algorithm execution time(nanoseconds)				Algorithm execution time(nanoseconds)			
Element quantity	Bubble			Average	Comb			Average	Inserts			Average
10	4700	4900	4900	4833.333	3100	3000	3600	3233.333	2500	2400	2700	2533.333
100	447100	483200	563400	497900	61900	46500	55000	54466.67	96700	147200	146700	130200
500	9596800	9652600	10088100	9779167	521700	347800	347200	405566.7	2583400	2308900	3496100	2796133
1000	41237100	41932300	39885900	41018433	814300	1017300	813800	881800	8864800	9237200	10994300	9698767
2500	1.51E+08	2.43E+08	2.37E+08	2.1E+08	2382700	3576700	2966900	2975433	56894800	53656200	55336300	55295767
	Algorithm execution time(nanoseconds)				Algorithm execution time(nanoseconds)				Algorithm execution time(nanoseconds)			
Element quantity	Shell			Average	Selection			Average	Dwarf			Average
10	2900	3500	2900	3100	4100	4000	4100	4066.667	4000	3200	3200	3466.667
100	49100	51100	75400	58533.33	196400	228300	208100	210933.3	230600	206900	256200	231233.3
500	443300	295000	442000	393433.3	4513800	5460700	4197400	4723967	5293800	5112200	5146700	5184233
1000	721300	901200	1081900	901466.7	17639900	17358000	16970500	17322800	20725700	19726600	21714300	20722200
2500	2396900	2694000	3711500	2934133	1.04E+08	1.06E+08	1.06E+08	1.05E+08	1.21E+08	1.24E+08	1.21E+08	1.22E+08

Algorithm execution time(nanoseconds)			
Quick			Average
3200	3000	3400	3200
43700	31700	33300	36233.33
190000	191000	190300	190433.3
625800	520800	520900	555833.3
1443800	1156200	1154700	1251567



При необходимости, вы можете найти всю историю разработки программы на моем GitHub:
https://github.com/DmitriiDerendyaev/algoritmization_and_programming