Иванков Евгений Александрович

**Лабораторная работа №1.**

**Оценка временной сложности алгоритмов**

Целью лабораторной работы является приобретение навыков исследования временной сложности алгоритмов и определения ее асимптотических оценок.

* + 1. Требования к содержанию, оформлению и порядку выполнения

В содержательной части отчета по выполнению лабораторной работы требуется привести описание алгоритма, выбранного согласно своему варианту, провести его анализ и определить асимптотические оценки его временной сложности. Алгоритм рекомендуется оформлять с помощью блок-схем.

* + 1. Теоретическая часть

Теоретические сведения, необходимые для выполнения лабораторной работы, представлены в лекции.

* + 1. Общая постановка задачи

Требуется провести анализ и оценку временной сложности заданного алгоритма. Варианты заданий представлены в таблице в следующем разделе.

В качестве дополнительных заданий рекомендуется программно реализовать заданный алгоритм.

* + 1. Список индивидуальных данных

Данные для выполнения лабораторной работы сведены в табл.Л2.1.

Таблица Л2.1.

Варианты заданий к лабораторной работе № 2

|  |  |
| --- | --- |
| *Вариант* | *Алгоритм* |
| 1 | Тривиальный алгоритм возведения в степень (рис. 3.1) |
| 2 | Рекурсивный алгоритм возведения в степень (рис. 3.2) |
| 3 | Алгоритм быстрого возведения в степень (рис. 3.3 а) |
| 4 | Алгоритм быстрого возведения в степень (рис. 3.3 б) |
| 5 | Алгоритм вычисления значения многочлена (рис. 3.4) |
| 6 | Алгоритм вычисления значения многочлена по схеме Горнера (рис. 3.5) |
| 7 | Алгоритм сортировки обменом (рис. 4.5) |
| 8 | Алгоритм сортировки выбором (рис. 4.7) |
| 9 | Алгоритм сортировки вставками (рис. 4.9) |
| 10 | Алгоритм быстрой сортировки (рис. 4.14, 4.16) |

* + 1. Выполнения работы (ход работы)

Требуется провести анализ и оценку временной сложности алгоритма быстрого возведения в степень.

### Общий механизм сортировки

Быстрая сортировка относится к алгоритмам «разделяй и властвуй».

Алгоритм состоит из трёх шагов:

1. Выбрать элемент из массива. Назовём его опорным.
2. *Разбиение*: перераспределение элементов в массиве таким образом, что элементы меньше опорного помещаются перед ним, а больше или равные после.
3. Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором только один элемент или отсутствуют элементы.

В наиболее общем виде алгоритм на псевдокоде (где A — сортируемый массив, а low и high — соответственно, нижняя и верхняя границы сортируемого участка этого массива) выглядит следующим образом:.

**algorithm** quicksort(A, low, high) **is**

**if** low < high **then**

p := partition(A, low, high)

quicksort(A, low, p - 1)

quicksort(A, p + 1, high)

Здесь предполагается, что массив A передаётся по ссылке, то есть сортировка происходит «на том же месте», а неописанная функция partition возвращает индекс опорного элемента.

Для выбора опорного элемента и операции разбиения существуют разные подходы, влияющие на производительность алгоритма.

### Разбиение Ломуто

Данный алгоритм разбиения был предложен Нико Ломуто[3] и популяризован в книгах Бентли (Programming Pearls) и Кормена (Введение в алгоритмы).[4] В данном примере опорным выбирается последний элемент. Алгоритм хранит индекс в переменной i. Каждый раз, когда находится элемент, меньше или равный опорному, индекс увеличивается, и элемент вставляется перед опорным. Хоть эта схема разбиения проще и компактнее, чем схема Хоара, она менее эффективна и используется в обучающих материалах. Сложность данной быстрой сортировки падает до *O*(*n*2), когда массив уже отсортирован или все его элементы равны. Существуют различные методы оптимизации данной сортировки: алгоритмы выбора опорного элемента, использование сортировки вставками на маленьких массивах. В данном примере сортируются элементы массива *A* от low до high (включительно)[4]:

**algorithm** partition(A, low, high) **is**

pivot := A[high]

i := low

**for** j := low **to** high - 1 **do**

**if** A[j] ≤ pivot **then**

swap A[i] with A[j]

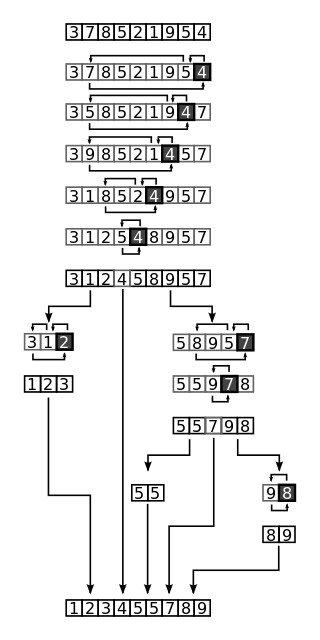
i := i + 1

swap A[i] with A[high]

**return** i

Сортировка всего массива может быть выполнена с помощью выполнения quicksort(A, 1, length(A)).

Пример быстрой сортировки. Здесь опорным является последний элемент массива (ячейка чёрного цвета), что в отсортированных массивах может приводить к ухудшению производительности.



Контрольные вопросы к защите

1) Временная сложность алгоритма (в худшем случае) — это функция от размера входных данных, равная максимальному количеству элементарных операций, проделываемых алгоритмом для решения экземпляра задачи указанного размера.

Аналогично понятию временной сложности в худшем случае определяется понятие временная сложность алгоритма в наилучшем случае. Также рассматривают понятие среднее время работы алгоритма, то есть математическое ожидание времени работы алгоритма. Иногда говорят просто: «Временная сложность алгоритма» или «Время работы алгоритма», имея в виду временную сложность алгоритма в худшем, наилучшем или среднем случае (в зависимости от контекста).

По аналогии с временной сложностью, определяют пространственную сложность алгоритма, только здесь говорят не о количестве элементарных операций, а об объёме используемой памяти.

2) Анализ сравнения затрат времени алгоритмов, выполняемых решение экземпляра некоторой задачи, при больших объемах входных данных, называется асимптотическим. Алгоритм, имеющий меньшую асимптотическую сложность, является наиболее эффективным.

3) Основные оценки роста, встречающиеся в асимптотическом анализе:

Ο (О-большое) – верхняя асимптотическая оценка роста временной функции;

Ω (Омега) – нижняя асимптотическая оценка роста временной функции;

Θ (Тета) – нижняя и верхняя асимптотические оценки роста временной функции.

4) Важные правила асимптотического анализа:

O(k\*f) = O(f) – постоянный множитель k (константа) отбрасывается, поскольку с ростом объема данных, его смысл теряется, например:

O(9,1n) = O(n)

O(f\*g) = O(f)\*O(g) – оценка сложности произведения двух функций равна произведению их сложностей, например:

O(5n\*n) = O(5n)\*O(n) = O(n)\*O(n) = O(n\*n) = O(n2)

O(f/g)=O(f)/O(g) – оценка сложности частного двух функций равна частному их сложностей, например:

O(5n/n) = O(5n)/O(n) = O(n)/O(n) = O(n/n) = O(1)

O(f+g) равна доминанте O(f) и O(g) – оценка сложности суммы функций определяется как оценка сложности доминанты первого и второго слагаемых, например:

O(n5+n10) = O(n10)