ЛАБОРАТОРНАЯ РАБОТА № 6.
“АНАЛИЗ АЛГОРИТМОВ СОРТИРОВКИ”

1. ЦЕЛЬ РАБОТЫ

Научиться оценивать сложность и количество операций для алгоритмов сортировки.

2. ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

Задача сортировки последовательности богата существующими алгоритмами её решения. В данной работе предлагается рассмотреть и реализовать некоторые из алгоритмов сортировки и сравнить их производительность.

Вначале сформулируем задачу сортировки.

Определение: *Задача сортировки заключается в построении алгоритма со следующими входом и выходом:*

INPUT: *последовательность* $A = (a_1, \cdots,$ *сравнимых элементов некоторого множества.*

OUTPUT: *перестановка* $A' = (a'_1, \cdots, a'_n)$, *в которой* $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

2.1 Сортировка вставками

В данном алгоритме главной идеей является следующее: предполагается, что массив уже отсортирован до некоторого элемента с номером i, тогда просто необходимо вставить этот элемент на необходимое место.

```
void insertSort(int *a, int n) {
	int i, j;
	for(i = 2; i <= n; i++) {	// установить i-й элемент на свое место
		a[0] = a[i];		// запомнить i-й элемент в нулевой ячейке
		for(j = i-1; a[j] > a[0]; j--)// пока j-ый элемент больше текущего
			a[j+1] = a[j];		// сдвигать его влево
		a[j+1] = a[0];		// на свободное место вернуть исходный элемент
	}
	}
```

2.2 Сортировка прямым выбором

Идея сортировки выбором заключается в поиске максимального элемента в неупорядоченной части последовательности и его последующего исключения путём записи в конец массива.

```
void selectSort(int *a, int n) {
	int i, j, imax;
	for(i = n; i > 1; i--) { // заполнение i-йячейки
		imax = 1;
		for(j = 1; j <= i; j++) // поиск максимального элемента
			if(a[j] > a[imax])
				imax = j;
		swap(a[i], a[imax]);//вставка максимального элемента на свое место
	}
    }
```

2.3 Сортировка пузырьком

При сортировке пузырьком в последовательности сравниваются соседние элементы и, если они не упорядочены, меняются местами.

```
void bubbleSort(int *a, int n) {
	int i, j;
	for(i = n; i > 1; i--)
		for(j = 1; j < i; j++)		// выталкивается самый тяжелый
			if(a[j] > a[j+1])	// если он больше своего соседа справа
				swap(a[j], a[j+1]);
}
```

2.4 Сортировка слиянием

Данный алгоритм сортировки относится к классу «разделяй и властвуй». Массив будет делиться на две половины, а затем функция сортировки будет вызывать саму себя для сортировки каждой из половин. После этого нам необходимо совместить два уже отсортированных массива, что можно сделать с помощью двух указателей.

```
void merge(int *a, int l, int r) {
	int m = (l+r) / 2;
	int i, j, k;

	// b - промежуточный массив, объявленный как глобальная переменная
```

```
	i = l;  // начало первой половины
	j = m+1; // начало второй половины
	k = l;
	while(i <= m || j <= r) { // пока в массив не добавятся все элементы
		if(i > m) { // если в первой половине чисел больше нет
			b[k++] = a[j++]; // добавляется число из второй половины
			continue;
		}
		if(j > r) { // если во второй половине чисел больше нет
			b[k++] = a[i++]; // добавляется число из первой
			continue;
		}
		if(a[i] < a[j]) // если в первой половине минимальное число меньше //минимального во второй
			b[k++] = a[i++]; // добавляем число из первой
		else  // иначе
			b[k++] = a[j++]; // из второй
	}
	for(i = l; i <= r; i++) // возвращение данных из промежуточного массива //в исходный
		a[i] = b[i];
}
void mergeSort(int *a, int l, int r) {
	if(l == r) // если один элемент, то он отсортирован
		return;
	int m = (l + r) / 2; // найти середину
	mergeSort(a, l, m);	// отсортивать первую половину
	mergeSort(a, m+1, r);  // отсортировать вторую половину
	merge(a, l, r);	// совместить две половины, не теряя свойства
	//возрастания
    }
```

2.5 Быстрая сортировка

Алгоритм быстрой сортировки заключается в следующем: выбирается некоторый элемент (в примере это будет центральный элемент, но наиболее эффективно было бы выбрать случайный), а затем массив изменяется таким образом, что в первой его части будут все элементы не большие выбранного числа, а во второй все остальные. То есть друг относительно друга две эти части будут отсортированы и нам останется только вызвать рекурсивно алгоритм для каждой из частей.

```
void quickSort(int *a, int n) {
```

```
	int i = 0, j = n;		// поставить указатели на исходные места
	int temp, p;

	p = a[n/2];		// центральный элемент

	// процедураразделения
	do {
		while (a[i] < p)
			i++;
		while (a[j] > p)
			j--;

		if (i <= j) {
			swap(a[i], a[j]);
			i++;
			j--;
		}
	}
	while(i <= j);
	// рекурсивные вызовы, если есть, что сортировать
	if(j > 0)
		quickSort(a, j);
	if(n > i)
		quickSort(a+i, n-i);
}
```

2.6 Сортировка Шелла

Данная сортировка является модификацией сортировки вставки, но с использованием шага, то есть рассматриваются не все элементы, а только те, которые находятся на расстоянии шага. Шаг постоянно уменьшается в два раза, что позволяет добиться сложности O(log(N)).

```
void shellSort(int *a, int n) {
	int i, j, temp, step = n / 2;//инициализацияшага
	while (step > 0) { //покашагненулевой
		for (i = 0; i < (n - step); i++) {
			j = i;
//будем идти начиная с i-го элемента
			while (j >= 0 && a[j] > a[j + step]) {
				//пока не достигнуто начало массива
				//и пока рассматриваемый элемент больше
				//чем элемент находящийся на расстоянии шага
				//элементы меняются местами
				swap(a[j], a[j+step]);
```

```
                        j--;
                    }
                }
        step = step / 2;//уменьшение шага
        }
}
```

2.7 Пирамидальная сортировка

Пирамидальная сортировка основывается на построение кучи(пирамиды) - бинарного дерева, в котором элемент каждого узла больше, чем элементы его потомков. После построение такого дерева несложно найти отсортированный массив используя алгоритм слияния.

```
void heapSort(int *a, int n) {
        int i, sh = 0; //смещение
bool b = false;
for(;;) {
                b = false;
                for (i = 0; i < n; i++ ) {
                        if( i * 2 + 2 + sh < n ) {
                                if( ( a[i + sh] > a[i * 2 + 1 + sh] ) || ( a[i + sh] >  a[i * 2
+ 2 + sh] ) ) {
                                        if ( a[i * 2 + 1 + sh] < a[i * 2 + 2 + sh] ) {
                                                swap( a[i + sh], a[i * 2 + 1 + sh] );
                                                b = true;
                                        }
                                        else
                                        if ( a[i * 2 + 2 + sh] < a[ i * 2 + 1 + sh]) {
                                                swap( a[ i + sh], a[i * 2 + 2 + sh]);
                                                        b = true;
                                                }
                                }
                                //дополнительная проверка для последних двух
элементов
//с помощью этой проверки можно отсортировать пирамиду
//состоящую всего лишь из трех элементов
                                if( a[i*2 + 2 + sh] < a[i*2 + 1 + sh] ) {
                                        swap( a[i*2+1+sh], a[i * 2 +2+ sh] );
                                        b = true;
                                }
                        }
                        else
                                if( i * 2 + 1 + sh < n ) {
                                        if( a[i + sh] > a[ i * 2 + 1 + sh] ) {
```

```
                                swap( a[i + sh], a[i * 2 + 1 + sh] );
                                b = true;
                            }
                        }
                }
                if (!b)
                        sh++; //смещение увеличивается, когда на текущем
этапе
                //сортировать больше нечего
                if ( sh + 2 == n ) break;
        } //конец сортировки
}
```

3. ХОД РАБОТЫ

1. Для каждого из приведённых алгоритмов найдите оценку для количества шагов и количества требуемой памяти.
2. Создайте структуру и реализуйте алгоритм сортировки согласно варианту задания. Критерий сортировки также указан в варианте.
3. Реализуйте более эффективные алгоритмы сортировки согласно варианту задания.
4. Сравните производительность различных алгоритмов.

4. ВАРИАНТЫ ЗАДАНИЙ

Таблица 1 – Варианты заданий для выполнения лабораторной работы

	Простейший алгоритм 1	Простейший алгоритм 2	Быстрый алгоритм 3	Структура и критерий сортировки
1	Метод «пузырька»	Сортировка вставками	Сортировка Шелла	Структура дата, содержит день, месяц и год, сортировка по возрастанию дат
2	Модифицированный метод «пузырька»	Сортировка посредством выбора	«Быстрая» сортировка	Структура дата, содержит день, месяц и год, сортировка по убыванию дат
3	Сортировка вставками	Метод «пузырька»	Сортировка слиянием	Структура Студент, содержит ФИО, курс, факультет, сортировка по возрастанию курса
4	Сортировка посредством	Модифицированный	Быстрая сортировка	Структура Студент, содержит ФИО, курс,

	Простейший алгоритм 1	Простейший алгоритм 2	Быстрый алгоритм 3	Структура и критерий сортировки
	выбора	метод «пузырька»		факультет, сортировка по убыванию курса
5	Метод «пузырька»	Сортировка вставками	Пирамидальная сортировка	Структура Время, содержит часы, минуты, секунды, сортировка по возрастанию времени
6	Модифицированный метод «пузырька»	Сортировка посредством выбора	Сортировка Шелла	Структура Время, содержит часы, минуты, секунды, сортировка по убыванию времени
7	Сортировка вставками	Метод «пузырька»	«Быстрая» сортировка	Структура Сотрудник, содержит ФИО, профессия, ЗП. Сортировка по возрастанию ЗП.
8	Сортировка посредством выбора	Модифицированный метод «пузырька»	Сортировка слиянием	Структура Сотрудник, содержит ФИО, профессия, ЗП. Сортировка по убыванию ЗП.
9	Метод «пузырька»	Сортировка вставками	Сортировка Шелла	Структура студент, содержит ФИО, группа. Сортировка по ФИО.
10	Модифицированный метод «пузырька»	Сортировка посредством выбора	Пирамидальная сортировка	Структура дробь, содержит числитель и знаменатель, сортировка по возрастанию дробей.
11	Сортировка вставками	Метод «пузырька»	Сортировка Шелла	Структура дробь, содержит числитель и знаменатель, сортировка по убыванию дробей.
12	Сортировка посредством выбора	Модифицированный метод «пузырька»	«Быстрая» сортировка	Структура Продукт, содержит название и стоимость, сортировка по возрастанию стоимости.
13	Метод «пузырька»	Сортировка вставками	Сортировка слиянием	Структура Продукт, содержит название и стоимость, сортировка по убыванию стоимости.

	Простейший алгоритм 1	Простейший алгоритм 2	Быстрый алгоритм 3	Структура и критерий сортировки
14	Модифицированный метод «пузырька»	Сортировка посредством выбора	Быстрая сортировка	Структура Продукт, содержит название и стоимость, сортировка по названию.
15	Сортировка вставками	Метод «пузырька»	Пирамидальная сортировка	Структура Товар, содержит название, стоимость и код, сортировка по возрастанию кода.
16	Сортировка посредством выбора	Модифицированный метод «пузырька»	Сортировка Шелла	Структура Товар, содержит название, стоимость и код, сортировка по убыванию кода.
17	Метод «пузырька»	Сортировка вставками	«Быстрая» сортировка	Структура Товар, содержит название, стоимость и код, сортировка по возрастанию стоимости.
18	Модифицированный метод «пузырька»	Сортировка посредством выбора	Сортировка слиянием	Структура Товар, содержит название, стоимость и код, сортировка по убыванию стоимости.
19	Сортировка вставками	Метод «пузырька»	Сортировка Шелла	Структура заказ, содержит день, месяц, год, а также часы и минуты заказа, сортировка по возрастанию времени заказа.
20	Сортировка посредством выбора	Модифицированный метод «пузырька»	Пирамидальная сортировка	Структура заказ, содержит день, месяц, год, а также часы и минуты заказа, сортировка по убыванию времени заказа.

5. СОДЕРЖАНИЕ ОТЧЁТА

1. Цель работы.
2. Вариант задания.
3. Текст программы, реализующей расчеты по соответствующему варианту.

4. Анализ результатов работы программы в виде таблицы результатов и графиков.

5. Развернутый вывод по работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для элементов каких множеств можно корректно поставить задачу сортировки?
2. Что такое стабильная сортировка?
3. Что такое беспорядок?
4. Какова сложность оптимального алгоритма сортировки в худшем случае?
5. Всегда ли существует решение задачи сортировки?
6. Единственно ли решение задачи сортировки?
7. Как работает встроенная сортировка std::sort