Аброськина

**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.

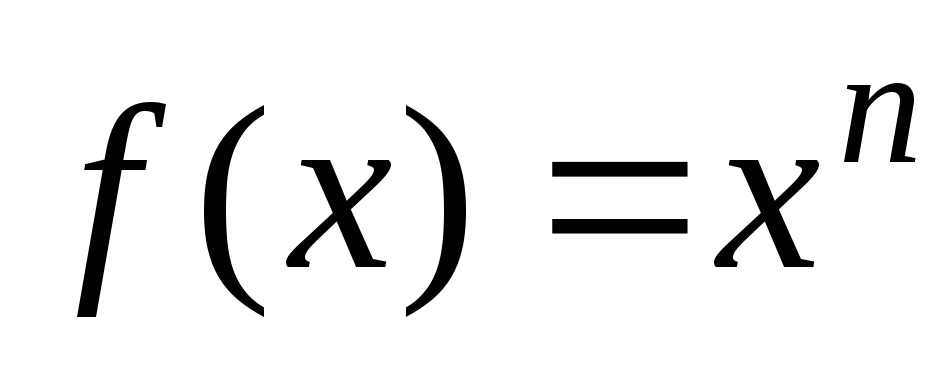
Таблица Л2.1.

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) |

Ход работы

Рассмотрим 3 вариант. Требуется провести анализ и оценку временной сложности алгоритма быстрого возведения в степень.

Алгоритм предназначен для решения следующей задачи: дано число *x*и натуральное (целое неотрицательное) число *n ≥ 0*. Вычислить значение функции

Двоичное представление степени n можно представить в виде:

, то есть 

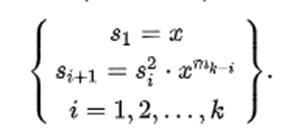
где 

Тогда:



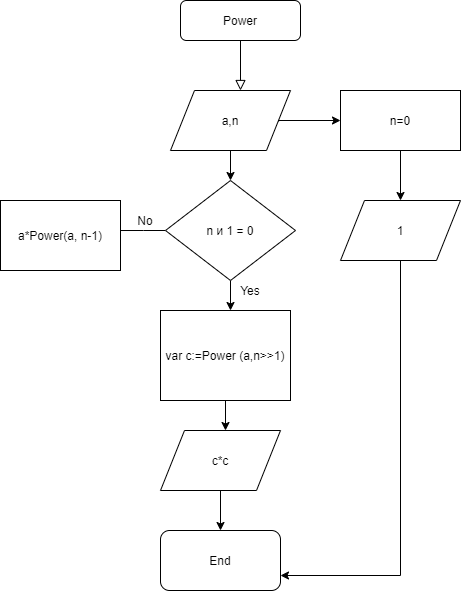
Последовательность действий при использовании схемы можно описать следующим образом. Представить показатель степени n в двоичном виде. Если = 1, то текущий результат возводится в квадрат и затем умножается на x. Если = 0, то текущий результат просто возводится в квадрат. Индекс i изменяется от k-1 до 0.

Таким образом, алгоритм быстрого возведения в степень сводится к мультипликативному аналогу схемы Горнера:



Алгоритм быстрого возведения в степень основан на выше приведенных соображениях.

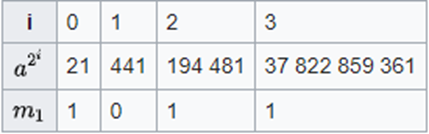
Блок – схема алгоритма:



Проведем анализ. Сначала отметим, что, степень, в которую возводится число, может служить мерой объема входных данных и все операторы присваивания и сравнения имеют некоторое постоянное время выполнения, независящее от размера входных данных. Временную составляет то, что данный алгоритм является рекурсивным, т.к в

некоторых блоках мы вызываем нашу функцию Power.

Пример. Посчитаем с помощью схемы возведения в степень «справа налево» значение 21 в 13 степени.



21 · 194 481 = 4084 101

1.4084 101 · 37 822 859 361 = 154 472 377 739 119 461

Всего получается m+2 итераций, где m определяется соотношением . Из последнего соотношения следует: – ближайшее целое не превосходящее log2 n.. Как итог, сложностью алгоритма является рекурсивное обращение к нему. Время на выполнения данного алгоритма больше всего тратят операторы условия, которые заново вызывают данную операцию умножения, но затраты на время минимальны. Согласно правилу сумм, общая временная сложность алгоритма T(n) = О(max(1,1,log n,1)) = О(log n).

Следует отметить, что в этом алгоритме имеет место ситуация: T(n) =  (log n), и следовательно T(n) =  (log n).

**Вывод:** В лабораторной работе проведен анализ алгоритма быстрого возведения в степень. имеет верхнюю оценку О(log n). Нижняя оценка составляет  (log n), следовательно, имеет место оценка  (log n). Данный алгоритм является рекурсивным, он имеет лучшую оценку временной сложности, который, "на вскидку" имеет временную сложность порядка О(n).

Контрольные вопросы к защите

1. Понятие временной сложности алгоритма.
2. Определение асимптотических оценок временной сложности.
3. Основные принципы получения асимптотических оценок.
4. *Правила анализа алгоритмов с целью определения их временной сложности.*

Ответы на контрольные вопросы

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.

1. 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.
2. 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.

1. 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)