

Лабораторная работа №3 «Алгоритмы сортировки»

Цель работы: освоить основные алгоритмы сортировки данных. Проанализировать быстродействие алгоритмов.

Вводная часть (виды сортировок, используемых в работе):

Сортировка Шелла

Сортировка Шелла является довольно интересной модификацией алгоритма сортировки простыми вставками.

Рассмотрим следующий алгоритм сортировки массива $a[0]..a[15]$.

1. Вначале сортируем простыми вставками каждые 8 групп из 2-х элементов ($a[0]$, $a[8]$), ($a[1]$, $a[9]$), ... , ($a[7]$, $a[15]$).

2. Потом сортируем каждую из четырех групп по 4 элемента ($a[0]$, $a[4]$, $a[8]$, $a[12]$), ..., ($a[3]$, $a[7]$, $a[11]$, $a[15]$).

В нулевой группе будут элементы 4, 12, 13, 18, в первой - 3, 5, 8, 9 и т.п.

3. Далее сортируем 2 группы по 8 элементов, начиная с ($a[0]$, $a[2]$, $a[4]$, $a[6]$, $a[8]$, $a[10]$, $a[12]$, $a[14]$).

4. В конце сортируем вставками все 16 элементов.

Очевидно, лишь последняя сортировка необходима, чтобы расположить все элементы по своим местам. Так зачем нужны остальные ?

На самом деле они продвигают элементы максимально близко к соответствующим позициям, так что в последней стадии число перемещений будет весьма невелико.

Последовательность и так почти отсортирована. Ускорение подтверждено многочисленными исследованиями и на практике оказывается довольно существенным.

Единственной характеристикой сортировки Шелла является приращение - расстояние между сортируемыми элементами, в зависимости от прохода. В конце приращение всегда равно единице - метод завершается обычной сортировкой вставками, но именно последовательность приращений определяет рост эффективности.

Использованный в примере набор ..., 8, 4, 2, 1 - неплохой выбор, особенно, когда количество элементов - степень двойки. Однако гораздо лучший вариант предложил Р.Седжвик. Его последовательность имеет вид

$$\text{inc}[s] = \begin{cases} 9 \cdot 2^s - 9 \cdot 2^{s/2} + 1, & \text{если } s \text{ четно} \\ 8 \cdot 2^s - 6 \cdot 2^{(s+1)/2} + 1, & \text{если } s \text{ нечетно} \end{cases}$$

При использовании таких приращений среднее количество операций: $O(n^{7/6})$, в худшем случае - порядка $O(n^{4/3})$.

Обратим внимание на то, что последовательность вычисляется в порядке, противоположном используемому: $\text{inc}[0] = 1$, $\text{inc}[1] = 5$, ... Формула дает сначала меньшие числа, затем все большие и большие, в то время как расстояние между сортируемыми элементами, наоборот, должно уменьшаться. Поэтому массив приращений inc вычисляется перед запуском собственно сортировки до максимального расстояния между элементами, которое будет первым шагом в сортировке Шелла. Потом его значения используются в обратном порядке.

При использовании формулы Седжвика следует остановиться на значении $\text{inc}[s-1]$, если $3 \cdot \text{inc}[s] > \text{size}$.

```
int increment(long inc[], long size) {
    int p1, p2, p3, s;

    p1 = p2 = p3 = 1;
    s = -1;
    do {
        if (++s % 2) {
            inc[s] = 8*p1 - 6*p2 + 1;
        } else {
            inc[s] = 9*p1 - 9*p3 + 1;
            p2 *= 2;
            p3 *= 2;
        }
    } while (3*inc[s] <= size);
    return s;
}
```

```

    }

    p1 *= 2;
} while(3*inc[s] < size);

return s > 0 ? --s : 0;
}

void shellSort(int a[], long size) {
    long inc, i, j, seq[40];
    int s;

    // вычисление последовательности приращений
    s = increment(seq, size);
    while (s >= 0) {
        // сортировка вставками с инкрементами inc[]
        inc = seq[s--];

        for (i = inc; i < size; i++) {
            int temp = a[i];
            for (j = i-inc; (j >= 0) && (a[j] > temp); j -= inc)
                a[j+inc] = a[j];
            a[j+inc] = temp;
        }
    }
}

```

Часто вместо вычисления последовательности во время каждого запуска процедуры, ее значения рассчитывают заранее и записывают в таблицу, которой пользуются, выбирая начальное приращение по тому же правилу: начинаем с `inc[s-1]`, если `3*inc[s] > size`.

Сортировка пузырьком

Идея метода: шаг сортировки состоит в проходе снизу вверх по массиву. По пути просматриваются пары соседних элементов. Если элементы некоторой пары находятся в неправильном порядке, то меняем их местами.

После нулевого прохода по массиву "вверху" оказывается самый "легкий" элемент - отсюда аналогия с пузырьком. Следующий проход делается до второго сверху элемента, таким образом второй по величине элемент поднимается на правильную позицию...

Делаем проходы по все уменьшающейся нижней части массива до тех пор, пока в ней не останется только один элемент. На этом сортировка заканчивается, так как последовательность упорядочена по возрастанию.

Качественно улучшение алгоритма можно получить из следующего наблюдения. Хотя легкий пузырек снизу поднимется наверх за один проход, тяжелые пузырьки опускаются со минимальной скоростью: один шаг за итерацию. Так что массив 2 3 4 5 6 1 будет отсортирован за 1 проход, а сортировка последовательности 6 1 2 3 4 5 потребует 5 проходов.

Чтобы избежать подобного эффекта, можно менять направление следующих один за другим проходов. Получившийся алгоритм иногда называют "**шейкер-сортировкой**".

Сортировка вставками

Сортировка простыми вставками в чем-то похожа на вышеизложенные методы.

Аналогичным образом делаются проходы по части массива, и аналогичным же образом в его начале "вырастает" отсортированная последовательность...

Однако в сортировке пузырьком или выбором можно было четко заявить, что на i -м шаге элементы $a[0]...a[i]$ стоят на правильных местах и никуда более не переместятся. Здесь же подобное утверждение будет более слабым: последовательность $a[0]...a[i]$ упорядочена. При этом по ходу алгоритма в нее будут вставляться (см. название метода) все новые элементы.

Будем разбирать алгоритм, рассматривая его действия на i -м шаге. Как говорилось выше, последовательность к этому моменту разделена на две части: готовую $a[0]...a[i]$ и неупорядоченную $a[i+1]...a[n]$.

На следующем, $(i+1)$ -м каждом шаге алгоритма берем $a[i+1]$ и вставляем на нужное место в готовую часть массива.

Поиск подходящего места для очередного элемента входной последовательности осуществляется путем последовательных сравнений с элементом, стоящим перед ним.

В зависимости от результата сравнения элемент либо остается на текущем месте (вставка завершена), либо они меняются местами и процесс повторяется.

Сортировка вставками со сторожевым элементом

Алгоритм можно слегка улучшить. Заметим, что на каждом шаге внутреннего цикла проверяются 2 условия. Можно объединить их в одно, поставив в начало массива специальный сторожевой элемент. Он должен быть заведомо меньше всех остальных элементов массива.

Тогда при $j=0$ будет заведомо верно $a[0] \leq x$. Цикл остановится на нулевом элементе, что и было целью условия $j \geq 0$.

Таким образом, сортировка будет происходить правильным образом, а во внутреннем цикле станет на одно сравнение меньше. Однако, отсортированный массив будет не полон, так как из него исчезло первое число. Для окончания сортировки это число следует вернуть назад, а затем вставить в отсортированную последовательность $a[1] \dots a[n]$.

Сортировка выбором

Идея метода состоит в том, чтобы создавать отсортированную последовательность путем присоединения к ней одного элемента за другим в правильном порядке.

Будем строить готовую последовательность, начиная с левого конца массива. Алгоритм состоит из n последовательных шагов, начиная от нулевого и заканчивая $(n-1)$ -м.

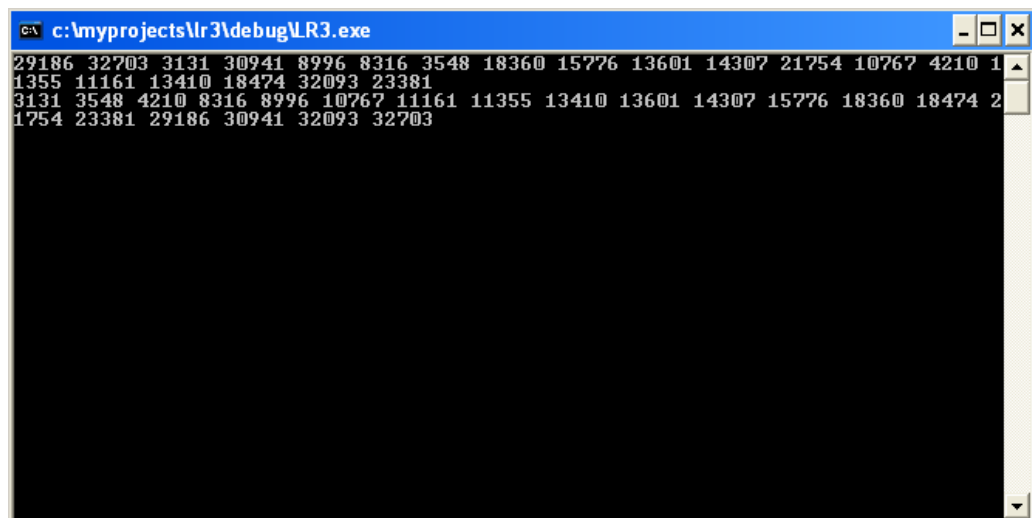
На i -м шаге выбираем наименьший из элементов $a[i] \dots a[n]$ и меняем его местами с $a[i]$.

Вне зависимости от номера текущего шага i , последовательность $a[0] \dots a[i]$ (выделена курсивом) является упорядоченной. Таким образом, на $(n-1)$ -м шаге вся

последовательность, кроме $a[n]$ оказывается отсортированной, а $a[n]$ стоит на последнем месте по праву: все меньшие элементы уже ушли влево.

Ход работы:

1. Создайте новый консольный проект (см л.р. 1).
2. Добавьте в проект сpp-файл, в котором будет располагаться функция `main()`.
3. Для удобства будем сортировать последовательности целых положительных чисел. Создайте функцию `Generate`, которая будет создавать массив заданного количества произвольных чисел от 0 до 32767, расположенных в произвольном порядке. Для генерации чисел можно воспользоваться функцией `int rand(void)` из библиотеки `stdlib.h`. Эта функция возвращает псевдослучайное число из диапазона 0 – 32767. Перед стартом цикла генерации чисел необходимо инициализировать систему псевдослучайных чисел. Сделать это можно вызовом функции `srand(GetTickCount());`
Эта функция берет текущее системное время и инициализирует им систему псевдослучайных чисел. Такая инициализация снижает вероятность повторения последовательностей.
Для работы функции `GetTickCount` необходимо подключить также библиотеку `windows.h`.
4. Создайте функцию `Print`, которая будет выводить на экран в строчку через пробел все числа заданной последовательности.
5. Реализуйте функцию сортировки Шелла, приведенную в приложении.
6. Составьте функцию `main` так, чтобы генерировалось 20 случайных чисел, затем они выводились на экран, сортировались и снова выводились на экран. Убедитесь, что сортировка работает:



```
c:\myprojects\lr3\debug\LR3.exe
29186 32703 3131 30941 8996 8316 3548 18360 15776 13601 14307 21754 10767 4210 1
1355 11161 13410 18474 32093 23381
3131 3548 4210 8316 8996 10767 11161 11355 13410 13601 14307 15776 18360 18474 2
1754 23381 29186 30941 32093 32703
```

7. Замерьте время, требуемое для сортировки. Для этого можно воспользоваться уже знакомой функцией `GetTickCount()` по следующему принципу:

```
DWORD t1, t2;  
t1 = GetTickCount();  
// ... Делаем что-то  
t2 = GetTickCount();
```

Разница $t2-t1$ даст время процесса в миллисекундах.

8. Результатом замера скорей всего будет 0, т.к. чисел очень мало для того, чтобы сортировка заняла сколько-нибудь ощутимое время. Количество сортируемых чисел необходимо увеличить. Сделайте замеры для массива из 10000000 элементов. Обратите внимание, что, если массив задается статически в стиле `int m[10000000]`, то скорей всего произойдет ошибка переполнения стека. Для того, чтобы можно было работать с большим количеством чисел, воспользуемся динамическим выделением:

```
int *m;  
m = new int[10000000];  
// ... Делаем что-то  
delete [] m;
```

9. Сделайте 10 замеров для количества 10000000 чисел. Посчитайте среднее время работы алгоритма сортировки. Замеры и результаты расчета отразите в отчете.
10. Сделайте такие же замеры для различного числа элементов так, чтобы по полученным данным можно было получить 5-7 точек для построения графика зависимости времени работы от количества данных. В отчете приведите таблицу с данными о замерах, о среднем времени и график зависимости.
11. Реализуйте алгоритм из индивидуального задания. Описания алгоритмов даны в приложении.
12. Проведите все необходимые замеры, приведите в отчет таблицы замеров и график на котором отражены зависимости для алгоритма Шелла и вашего алгоритма.

Индивидуальные задания:

1. Сортировка пузырьком.
2. Сортировка вставками.
3. Сортировка выбором.
4. Сортировка вставками со сторожевым элементом.
5. Шейкер-сортировка.