Санкт-Петербургский политехнический университет Петра Великого

К У Р С О В А Я Р А Б О Т А

Insertion Sort

по дисциплине «Программирование на языках высокого уровня»

Выполнил
студент гр. 33335/2 В.В. Бурин

Преподаватель М.С. Ананьевский

«___» _____ 2018 г.

Санкт-Петербург

2018

1. Введение

Пусть существует N элементов (N_1, N_2, …, N_n). Каждый i-ый элемент содержит некоторую информацию и ключ K_i, который и будет управлять процессом сортировки. Необходимо расположить элементы так, чтобы их ключи располагались в порядке возрастания.

$$K_i \leq K_{i+1} \leq \cdots \leq K_n$$

Существует множество различных алгоритмов сортировки. В данной курсовой работе рассмотрен алгоритм сортировки вставками.

При изучении алгоритма использовалась следующая литература:

- Дональд Э. Кнут, «Искусство программирования, том 3. Сортировка и поиск» [1];
- Макконнелл Дж. «Основы современных алгоритмов» [2].

2. Описание алгоритма

2.1. Сортировка простыми вставками

Суть сортировки вставками:

- Существует две части массива: отсортированная и неотсортированная.
- Из неотсортированной части берется любой элемент и вставляется на своё место в отсортированной части массива, таким образом отсортированная часть массива увеличивается, а неотсортированная уменьшается.

Псевдокод для метода сортировки простыми вставками (сортируемый массив имеет размер SIZE, нумерация элементов начинается с 0) выглядит следующим образом:

```
for i = 1 to (SIZE-1) do
                tmp = A[i];
                j=i-1;
    while (j >= 0 and A[j] > tmp) do
                A[j + 1] = A[j];
                A[j]=tmp;
                j = j – 1;
          end while
end for
```

Данный метод сравнивает рассматриваемый элемент с элементами отсортированной части. Если рассматриваемый элемент меньше какого-либо элемента в отсортированной части, то они меняются местами, если рассматриваемый элемент больше, то цикл обрывается, т.к. нет смысла проверять остальные элементы.

Существуют улучшенные методы сортировок вставками.

2.2. Сортировка вставками с бинарный поиском

Т.к. необходимо искать место для элемента в уже отсортированной части массива, то можно использовать бинарный поиск. То есть сначала элемент сравнивается с серединой отсортированного массива, далее рассматривается только та часть, к которой элемент принадлежит. Это сократит кол-во сравнений, но не кол-во сдвигов элементов.

2.3. Парная сортировка вставками

Вместо одного рассматриваемого элемента, можно рассматривать сразу два элемента. Изначально больший из двух элементов, вставляется в отсортированный массив, далее меньший элемент вставляется в отсортированную часть массива, но сравнения начинаются с номера, на который встал больший элемент. То есть кол-во сравнений для меньшего элемента будет меньше, чем при сортировке простыми вставками.

2.4. Сортировка Шелла

В сортировке Шелла нет отсортированной и неотсортированной части. Изначально выбирается некоторая разность *d*. Далее сортируется подмассивы, которые состоят из элементов, номера которых отличаются на *d*. После сортировки подмассивов, их элементы ставятся на те же места в исходный массив, откуда эти элементы были взяты, но уже в отсортированном порядке

Так происходит пока все элементы не будут рассмотрены. Далее *d* уменьшается и действия повторяются. Когда *d=1* сортировка станет обычной сортировкой вставками.

Смысл метода в том, что с помощью такого разбиения на мелкие подмассивы, многие элементы уже встанут на свои места и соответственно в последующих проходах кол-во сдвигов и сравнений элементов будет меньше.

3. Реализация алгоритма

Реализация на C:

```
for(unsigned int i=1; i<Size; i++)
   {
      int j = i-1;
      while (j >= 0 && Array[j] > Array[j + 1])
      {
         Array[j + 1]= Array[j + 1]^Array[j];
         Array[j] = Array[j + 1]^Array[j];
         Array[j + 1]= Array[j + 1]^Array[j];
         j--;
      }
   }
```

Во внешнем цикле определяется рассматриваемый элемент.

Во внутреннем цикле рассматриваемый элемент сравнивается с меньшими по номеру элементами (с элементами отсортированного массива) и вставляется на своё место.

4. Анализ

Проанализировав кол-во операций псевдокода и их кол-во повторений, сделан вывод, что наиболее частая операция — это операция сравнения **while** (j >= 0 and A[j] > key). Поэтому на её основе будут рассмотрены лучший, худший и средний случаи.

4.1. Средний случай

Рассмотрен средний случай. i-ый элемент имеет возможность встать на одно из i+1 положений. Предположено, что все случаи равновероятны.

Элемент может остаться на своей позиции, тогда произойдет будет 1 сравнение, если он встанет на i-1, произойдёт 2 сравнения и т.д. То есть если элемент должен встать на 2-ую позицию, то нужно i сравнений, и также i сравнений для попадания на 1 элемент.

Кол-во сравнений i-го элемента будет равно

$$\sum_{k=1}^{i} i + i = \frac{i(i+1)}{2} + i$$

Умножаем на вероятность встать на какую-либо позицию:

$$\frac{1}{i+1}\left(\frac{i(i+1)}{2} + i\right) = \frac{i}{2} + \frac{i}{i+1} = \frac{i}{2} + 1 - \frac{1}{i+1}$$

Получено среднее кол-во сравнений для i-го элемента.

Тогда для всех элементов кол-во сравнений будет равно:

$$\sum_{1}^{n-1}\left(\frac{i}{2} + 1 - \frac{1}{i+1}\right) = \sum_{1}^{n-1}\frac{i}{2} + \sum_{1}^{n-1} 1 - \sum_{1}^{n-1}\frac{1}{i+1} = \sum_{1}^{n-1}\frac{i}{2} + \sum_{1}^{n-1} 1 - \sum_{2}^{n}\frac{1}{i}$$

$$= \sum_{1}^{n-1}\frac{i}{2} + \sum_{1}^{n-1} 1 - \sum_{1}^{n}\frac{1}{i} - 1 = \frac{n^2 - n}{4} + (n-1) - \ln(n) + 1$$

Тогда временная сложность:

$$T(n) \approx \frac{n^2 - n}{4} + (n-1) - \ln(n) + 1 = O(n^2)$$

Видно, что время зависит нелинейно.

Для того, чтобы убедиться, что сложность $O(n^2)$, написана программа, которая считает кол-во входов во внутренний цикл (кол-во сравнений) функции сортировки при различных N. Массив заполняется случайными числами от -99 до 99.

Результат показан на рис.1.

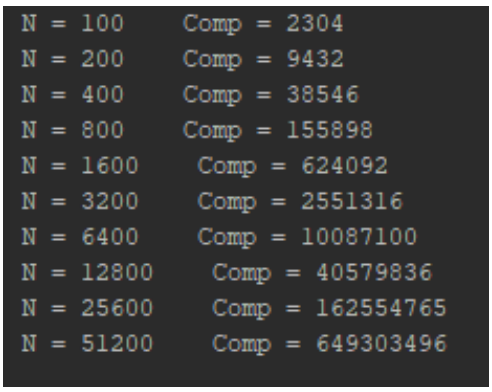

Рис. 1 — Зависимость кол-ва сравнений от размера данных

Показанные результаты подтверждают, что зависимость кол-ва сравнений от размера данных нелинейна.

4.2. Худший случай

Наихудшим является случай, когда массив отсортирован в обратном порядке. При этом каждый новый элемент сравнивается со всеми элементами отсортированной части массива. То есть, i-ый элемент сделает i сравнений. Тогда общее кол-во сравнений:

$$\sum_{1}^{n-1} i = \frac{1+n-1}{2} \cdot (n-1) = \frac{n \cdot (n-1)}{2}$$

Следовательно:

$$T(n) \approx \frac{n^2 - n}{2} = O(n^2)$$

4.3. Лучший случай

Лучшим является случай, когда массив уже отсортирован. При таком исходе каждый элемент должен сделать только одно сравнение.

Тогда общее кол-во сравнений:

$$\sum_{1}^{n-1} 1 = n - 1$$

Следовательно:

$$T(n) \approx n - 1 = O(n)$$

4.4. Время работы алгоритма сортировки

С помощью реализации алгоритма (3 пункт) найдено время работы функции при разном размере n массива случайных чисел от -99 до 99. Результаты показаны на рис. 2.

Количество элементов	Время, сек
n :=	T :=
10000	1
20000	2
30000	3
40000	7
50000	9
60000	14
70000	19
80000	25
90000	33
100000	39

Рис. 2 — Время работы алгоритма при разном наборе данных

Аппроксимируя полученные результаты, построен график зависимости времени от кол-ва элементов (рис. 3).

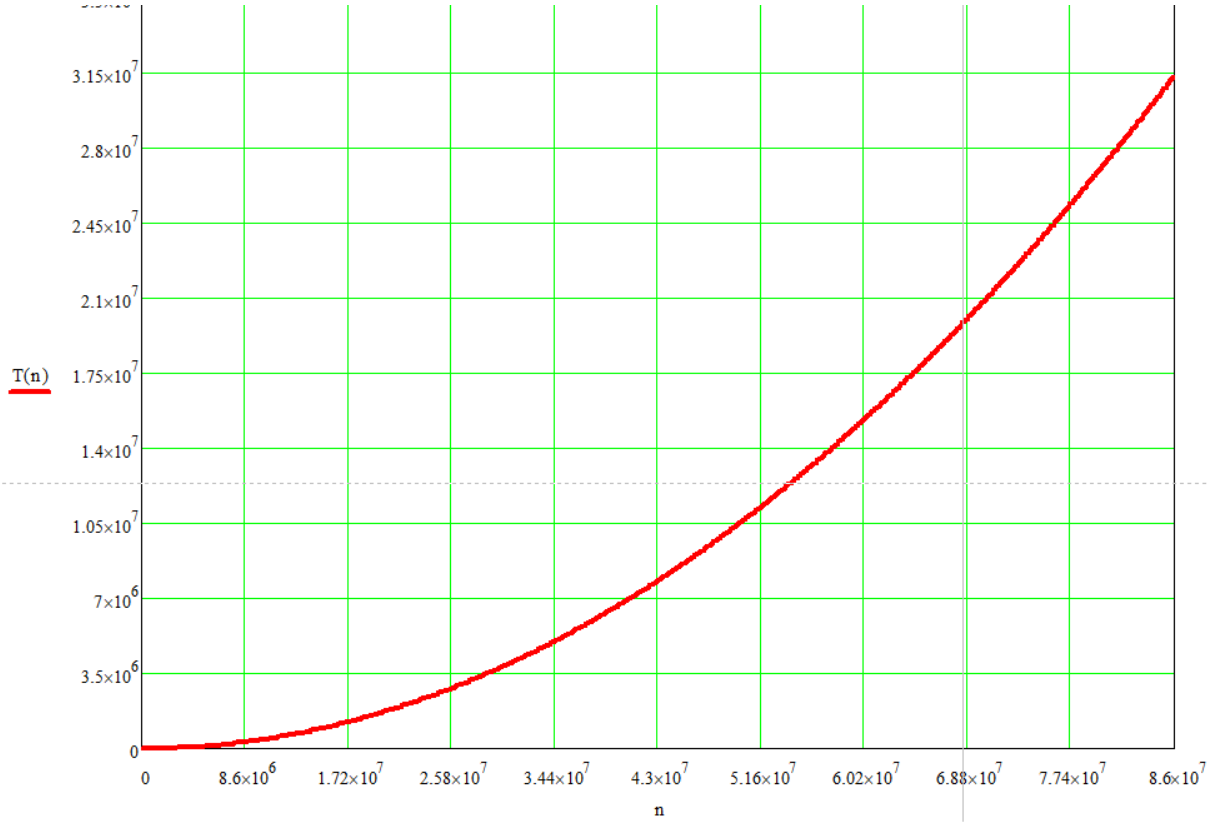

Рис. 3 — График зависимости T(n)

С помощью полученного графика найдено кол-во данных, при которых время сортировки равно:

T(n)=1 час при **n ≈ 950 000**;

T(n)=1 сутки при **n ≈ 4 500 000**;

T(n)=1 месяц при **n ≈ 25 000 000**;

T(n)=1 год при **n ≈ 85 000 000**.

5. Применение алгоритма

Сортировка простыми вставками уступает по временной сложности некоторым другим алгоритмам сортировки, поэтому её применение достаточно редко.

Её имеет смысл применять, когда массив уже частично отсортирован т.к. это сокращает кол-во сдвигов элементов, либо же для массивов малого размера. Именно поэтому сортировка вставками используется в функции сортировки qsort() стандартной библиотеки stdlib при малых размерах сортируемого массива, т.к. сортировка вставками работает быстрее для массивов малого размера нежели быстрая сортировка.

Список литературы

1. Дональд Э. Кнут, «Искусство программирования, том 3. Сортировка и поиск», М., 2018. – 824 с.
2. Макконнелл Дж. «Основы современных алгоритмов». М.: Техносфера, 2004. - 368 с