Практическое занятие №1.1: «Оценка вычислительной сложности алгоритма»

Цель: приобретение практических навыков:

- эмпирическому определению вычислительной сложности алгоритмов на теоретическом и практическом уровнях;
- выбору эффективного алгоритма решения вычислительной задачи из нескольких.

Задание 1. Выбрать эффективный алгоритм вычислительной задачи из двух предложенных, используя теоретическую и практическую оценку вычислительной сложности каждого из алгоритмов, а также его ёмкостную сложность.

Пусть имеется вычислительная задача:

– дан массив **x** из **n** элементов целого типа; удалить из этого массива все значения равные заданному (ключевому) **key**.

Удаление состоит в уменьшении размера массива с сохранением порядка следования всех элементов, как до, так и следующих после удаляемого.

Например, необходимо удалить из массива все значения равные 2.

Исходный массив (n=10): 1 2 3 2 2 2 5 2 2 2. Результат: n=3; 1 3 5.

Можно предложить два подхода к решению данной задачи, т.е. два алгоритма. Они представлены в табл. 1. Необходимо реализовать эти алгоритмы, оценить их вычислительную сложность теоретически и практически и сделать вывод об их эффективности.

Таблица 1. Два алгоритма решения задачи

<table>
<tr><td colspan="2">x-массив, n – количество элементов в массиве, key – удаляемое значение</td></tr>
<tr><td>Алгоритм 1:
delFirstMetod(x,n,key) {
i ← 1
while (i<=n) do
    if x[i]=key then
        //удаление
        for j ← i to n-1 do
         x[j] ← x[j+1]
        od
        n ← n-1
   else
      i ← i+1
   endif
od
}</td><td>Алгоритм 2:
delOtherMetod(x,n,key) {
j ← 1
for i ← 1 to n do
  x[j] ← x[i];
  if x[i] != key then
    j++
  endif
od
n ← j
}</td></tr>
</table>

Требования к выполнению задания 1:

1. Для каждого алгоритма в отчёте привести этапы решения:

1.1. Формулировка задачи.
1.2. Математическая модель решения задачи:
   a) Описать, как выполняется алгоритм (словами и блок-схемой).
   b) Определить для цикла инвариант цикла и доказать его конечность, т.е. доказать корректность цикла[1].
   c) Определить вычислительную сложность алгоритма используя теоретический подход, т.е. вывести функцию роста T(n) отдельно для худшего и лучшего случаев. Сделать вывод о вычислительной сложности алгоритма в среднем случае.
1.3. Реализовать алгоритм в виде функции и отладить на массиве при n=10, n=100 (см. примечание[2]). Включить в функцию подсчет суммарного количества выполненных сравнений и перемещений элементов при решении задачи удаления ключевых элементов.
1.4. Реализовать функции: заполнение массива датчиком случайных чисел, вывод массива на экран монитора.
1.5. Протестировать алгоритм в ситуациях: а) случайное заполнение массива; б) все элементы массива должны быть удалены; в) ни один элемент не удаляется. Сравнить результаты вычислительной сложности этих случаев.
1.6. Представить в отчёте результаты тестирования, указав для каждого случая количество операций: а) согласно теоретическим расчетам – по функции роста и б) полученным при выполнении алгоритма, т.е. практически.
1.7. Оценить ёмкостную сложность алгоритма.

2. Включите в отчёт обоснованный вывод по заданию 1 о том, какой алгоритм из двух рассмотренных является более эффективным.

Задание 2. Индивидуальная задача

1. Выполнить разработку алгоритма в соответствии с задачей варианта (табл. 2), включив в отчёт описание следующих этапов:
   1.1. Формулировка задачи.
   1.2. Математическая модель решения[3].
   1.3. Разработка **эффективного** алгоритма:
      а) разработать алгоритм;

[1] В случае наличия вложенных циклов можно доказать корректность только внешнего цикла.
[2] Во всех заданиях этой работы допустимо использовать только динамические массивы (new/delete или malloc/free).
[3] Аналогично заданию 1.

в) доказать корректность циклов в алгоритме (определить инвариант цикла и доказать его конечность);

г) определить вычислительную сложность алгоритма на основе теоретического подхода (вывести функцию роста) отдельно в худшем и лучшем случаях; сделать вывод о вычислительной сложности алгоритма в среднем случае.

1.4. Реализовать алгоритм решения задачи варианта в виде одной функции (без декомпозиции на другие функции).

1.5. Провести тестирование алгоритма. Для этого разработать таблицу тестов в соответствии с ограничениями постановки задачи. Выполнить тестовые прогоны и убедиться, что все требования выполняются.

1.6. Выполнить практическую оценку сложности алгоритма для больших n (по счётчикам количества сравнений и перемещений данных в памяти). Показать результаты прогонов для заданного n в лучшем, худшем и среднем случаях.

1.7. Оценить ёмкостную сложность алгоритма.

2. Включите в отчет результаты по заданию 2, отобразив описание выполнения всех этапов с 1.1 по 1.6. и код всей программы со скринами результатов тестирования.

Таблица 2. Варианты индивидуальных заданий

№ варианта	Задача
1	Умножение квадратных матриц.
2	Умножение матрицы на вектор.
3	Сложение двух матриц
4	Получение матрицы обратной данной матрице
5	Обход матрицы по спирали (по часовой стрелке: первая строка, последний столбец, нижняя строка, первый столбец)
6	Найти максимальный элемент в части матрицы, расположенной над главной диагональю.
7	Найти минимальное четное число в части матрицы – между главной и побочной диагоналями (диагонали образуют вертикальные песочные часы).
8	Найти восходящую диагональ матрицы с максимальной суммой элементов.
9	Определить, симметрична ли матрица относительно главной диагонали.
10	Выполнить транспонирование матрицы
11	Дан одномерный массив из n элементов целого типа. Определить, сколько раз в массив входит максимальное значение.

12	Реализовать алгоритм «схема Горнера» вычисления значения линейного многочлена n-ой степени.
13	Дана прямоугольная матрица размером n*m. Определить максимальное из чисел, встретившихся в матрице более одного раза.
14	Коэффициенты системы линейных уравнений заданы в виде прямоугольной матрицы размером. С помощью допустимых преобразований привести систему к треугольному виду (коэффициенты должны быть только над главной диагональю). Примечание. Система состоит из n уравнений с n неизвестными. Матрица имеет размер n*(n+1). Т.е. i-ая строка матрицы хранит коэффициенты i-ого уравнения и свободный член.
15	Дана целочисленная прямоугольная матрица размером n*m. Характеристикой строки матрицы назовем сумму ее положительных четных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с ростом характеристик.
16	Дана целочисленная квадратная матрица размером n*n. Найти минимум среди сумм модулей элементов диагоналей параллельных побочной диагонали.
17	Дана целочисленная прямоугольная матрица размером n*m. Определить номер строки, в которой находится самая длинная серия одинаковых элементов. Пример строки с серией из четырех чисел 3: 1 2 3 3 3 3 5.
18	Дан массив из n элементов целого типа. Преобразовать массив следующим образом, чтобы сначала располагались все элементы равные 0, затем все остальные.
19	Найти количество натуральных чисел, не превосходящих заданного n и делящихся на каждую из своих цифр.

Приложение 1. Титульный лист отчёта

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания № НОМЕР

Тема:

Тема

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: ФИО

Фамилия И.О.

Группа: Группа

Номер группы

Москва – 2023

Приложение 2. Примеры определений функции роста

Теоретический подход позволяет определить функцию роста времени в зависимости от размера задачи - размера входных данных, который определяет количество элементов структуре, чаще обозначаемое через n. Т.е. функция роста показывает рост времени при увеличении размера входных данных.

Введем понятие *скорость роста* (rate of growth), или *порядок роста* (order of growth), времени работы, который и интересует нас на самом деле. Таким образом, во внимание будет приниматься только главный член формулы (т.е. функция более высокого порядка), поскольку при больших n, членами меньшего порядка можно пренебречь.

Постоянные множители при главном члене также будут игнорироваться, так как для оценки вычислительной эффективности алгоритма с входными данными большого объема они менее важны, чем порядок роста.

При определении функции роста подсчитывается количество выполняемых в алгоритме операторов. Подсчитывать можно не все, а только те, которые чаще всего применяются в алгоритме – критические, например, в алгоритме поиска критическим является оператор сравнения, а в алгоритме сортировки – два критических оператора: сравнение и перемещение.

<u>Примеры определения функции роста</u>

В качестве тестовых данных будет использоваться массив из n элементов целого типа.

Для удобства подсчета количества операторов в алгоритме и определении функции роста времени, зависящего от n, будет использоваться таблица, формат которой определен в табл.3.

Таблица 3

Оператор	Кол-во выполнений оператора в строке	
	в лучшем случае	в худшем случае

Некоторые правила построения таблицы:

В столбце *Оператор* построчно размещаем операторы[4] алгоритма (псевдокод или текст на ЯВУ, например, на С++).

[4] В примерах ниже ведётся подсчёт не всех операторов алгоритма, а только критических, т.е. тех, которые вносят наибольший вклад в общее время работы алгоритма.

Если в операторе if в блоках истина и ложь несколько операторов, то можно разместить их в одной строке, эти операторы будут выполняться столько раз, сколько раз выполниться if.

Если в теле цикла несколько операторов, то каждый оператор будет выполняться столько раз, сколько входов в цикл.

Если в теле цикла есть вложенный цикл, то оператор заголовка также оператор внешнего цикла и выполняться будет столько же раз, сколько и другие операторы внешнего цикла. Но вложенный цикл тоже будет выполнять итерации по своему условию, и тогда, количество его выполнений будет рассчитываться так:

- при каждой активизации вложенного цикла считается время его выполнения, как функция, зависящая от размера обрабатываемых, при данной активизации, данных;
- затем все полученные значения (время каждого выполнения оператора) складываются, получаем сумму, в которой столько слагаемых, сколько раз выполнялось тело внешнего цикла.

В столбце *Кол-во выполнений оператора в строке* указывается сколько раз оператор будет выполняться при полном выполнении алгоритма – константой или выражением, зависящим от n.

Пример 1. Поиск заданного значения в массиве из n элементов. Считать, что элемент с таким значением единственный (если присутствует в массиве).

Обозначения:

x – массив, n – количество элементов в массиве, key – значение, которое нужно найти, ikey – индекс найденного элемента.

Результат: если значение будет найдено, то вернуть индекс найденного элемента; иначе вернуть -1. Все данные алгоритма, для вывода функции роста представлены в табл. 4.

Таблица 4

Номер оператора	Оператор	Кол-во выполнений оператора в строке
1	ikey = -1;	1
2	for (int i=0; i<n; i++){	n+1
3	if (x[i]==key)	n
4	ikey = i;	n
	}	
5	return ikey;	

Описание результатов, представленных в табл. 4:

Оператор 1 выполняется один раз.

Оператор 2. Определяется количество выполнений условия цикла при просмотре всех значений (худший случай), т.е. сколько раз он применится
Согласно циклу с предусловием: первый вход в цикл при i=0; оследний вход в цикл при i=n-1; после последнего входа i=n, т.е. ещё одна проверка и завершение цикла. Считаем сколько раз выполнялся оператор i <n: n раз обеспечивался вход в цикл и один раз при выходе из цикла, таким образом, всего n+1 раз за время работы.
Оператор 3(if). Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.
Оператор 4. Это оператор – блок оператора if. Выполняется столько раз, сколько и if (в худшем случае) – n раз.
Оператор 5. Этот оператор вне цикла ион выполняться будет 1раз.
Обозначим время выполнения алгоритма T(n) – функция, зависящая от n.
Определим время выполнения алгоритма - как сумму времени выполнения каждого оператора:
T(n) = 1+(n+1)+n+n.
Раскроем скобки и приведем подобные:
T(n) = 3*n+2, т.е. имеем линейную зависимость времени от размера входных данных.
Так как требуется определить порядок роста времени от n, то константами можно пренебречь.
В результате T(n) в худшем случае линейно зависит от n.
Рассмотрим лучший случай: значение равное key – это значение первого элемента массива. Цикл будет выполняться все равно n+1 раз. Тогда T(n) в лучшем случае линейно зависит от n.
Рассмотрим средний случай: значение равное key – это значение элемента в позиции i=n/2 (целое значение) массива. Цикл будет выполняться все равно n+1 раз. Тогда T(n) в среднем случае линейно зависит от n.
Вывод. Порядок роста – линейный.

Пример 2. Реализация алгоритма предыдущего примера 1 с повышением эффективности в наилучшем случае за счет завершения выполнения функции при удачном поиске – выполнение оператора return i.
В табл. 5 представлены данные для подсчета количества выполняемых алгоритме операторов.
Внешний цикл зависит от n и выполняется n+1 раз.
Операторы тела цикла n раз.

Операторы тела условного оператора выполняются столько же раз, сколько и сам условный оператор.

Таблица 5

Номер оператора	Оператор	Кол-во выполнений оператора в строке
1	for (int i=0;i<n; i++){	n+1
2	if (x[i]==key)	n
3	return i;	
	}	
4	return -1;	

Время выполнения алгоритма

$$T(n)= 2*n+1 \qquad (1)$$

В худшем случае: цикл выполняется полное число раз – n+1.

В *наилучшем случае*: значение, которое ищем, в первом элементе массива. Условие цикла выполнится два раза, не зависит от n.

В *среднем случае*: значение, которое ищем, где-то в середине массива в позиции n/2. Подставим n/2 в формулу (1) вместо n:

$T(n)= n +1$, т.е. тоже линейная зависимость.

Вывод. Худший и средний случай дают одинаковый порядок роста вымени от n. Значит, в целом, скорость роста – линейная.

Пример 3. Пример алгоритма с вложенным циклом, время выполнения которого зависит от n, как и время его внешнего цикла. Рассмотрим подход более точного определения функции роста времени от n.

Рассмотрим алгоритм сортировки прямого обмена или «Пузырек» на примере сортировки по возрастанию массива A из n элементов целого типа. Представим массив вертикально, чтобы показать механизм обмена, в соответствии с названием алгоритма «Пузырек», легкие пузырьки будут всплывать, обгоняя тяжелые. Размер входных данных n=5.

За один проход по массиву на место, согласно правилу сортировки, встает один элемент. Проходов будет n-1.

Описание алгоритма первого прохода, остальные будут выполняться так же. Проход начинается от последнего элемента массива и перемещается вверх.

Сравниваются элементы: A[i] и A[i-1], если A[i]<A[i-1], то происходит обмен значениями этих элементов – меньший «всплывает», так как сортировка по возрастанию. В результате первого прохода значение 1 станет первым элементом массива.

Исходный массив: 5 6 1 2 3. Выполнение алгоритма сортировки подставлено в таблице табл. 6.

Холостые проходы, т.е. элементы не сортировались.

Первый проход затрагивает элементы от n до 1, т.е. будет выполнено n-1 сравнение (т.е. 4).

Второй проход затрагивает элементы от n до 2, будет выполнено n-2 сравнения (т.е.3).

Третий проход затрагивает элементы от n до 3, будет выполнено n-3 сравнения (т.е. 2).

Таблица 6

Номер прохода j	*1*	*2*	*3*	*4*	
Индекс элемента i					
1	1	1	1	1	
2	5	2	2	2	
3	6	5	3	3	
4	2	6	5	5	
n=5	3	3	6	6	

Четвертый проход затрагивает элементы от n до 4, будет выполнено n-4 сравнений (т.е. 1).

Как видно из описания проходов количество элементов в очередном проходе уменьшалось на значение равное j.

Составим алгоритм и определим порядок роста алгоритма. В табл. 7 представлен алгоритм и определено максимальное количество выполнений внешнего цикла. В табл.6 будет представлено количество выполнений вложенного цикла, после обсуждения технологии для расчета его количества выполнений.

Таблица 7

Номер опера-тора	Оператор		Кол-во выполнений оператора в строке
1	for(int j=1;j<=n; j++){		n+1
2	for(int i=n;i>j;i--){		
3	if(A[i]<A[i-1])		
4	swap(A[i],A[i-1])		
	}		
	}		

Определим количество выполнений вложенного цикла (подсчитано в примере сверху для каждого прохода):

При j=1, оператор 2 выполнится n-1 раз.
При j=2, оператор 2 выполнится n-2 раза.
………………………………………………..
При j=k, оператор 2 выполнится n-k раз.
……………………………………………….
При j=n-2, оператор 2 выполнится 2 раз.
При j=n-1, оператор 2 выполнится 1 раз.
При j=n, оператор 2 выполнит еще одно сравнение и цикл завершиться.

Общее количество выполнений этого оператора будет равно сумме (запишем слагаемые от последнего значения(j=n-1) к первому(j=1)): 1+2+……+(n-k)+….+(n-2) +(n-1), т.е. получили сумму членов арифметической прогрессии с разностью равной 1.

Тогда общее количество выполнений этого оператора можно представить формулой: $\sum_{j=1}^{n-1}((n-j)+1)$ или $\sum_{j=1}^{n}(n-j)$. Далее будет использоваться $\sum_{j=1}^{n-1} t_j$, где t_j количество выполнений j-ого оператора.

Внесем полученные результаты в таблицу табл. 8.

Таблица 8

Номер опера-тора	Оператор	Кол-во выполнений оператора в строке
1	for(int j=1;j<=n; j++){	n+1
2	for(int i=n;i>j;i--){	$\sum_{j=1}^{n} t_j$
3	if(A[i]<A[i-1])	$\sum_{j=1}^{n-1} t_j$
4	swap(A[i],A[i-1])	$3*\sum_{j=1}^{n-1} t_j$
	}	
	}	

Выведем функцию роста для времени выполнения алгоритма:

T(n)=(n+1)+$\sum_{j=1}^{n} t_j$+$\sum_{j=1}^{n-1} t_j$+ 3*$\sum_{j=1}^{n-1} t_j$ (1)

Определим *порядок роста в лучшем случае*, т.е. когда тело вложенного цикла не выполняется (массив уже отсортирован по рассматрваемому в алгоритме правилу), но условие один раз проверяется (т.е. оператор 2 один раз выполниться обязательно). В наилучшем случае в сумме $\sum_{j=1}^{n} t_j$ значение t_j равно 1. Тогда значение $\sum_{j=1}^{n} t_j$ = n. Значение $\sum_{j=1}^{n-1} t_j$ =n-1; Подставим эти значения в формулу (1).

Порядок роста времени в зависимости от n наилучшем случае линейный.

Определим *порядок роста в худшем случае*, т.е. когда оператор 2 выполняется полное количество раз (массив упорядочен, но по правилу, противоположному тому, что рассматривает алгоритм).
В этом случае в сумме
$\sum_{j=1}^{n} t_j$ значение t_j равно j.
Тогда значение
$\sum_{j=1}^{n} t_j = \sum_{j=1}^{n} t_j = \frac{n(n-1)}{2} = \frac{n^2-n}{2}$.
Значение $\sum_{j=1}^{n-1} t_j = \frac{(n-1)(n-1)}{2} = \frac{n^2-2n+1}{2}$.
Подставим эти значения в формулу (1). Получили порядок роста как функцию – квадратный трехчлен вида $T(n)=An^2+Bn+C$.
Функция n^2 имеет порядок роста выше, чем функция n. Таким образом, в выражении для T(n) доминирующей функцией является $n^{2,}$ и она определяет порядок роста для алгоритма в худшем случае. Т.е. скорость роста - квадратичная.
Пример 4. Пример определения сложности алгоритма пузырьковой сортировки в худшем случае. Другой подход к определению количества выполнений вложенного цикла.
Будем считать, что вложенный цикл при каждом проходе выполняется n раз. Тогда за n проходов внешнего цикла (оператор 1), вложенный цикл (оператор 2) активизируется n раз и выполнит (n+1) сравнение. Операторы тела цикла выполняться на один раз меньше, чем оператор 2 (для последнего сравнения, когда цикл завершается, эти операторы не выполняются). В табл. 9 представлена другая технология подсчета количества операторов, выполняемых в алгоритме.
Приведя подобные, получаем квадратичный вид функции роста T(n), т.е. порядок роста – квадратичный.

Таблица 9

Номер оператора	Оператор	Кол-во выполнений оператора в строке
1	for(int j=1;j<=n; j++){	n+1
2	for(int i=n;i>j;i--){	n(n+1)
3	if(A[i]<A[i-1])	n(n+1)-1
4	swap(A[i],A[i-1])	3*(n(n+1)-1)
	}	
	}	