Список литературы

1. ***Н.Вирт***. «Алгоритмы и структуры данных», 1989.

2. ***Д.Кнут***. «Искусство программирования для ЭВМ», том 1 и 3, 1976-78.

3. ***Т.Кормен, Ч.Лейзерсон, Р.Ривест***. «Алгоритмы: построение и анализ», 2001,2004,2009,2011.

4. ***Р.Седжвик***. «Фундаментальные алгоритмы на С++», 2002.

5. ***Е.В.Курапова, Е.П.Мачикина***. «Структуры и алгоритмы обработки данных», 2006.

6. ***Е.В.Курапова, Е.П.Мачикина***. «Основные методы кодирования данных», 2010.

Никлаус Вирт

День рождения 15 февраля 1934

Швейцарский учёный,
специалист в области информатики,
известный теоретик в области разработки языков программирования,
профессор компьютерных наук.
Участвовал в разработке языков ALGOL68 и PL/360
Разработал языки Паскаль и Модула-2

Дональд Кнут

День рождения 10 января 1938

Американский учёный,

почётный профессор университетов в разных странах, иностранный член Российской академии наук,

преподаватель и идеолог программирования,

автор 19 монографий

Метод прямого выбора SelectSort

Находим наименьший элемент массива и переставляем его на первое место.

Среди оставшихся элементов (начиная со второго) находим наименьший элемент и переставляем его на второе место в массиве.

Среди оставшихся элементов (начиная с третьего) находим наименьший элемент и переставляем его на третье место

и т. д. **сколько раз???**

Метод прямого выбора

Алгоритм на псевдокоде

```
DO ( i := 1, 2, ... n-1)
    k := i
    DO  ( j := i+1, i+2,… ,n )
            IF ( a_j < a_k )  k := j   FI
      OD
      a_i <--> a_k
OD
```

К У Р А П О В

А

А У Р К П О В

А

А А Р К П О В

У

А А В К П О Р

У

А А В К П О Р

У

Трудоемкость метода SelectSort

Дадим оценку трудоёмкости метода прямого выбора, т.е. определим количество пересылок и сравнений.

1) *По количеству пересылок*: на каждой итерации старшего цикла выполняется один обмен (3 пересылки).

$$M = 3(n-1)$$

2) *По количеству сравнений* можем сказать:

когда i=1 требуется (n-1) сравнение,

когда i=2 требуется (n-2) сравнения, и т.д.

Суммируя, получим:

$$
\begin{array}{rl}
+ & C = (n-1) + (n-2) + (n-3) + \ldots + 1 \\
 & C = 1 + 2 + 3 + \ldots + (n-1) \\
\hline
 & 2C = n + n + n + \ldots + n
\end{array}
$$

$$2C = n(n-1)$$

$$C = \frac{n^2 - n}{2}$$

При подсчете трудоемкости учитываются ***только те операции***, в которых участвуют ***элементы массива***.

Для удобства реализации алгоритмов на Си массив можно описывать следующим образом:

int A [1+n];

Тогда при заполнении и выводе массива элемент **A[0]** не используется.

<u>Для метода прямого выбора SelectSort</u>:

- Значения M и C *не зависят* от исходной упорядоченности массива.
- Сортировка *не устойчива*.

Пример:

3’ 3” 2 -> 2 3” 3’

Видео SelectSort

Классы сложности алгоритмов

Часто бывает трудно определить точное время работы алгоритма, тогда пользуются асимптотической или приближенной оценкой времени работы.

Асимптотическое доминирование функций.

<u>Определение</u>.

Функция ***f(x)*** асимптотически доминирует над функцией ***g(x)*** или $g(x) = O(f(x))$,

если $|g(x)| \leq const\ |f(x)|$ при $x \to \infty$.

Примеры:

1) $g(x) = 10x \quad f(x) = x$
2) $g(x) = 1 \quad f(x) = x$
3) $g(x) = 2x \quad f(x) = x^2$

Свойства асимптотического доминирования функций

Для функций f, f_1, f_2 и константы k справедливы **свойства**:

1. $f = O(f)$
2. $O(k*f) = O(f)$
3. $O(f_1+f_2) = O(f_1) + O(f_2)$

<u>Пример</u>: $T = 10n + 20$

$T = O(10n+20) = O(10n) + O(20) = O(n) + O(1) = O(n)$,

при $n \to \infty$.

Приведем ряд доминирования основных функций

$\mathbf{O(1) < O(\log n) < O(n) < O(n \log n) < O(n^a) < O(a^n) <}$

$\mathbf{< O(n!) < O(n^n)}$, при $n \to \infty$, $a > 1$.

Трудоемкость SelectSort

- $$\mathbf{M = 3(n-1)}$$

$$\mathbf{C = \frac{n^2 - n}{2}}$$

$$T = C + M = O(T) = O(C+M) = O(\frac{n^2 - n}{2} + 3(n-1)) =$$

$$= O(\frac{n^2}{2}) - O(\frac{n}{2}) + O(3n) - O(3) =$$

$$= O(n^2) - O(n) + O(n) - O(1) = O(n^2)\ ,\ n \to \infty$$

$$\boxed{\mathbf{T = O(n^2)}\ ,\ n \to \infty}$$

Метод	Трудоемкость	Устойчивость	Зависимость от упорядоченности
SelectSort	$O(n^2)$	Не устойчив	Не зависит

Пузырьковая сортировка BubbleSort

Двигаясь от конца массива к его началу, будем сравнивать между собой соседние элементы. Если правый элемент будет меньше, чем левый, то поменяем их местами.

При первом проходе наименьший элемент переместится на первое место. При втором проходе наименьший элемент из оставшихся “всплывёт” на второе место, и т.д.

Через (n-1) итерацию массив будет отсортирован.

Пузырьковая сортировка

Алгоритм на псевдокоде

Обозначим i – номер итерации,

j – индекс правого элемента в текущей паре.

DO (i := 1, 2, ... n-1)

DO (j := n, n-1, ... i+1)

IF ($a_j < a_{j-1}$) $a_j \leftrightarrow a_{j-1}$ FI

OD

OD

К У Р А П О В А

А В

А О

А П

А А

А Р

А У

А К

А К У Р А П О В

В О

В П

А В

А Р

А У

А К

А А К У Р В П О

О П

В О

В Р

В У

В К

А А В К У Р О П

О П

О Р

О У

К О

А А В К О У Р П

П Р

А А В К О П У Р

Р У

П Р

А А В К О П Р У

Р У

А А В К О П Р У

А А В К О П Р У

Если сравнить **BubbleSort** с алгоритмом **SelectSort**, то по количеству сравнений эти методы идентичны.

$$C = \frac{n^2 - n}{2}$$

Количество пересылок **M** зависит от исходной упорядоченности массива. Рассмотрим случаи:

1) Массив упорядочен -> пересылок не будет ($M_{min} = 0$)
2) Массив обратно упорядочен -> максимальное количество пересылок ($M_{max} = 3C$)
3) Массив случайный $M_{ср} = \frac{3C}{2} = \frac{3(n^2 - n)}{4}$

Метод зависит от исходной упорядоченности по количеству пересылок.

Пузырьковая сортировка устойчива.

Метод	Трудоемкость	Устойчивость	Зависимость от упорядоченности
SelectSort	$O(n^2)$	Не устойчив	Не зависит
BubbleSort	$O(n^2)$	Устойчив	Зависит