МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

Отчет по выполнению индивидуального задания № 1

по дисциплине «Структуры и алгоритмы обработки данных»

Тема: «Эмпирический анализ алгоритмов сортировки»

Выполнил: Хан Анастасия Александровна
Студент группы ИКБО-29-20

Проверил: Копылова А.В.

Москва 2021

СОДЕРЖАНИЕ

Отчет по заданию 3.
Оценка эффективности алгоритмов простых сортировок

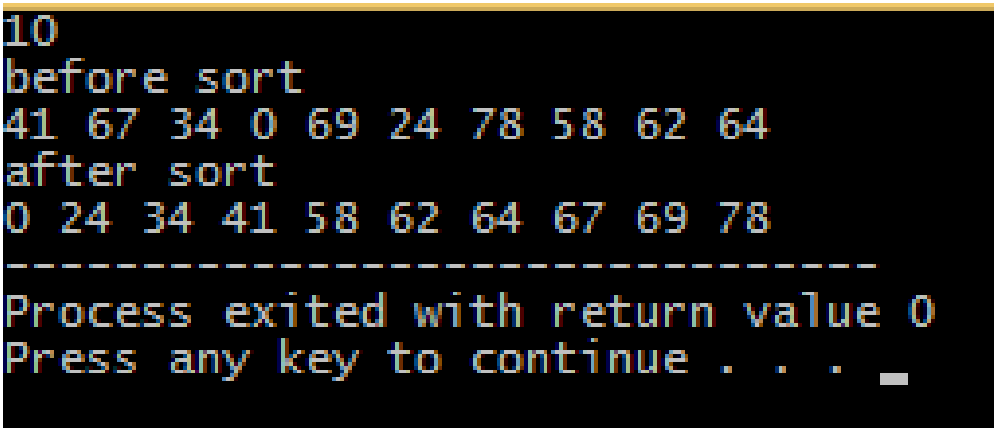
Отчет по заданию 1
Зависимость вычислительной сложности алгоритма сортировки массива от размера массива n

Персональный вариант: 29%3 + 1 = 3

1. **Алгоритм сортировки по методу варианта Алгоритм задания 1 на случайно заполненном массиве.**

```
void selection_sort(int* a, int n)
{
  int mini;
  int buff;
  for (int i = 0; i < n - 1; i++)
  {
        mini = i;
        for (int j = i + 1; j < n; j++)
        {
              if (a[j] < a[mini])
              {
                    mini = j;
              }
        }
        if (mini != i)
        {
              buff = a[i];
              a[i] = a[mini];
              a[mini] = buff;
        }
  }
}
```

Работа функции сортировки выбором на случайно заполненном массиве:



2. **Процесс определения функции роста времени выполнения сортировки методом Алгоритм задания 1 при увеличении объема массива n**

Оператор	**Время выполнения инструкции**	**Количество выполнений оператора**
int mini;	c_1	1
int buff;	c_2	1
for (int i = 0; i < n - 1; i++)	c_3	n
{		
mini = I;	c_4	n-1
for (int j = i + 1; j < n; j++)	c_5	$\sum_{j=1+i}^{n} t_j$
{		
if (a[j] < a[mini])	c_6	$\sum_{j=1+i}^{n} t_j - 1$
{		
mini = j;	c_7	$\sum_{j=1+i}^{n} t_j - 1$
}		
}		
if (mini != i)	c_8	n-1
{		
buff = a[i];	c_9	$\sum_{j=k}^{n-1} t_j,$
a[i] = a[mini];	c_{10}	$\sum_{j=k}^{n-1} t_j,$
a[mini] = buff;	c_{11}	$\sum_{j=k}^{n-1} t_j,$
}		
}		

((n – 1) – k) – количество раз, когда условие mini != I выполнилось

Время выполнения алгоритма: $T(n) = c_1 + c_2 + c_3*n + c_4*(n-1) + c_5* *\sum_{j=1+i}^{n} t_j + c_6*(\sum_{j=1+i}^{n} t_j - 1) + c_7*(\sum_{j=1+i}^{n} t_j - 1) + c_8*(n-1) + c_9*(\sum_{j=k}^{n-1} t_j,) + c_{10}*(\sum_{j=k}^{n-1} t_j,) + c_{11}*(\sum_{j=k}^{n-1} t_j,)$

Рассмотрим худший случай, когда все элементы стоят по убыванию, чтобы определить O(n):

Тогда $\sum_{j=1+i}^{n} t_j - 1$ превращается в формулу для суммы членов арифметической прогрессии, $\sum_{j=1+i}^{n} t_j - 1 = (1 + 2 + \ldots + n) - 1 = (1 + n)*n/2 - 1$; а $(n - 1 - k) = 0$ в формуле $\sum_{j=k}^{n-1} t_j$, что значит, что все элементы переставляются и $\sum_{j=0}^{n-1} t_j = n - 1$

Подставим полученное выражение в формулуT(n):

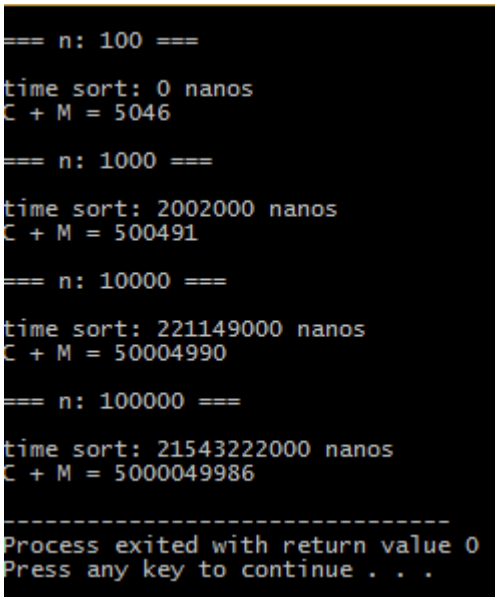
$T(n) = c_1 + c_2 + c_3*n + c_4*(n-1) + c_5* *(1 + n)*n/2 + c_6*((1 + n)*n/2 - 1) + c_7*((1 + n)*n/2 - 1) + c_8*(n-1) + c_9*(n-1) + c_{10}*(n-1) + c_{11}*(n-1) = n^2/2*(c^5 + c_6 + c_7) + n*(c_3 + c_4 + c_5/2 + c_6/2 + c_7/2 + c_8 + c_9 + c_{10} + c_{11}) + (c_1 + c_2 - c_4 - c_6 - c_7 - c_8 - c_9 - c_{10} + c_{11}) = n^2/2 + n \Rightarrow O(n) = n^2$

3. Сводная таблица результатов выполнения сортировки по указанным объемам по формату Таблица 1 на случайно заполненном массиве для всех указанных объемов

Таблица1

n	T(n)	$T_т = f(C + M)$	$T_п = C_ф + M_ф$
100	0	10000	5046
1000	0,003	1000000	500491
10000	0,221	100000000	50004990
100000	21,6	10000000000	5000049986

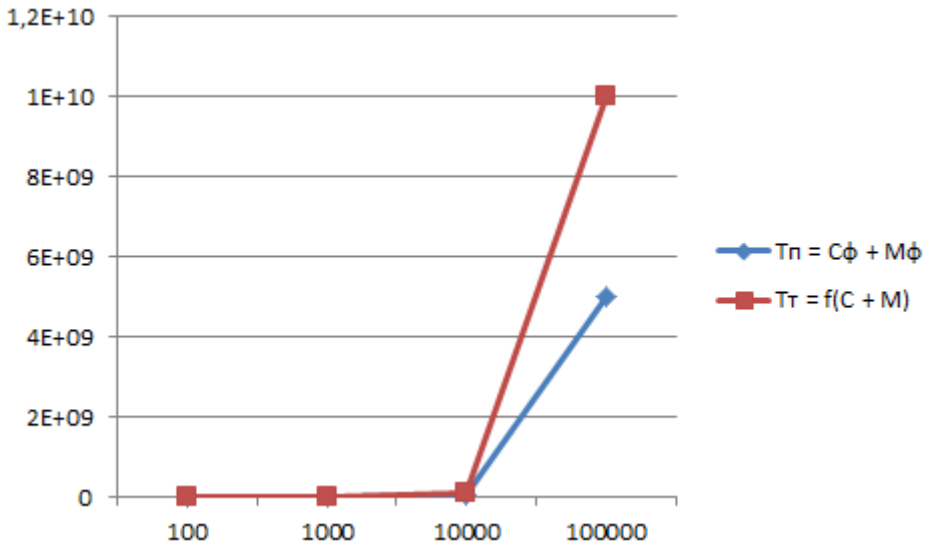
4. Код алгоритма и основной программы, доказывающей выполнение тестирования




```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <chrono>
using namespace std;

using namespace chrono;
void input(int* a, int n)
{
	for (int i = 0; i < n; i++)
	{
		a[i] = rand()%1000;
	}
}

void output(int* a, int n)
{
	cout << endl;
	for (int i = 0; i < n; i++)
	{
		cout << a[i] << ' ';
	}
}

unsigned long long selection_sort(int* a, int n)
{
	int buff, mini;
	unsigned long long operSum = 0;
	for (int i = 0; i < n - 1; i++)
```

```
	{
		mini = i;
		for (int j = i + 1; j < n; j++)
		{
			operSum++;
			if (a[j] < a[mini])
			{
				operSum++;
				mini = j;
			}
		}
		if (mini != i)
		{
			operSum++;
			buff = a[i];
			a[i] = a[mini];
			a[mini] = buff;
}		}
	return operSum;
}

int main()
{
	srand(time(0));
	int a[100000], n;
	unsigned long long operSum;
	n = 100;
	while (n < 1000000)
	{
	cout << "=== n: " << n << " ===" << endl;
	input(a, n);
	auto begin = high_resolution_clock::now();
	operSum = selection_sort(a, n);
	auto end = high_resolution_clock::now();
	auto dur = duration_cast<nanoseconds>(end - begin);
	cout << "time sort: " << dur.count() << " nanos" <<  endl;
	cout << "C + M = " << operSum << endl;
	n*=10;
	}
}
```

5. **Представить график зависимости теоретической и практической вычислительной сложности алгоритма**



6. Анализ результатов

Посмотрев практические данные и теоретические, можно сказать, что время выполнения алгоритма квадратично зависит от количества элементов и теоретическая оценка, сопоставленная с практическими данными, носит приблизительный, оценочный характер.

Отчет по заданию 2. Зависимость сложности алгоритма сортировки от упорядоченности массива (худший и лучший случаи)

1. Таблица результатов при применении метода к массиву, упорядоченному по возрастанию

Таблица 2

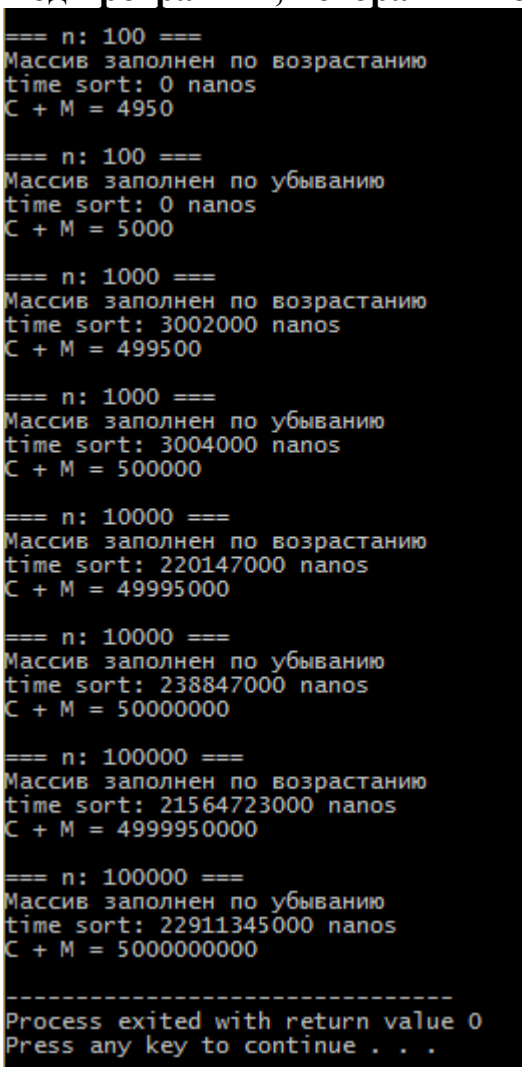
n	T(n)	$T_т = f(C + M)$	$T_п = C_ф + M_ф$
100	0	10000	4950
1000	0,003	1000000	499500
10000	0,221	100000000	49995000
100000	21,667	10000000000	4999950000

2. Таблица результатов при применении метода к массиву, упорядоченному по убыванию

Таблица 3

n	T(n)	$T_т = f(C + M)$	$T_п = C_ф + M_ф$
100	0	10000	5000
1000	0,003	1000000	500000
10000	0,241	100000000	50000000
100000	22,911	10000000000	5000000000

3. Код программы, которая выполняет тестирование алгоритма

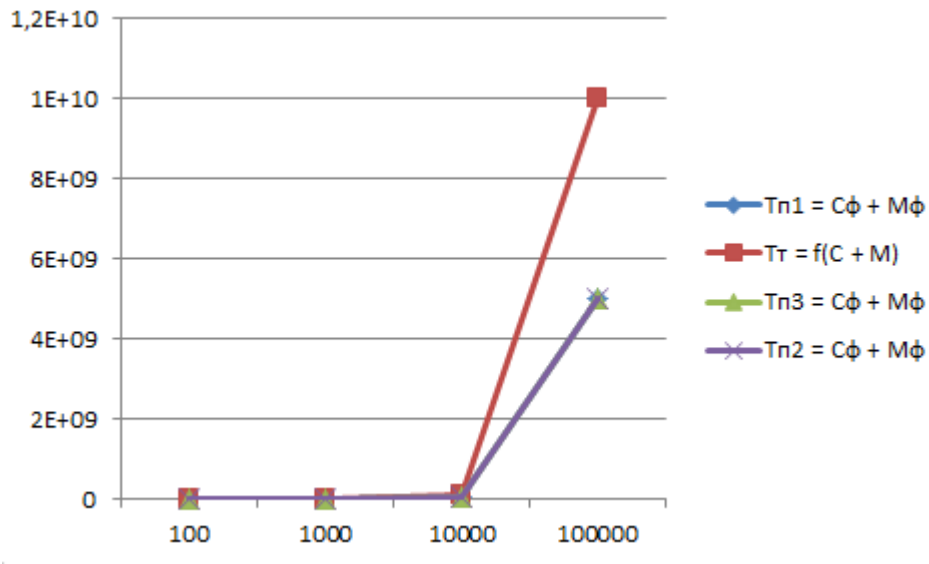



```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <chrono>
using namespace std;

using namespace chrono;
void input(int* a, int n, bool flag)
{
	if (flag == true)
	{
		for (int i = 0; i < n; i++)
		{
			a[i] = i;
		}
	}
	else
```

```
	{
		for (int i = 0; i < n; i++)
		{
			a[i] = n - i;
		}
	}
}

void output(int* a, int n)
{
	cout << endl;
	for (int i = 0; i < n; i++)
	{
		cout << a[i] << ' ';
	}
}

unsigned long long selection_sort(int* a, int n)
{
	int buff, mini;
	unsigned long long operSum = 0;
	for (int i = 0; i < n - 1; i++)
	{
		mini = i;
		for (int j = i + 1; j < n; j++)
		{
			operSum++;
			if (a[j] < a[mini])
			{
				operSum++;
				mini = j;
			}
		}
		buff = a[i];
		a[i] = a[mini];
		a[mini] = buff;
	}
	return operSum;
}

int main()
{
	setlocale(LC_ALL, "Russian");
	srand(time(0));
```

```
	int a[100000], n;
	unsigned long long operSum;
	n = 100;
	bool flag = true;
	while (n < 1000000)
	{
		input(a, n, flag);
		auto begin = high_resolution_clock::now();
		operSum = selection_sort(a, n);
		auto end = high_resolution_clock::now();
		auto dur = duration_cast<nanoseconds>(end - begin);
		cout << endl << "=== n: " << n << " ===" << endl;
		if (flag == true)
		{
			cout << "Массив заполнен по возрастанию";
			flag = false;
		}
		else
		{
			cout << "Массив заполнен по убыванию";
			flag = true;
			n*=10;
		}
		cout << endl << "time sort: " << dur.count() << " nanos" <<  endl;
		cout << "C + M = " << operSum << endl;
	}
}
```

4. **График зависимости теоретической и практической вычислительной сложности алгоритма для трех рассмотренных случаев: по Таблице 1, по Таблице 2, по Таблице 3**



5. **Определить емкостную сложность алгоритма от n**

Для сортировки не требуется дополнительного массива, следовательно, ёмкостная сложность равна n.

6. **Анализ результатов и выводы**

По полученным практическим данным, по графикам можно судить, что алгоритм сортировки выбором выполняется в худшем и в лучшем случае с незначительной разницей в количестве операций и времени.

Отчет по заданию 3.
Оценка эффективности алгоритмов простых сортировок

Персональный вариант: 29%3 + 1 = 3

1. **Алгоритм сортировки по методу варианта Алгоритм задания 3**

```
void bubble_sort(int* a, int n)
{
  int buff;
  for (int i = 0; i < n - 1; i++)
  {
        for (int j = i + 1; j < n; j++)
        {
              if (a[i] > a[j])
              {
                    buff = a[i];
                    a[i] = a[j];
                    a[j] = buff;
              }
        }
  }
}
```

2. **Процесс определения функции роста времени выполнения сортировки методом Алгоритм задания 3 при увеличении объема массива n**

Оператор	**Время выполнения инструкции**	**Количество выполнений оператора**
int buff;	c_1	1
int buff;	c_2	1
for (int i = 0; i < n - 1; i++)	c_3	n
{		
for (int j = i + 1; j < n; j++)	c_4	$\sum_{j=1+i}^{n} t_j$
{		
if (a[i] < a[j])	c_5	$\sum_{j=1+i}^{n} t_j - 1$
{		

buff = a[i];	c_6	$\sum_{j=k}^{n} t_j - 1$
a[i] = a[mini];	c_7	$\sum_{j=k}^{n} t_j - 1$
a[mini] = buff;	c_8	$\sum_{j=k}^{n} t_j - 1$
}		
}		
}		

n – k – количество раз, когда условие a[i] <a[j] выполнилось

Время выполнения алгоритма: $T(n) = c_1 + c_2 + c_3*n + c_4*(\sum_{j=1+i}^{n} t_j) + c_5*(\sum_{j=1+i}^{n} t_j - 1) + c_6*(\sum_{j=k}^{n} t_j - 1) + c_7*(\sum_{j=k}^{n} t_j - 1) + c_8*(\sum_{j=k}^{n} t_j - 1)$

Рассмотрим худший случай, когда все элементы стоят по убыванию, чтобы определить O(n):

Тогда $\sum_{j=1+i}^{n} t_j - 1$ превращается в формулу для суммы членов арифметической прогрессии, $\sum_{j=1+i}^{n} t_j - 1 = (1 + 2 + \ldots + n) - 1 = (1 + n)*n/2 - 1 = n^2/2 + n/2 - 1$; $\sum_{j=1+i}^{n} t_j = n^2/2 + n/2$; по мере сортировки n – k = n-1; n-2…1, тогда $\sum_{j=k}^{n} t_j - 1 = (1 + n - 1)/2*(n-1) - 1 = n^2/2 - n/2 - 1$.

Подставим полученное выражение в формулуT(n):

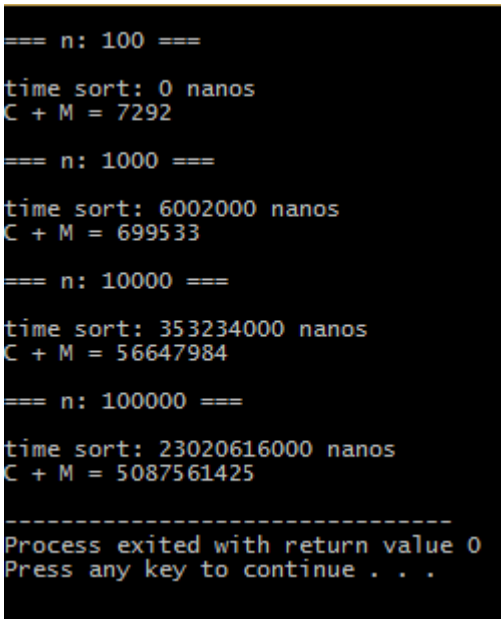
$T(n) = c_1 + c_2 + c_3*n + c_4*(n^2/2 + n/2) + c_5*(n^2/2 + n/2 - 1) + c_6*(n^2/2 - n/2 - 1) + c_7*(n^2/2 - n/2 - 1) + c_8*(n^2/2 - n/2 - 1) = n^2*(c_4 + c_5 + c_6 + c_7 + c_8) + n*(c_3 + c_4/2 + c_5/2 - c_6/2 - c_7/2 - c_8/2) + (c_1 + c_2 + c_3 - c_5 - c_6 - c_7 - c_8) = n^2 + n \Rightarrow O(n) = n^2$

3. Сводная таблица результатов выполнения сортировки по указанным объемам по формату Таблица 4 на случайно заполненном массиве для всех указанных объемов

Таблица4

n	**T(n)**	**$T_т = f(C + M)$**	**$T_п = C_ф + M_ф$**
100	0	10000	7292
1000	0,006	1000000	699533
10000	0,353	100000000	50490873
100000	23,02	10000000000	5087561425

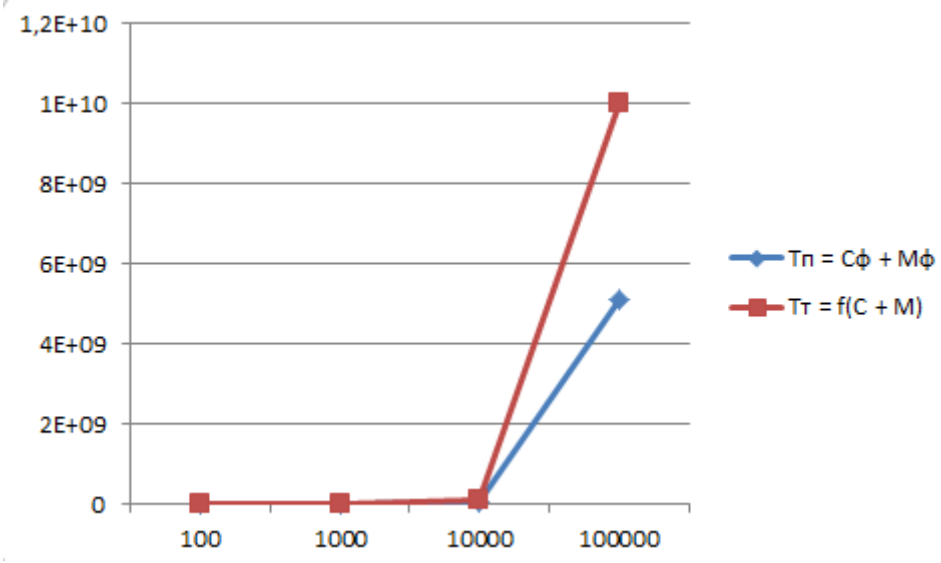
4. **Код алгоритма и основной программы, доказывающей выполнение тестирования**




```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <chrono>
using namespace std;

using namespace chrono;
void input(int* a, int n)
{

	for (int i = 0; i < n; i++)
	{
		a[i] = rand()%100 + rand()%1000 - rand()%800;
	}
}

void output(int* a, int n)
{
	cout << endl;
	for (int i = 0; i < n; i++)
	{
		cout << a[i] << ' ';
	}
}

unsigned long long bubble_sort(int* a, int n)
{
```

```
	int buff;
	unsigned long long operSum = 0;
	for (int i = 0; i < n - 1; i++)
	{
		for (int j = i + 1; j < n; j++)
		{
			operSum++;
			if (a[i] > a[j])
			{
				operSum++;
				buff = a[i];
				a[i] = a[j];
				a[j] = buff;
			}
		}
	}
	return operSum;
}

int main()
{
	setlocale(LC_ALL, "Russian");
	srand(time(0));
	int a[100000], n;
	unsigned long long operSum;
	n = 100;
	bool flag = true;
	while (n < 1000000)
	{
		input(a, n);
		auto begin = high_resolution_clock::now();
		operSum = bubble_sort(a, n);
		auto end = high_resolution_clock::now();
		auto dur = duration_cast<nanoseconds>(end - begin);
		cout << endl << "=== n: " << n << " ===" << endl;
		cout << endl << "time sort: " << dur.count() << " nanos" <<  endl;
		cout << "C + M = " << operSum << endl;
		n*=10   ;
	}
}
```

5. **Представить график зависимости теоретической и практической вычислительной сложности алгоритма**



6. **Определение эффективности алгоритма**

Для сравнения алгоритмов принято использовать обобщенную характеристику, называемую эффективностью. Говорят, что алгоритм А1, эффективнее алгоритма А2, если алгоритм А1, выполняется за меньшее время и (или) требует меньше компьютерных ресурсов (оперативной памяти, дискового пространства, сетевого трафика и т.п.).

7. **Анализ результатов и определение более эффективного алгоритма**

Посмотрев практические данные и теоретические, сравнив графики, можно сказать, что двух алгоритмов сортировки – выбором и пузырьковой – эффективнее выбором, потому что она выполняется за меньшее время.

Выводы

В ходе практической работы я получила знания и практические навыки определения теоретической и практической сложности алгоритмов, научилась определять эффективности алгоритмов.

Информационные источники

1) Лекции по Структурам и алгоритмам обработки данных Скворцова Л. А. – МИРЭА – Российский технологический университет 2021 год
2) Искусство программирования Т. 1 / Д. Кнут – М.: Вильямс, 2014. – 712 с.
3) https://pro-prof.com/archives/4275