Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н. Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Р А С Ч Ё Т Н О - П О Я С Н И Т Е Л Ь Н А Я З А П И С К А

к лабораторной работе на тему:

Оценка трудоемкости сортировок

Студент _____ Киселев А.М.
(Подпись, дата)

Преподаватель _____ Волкова Л.Л.
(Подпись, дата)

Москва 2018

Содержание

1 Аналитический раздел

Задача упорядочивания элементов имеет широкое применения в совершенно разных сферах: упорядочивание слов по алфавиту, ряда чисел или структур данных. Многие алгоритмы нуждаются в упорядоченных рядах чисел для продолжения вычислений.

Сортировка ряда чисел сводится к получению упорядоченнего ряда. Пусть задан ряд $\{5, 3, 2, 6, 1\}$, тогда результат сортировки будет $\{1, 2, 3, 5, 6\}$. Существует множество методов сортировки, которые позволяют получить упорядоченный массив чисел. Одними из таких алгоритмов являются: метод пузырька, метод слияния и метод простыми вставками.

1.1 Сортировка пузырьком

Пусть дан массив $\{3, 2, 6, 7, 9, 4\}$. Дана задача отсортировать его по убыванию. При решении этой задачи, алгоритм пройдется несколько раз по этому массиву, каждый раз проходя все меньше количество элементов. Первый проход представлен в таблице 1.1.

Таблица 1.1 — Процесс обработки массива на первой итерации алгоритма пузырька

Нет обмена	**2 ↔ 6**	**2 ↔ 7**	**2 ↔ 9**	**2 ↔ 4**
4	4	4	4	**2**
9	9	9	**2**	**4**
7	7	**2**	**9**	9
6	**2**	**7**	7	7
2	**6**	6	6	6
3	3	3	3	3

Далее последний эллемент массива фиксируется и действия сортировки продолжаются, но уже не до последнего элемента массива, а до предпоследнего. Количество обрабатывающихся элементов на каждой итерации изменяется как $N = N - 1$, где N изначально является длинной массива. Полный результат работы алгоритма на каждой итерации предеставлен в таблице.

1.1.1 Сложность алгоритма пузырька[1]

При первой итерации совершается n-1 сравнений, поэтому даже в наилучшем случае(массив и так упорядочен) совершается n-1 сравнений.

Рассматривая худший случий – сортируемые элементы следуют в порядке, обратному требуемому. В этом случае цикл будет повторен n-1 раз. На превой итера-

ции выполняется n-1 сравнений, на второй итерации n-2 сравнений, и т.д. На последней итерации будет выполнено 1 сравнение. Общее число сравнений в наихудшем случае равно:

$$C_{max} = \sum_{i-1}^{n-1} i = \frac{(n-1)*n}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2) \tag{1.1}$$

Перестановка возникает, если порядок сравниваемых элементов противоположен требуемому (а в наихудшем случае – всегда). Поскольку каждая перестановка содержит 3 пересылки, общее число пересылок:

$$M_{max} = 3 * C_{max} = O(n^2) \tag{1.2}$$

Рассмастривая средний случай, надо учитывать, что появление прохода без перестановки равновероятно в любом из n-1 проходов. Пусть C_i – число сравнений, выполняемых на первых i проходах. Тогда среднее число сравнений:

$$C_{mid} = \frac{1}{(n-1)} \sum_{i=1}^{n-1} C_i \tag{1.3}$$

Из анализа структуры цикла for получаем:

$$C_i = \sum_{j=1}^{n-1} j = \sum_{j=1}^{n-1} j - \sum_{j=1}^{i} j = \frac{n*(n-1)}{2} - \frac{i*(i-1)}{2} \tag{1.4}$$

Из 1.4

$$C_{mid} = \frac{n^2}{3} - \frac{n}{6} = O(n^2) \tag{1.5}$$

Таблица 1.2 — Результат работы алгоритма пузырька на каждой итерации

4	**2**	**2**	**2**	**2**	**2**
9	4	**3**	**3**	**3**	**3**
7	9	4	**4**	**4**	**4**
6	7	9	9	**6**	**6**
2	6	7	7	9	**7**
3	3	6	6	7	9
i=0	**i=1**	**i=2**	**i=3**	**i=4**	**i=5**

1.2 Сортировка простыми вставками

В сортировке вставками последовательно обрабатываются отрезки массива, начиная с первого элемента и затем последовательно расширяя подмассив, вставляя на своё место очередной неотсортированный элемент.

Для вставки используется буферная область памяти, в которой хранится элемент, ещё не вставленный на своё место (так называемый ключевой элемент). В подмассиве, начиная с конца отрезка, перебираются элементы, которые сравниваются с ключевым. Если эти элементы больше ключевого, то они сдвигаются на одну позицию вправо, освобождая место для последующих элементов. Если в результате этого перебора попадается элемент, меньший или равный ключевому, то значит в текущую свободную ячейку можно вставить ключевой элемент.

1.3 Сортировка слиянием

Это рекурсивный алгоритм, который постоянно разбивает список пополам. Если список пуст или состоит из одного элемента, то он отсортирован по определению (базовый случай). Если в списке больше, чем один элемент, мы разбиваем его и рекурсивно вызываем сортировку слиянием для каждой из половин. После того, как обе они уже отсортированы, выполняется основная операция, называемая слиянием. Слияние - это процесс комбинирования двух меньших сортированных списков в один новый, но тоже отсортированный.

Таким образом задача разбивается на подзадачи, которые в свою очередь рекурсивно доходят до базовых случаев. Предположим, что нужно отсортировать массив A. Подзадачей будет сортировка подраздела этого массива начиная с индекса p и заканчивая индексом r, обозначенного как A[p..r].

Если q – точка на полпути между p и r, то подмассив A[p..r] можно разделить на A[p..q] и A[q+1,r]. Далее оба массива сортируются и если базовый случай не был достигнут, то повторяем те же действия для каждогои з подмассивов.

1.3.1 Сложность сортировки слиянием[1]

Данная сортировка всегда имеет сложность $O(nlogn)$ это вытекает из уравнения 1.6

$$T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + O(n) \tag{1.6}$$

$logn$ – добавление узла в сбалансированное дерево.

1.4 Рассчет трудоемкости алгоритма

Рассчет трудоемкости алгоритма опирается на модель вычисления, в которой нужно определить свод правил : оценка операций, циклов, стоимость перехода по условиям.

1.4.1 Модель вычеслений

Стоимость каждой операции из следующего множества будем считать 1: {+,-,*, /, %, <, <=, !=, >=, >, =, :=, &, ||, []}

Стоимость условного перехода if-else – 0 и будет учитывать лишь стоимость выражения условия.

Стоимость циклал будет рассчитываться следующим образом:

Пусть цикл задан так:

$$for(i = 0, i < N, i + +)\{body\} \tag{1.7}$$

, где N – количество итераций

Тогда i=0, i<N – операции, выполняемые перед началом выполнения тела цикла и их общая стоимость равна 2, i++, i<N – операции, которые выполняются каждую итерацию(их стоимость в общем тоже равна 2)

$$f_{for} = 2 + N * (2 + f_{body}) \tag{1.8}$$

2 Конструкторский раздел

2.1 Разработка алгоритмов

2.1.1 Алгоритм сортировки пузырька

В листенге 2.1 представлен алгоритм сортировки пузырьком.

Листинг 2.1 — Pseudocode of bubble sort

```
цикл для j от min до max
    цикл для i от min до max-1
        если A[i] больше чем A[i+1] то:
            обменять местами A[i] и A[i+1]
```

2.1.2 Алгоритм сортировки слиянием

В листинге 2.2 представлен алгоритм сортировки слиянием.

Листинг 2.2 — Pseudocode of merge sort

```
функция mergesort( массив A )
    Если n=1 то:
        вернуть A

    l1 = A[0] .. A[n/2]
    l2 = A[n/2+1] .. A[n]

    l1 = mergesort( l1 )
    l2 = mergesort( l2 )

    вернуть merge( l1, l2 )
конец функции

функция merge( массив A, массив B )
    пока в A и в B есть элементы:
        если A[0] > B[0]:
            добавить B[0] в конец C
            удалить B[0] из B
        иначе
            добавить A[0] в конец C
            удалить A[0] из A
    пока в A есть элементы:
        добавить A[0] в конец C
        удалить A[0] из A
    пока в B есть элементы:
        добавить B[0] в конец C
```

```
27          удалить B[0] из B
28      вернуть C
29  конец функции
```

2.1.3 Алгоритм сортировки простыми вставками

Листинг алгоритма сортировки простыми вставками представлен в 2.3

Листинг 2.3 — Pseudocode of insertions algorithm

```
1  цикл для i от 0 до n - 1
2      j = i - 1
3      пока j >= 0 и a[j] > a[j + 1]
4          поменять местами a[j] и a[j + 1]
5          j—
```

3 Технологический раздел

Данный раздел содержит информацию о реализации ПО и листингах программ.

3.1 Требования к программному ПО

Программы реализованы в двух вариациях. Первая – каждая программа реализована поотдельности и обладает своим собственным функционалом – возможность определять размеры массива и получать вывод в поток вывода. Вторая – программа, содержащая в себе функции с реализованными алгоритмами, в которой производится тестирование времени алгоритмов и вывод отправляется в поток вывода информации.

3.2 Средства реализации

Данные реализации разрабатывались на языке C, использовался компилятор gcc-8.2.1. Замер времени производится с помощью библиотеки *time.h*. Данная библиотека позволяет производить замер тиков, откуда можно получать время в секундах. Программы компилировались с выключенной оптимизацией с флагом -O0

3.3 Листинг

Алгоритм сортировки пузырьком представлен в листинге 3.1

```
void bubble(int* a, int size)
{
    for(int i = 0; i < size; i++)
        for(int j = 0; j < size-j-i; j++)
            if(a[j] > a[j+1])
                swap(&a[j], &a[j+1]);
}
```

Листинг 3.1 — bubble sort

Алгоритм сортировки слиянием представлен в листинге 3.2

```
void merge(int* a, int* l, int leftCount, int* r, int rightCount)
{
    int i = 0,
        j = 0,
        k = 0;

    while(i<leftCount && j< rightCount) {
        if(l[i]  < r[j]) a[k++] = l[i++];
```

```
        else a[k++] = r[j++];
    }
    while(i < leftCount) a[k++] = l[i++];
    while(j < rightCount) a[k++] = r[j++];
}

void mergeSort(int* a, int n)
{
    int mid, i, *l, *r;
    if(n < 2) return;

    mid = n/2;

    l = (int*)malloc(mid*sizeof(int));
    r = (int*)malloc((n - mid)*sizeof(int));

    for(i = 0; i<mid; i++) l[i] = a[i]; // creating left subarray
    for(i = mid; i<n; i++) r[i-mid] = a[i]; // creating right subarray

    mergeSort(l, mid);
    mergeSort(r, n-mid);
    merge(a, l, mid, r, n-mid);
    free(l);
    free(r);
}
```

Листинг 3.2 — merge sort

Листинг алгоритма сортировки простыми вставками представлен в 3.3

```
void insert(int* a, int size)
{
    int j, tmp;

    for(int i = 1; i < size; i++) {
        tmp = a[i];
        for (j = i - 1; j >= 0 && a[j] > tmp; j--)
            a[j + 1] = a[j];
        a[j + 1] = tmp;
    }
}
```

Листинг 3.3 — Insertions sort

4 Экспериментальный раздел

Результаты работы алгоритмов представлены на диаграммах изображенных на рисунках 4.1, 4.2 и 4.3 на которых:

1 – Сортировка простыми вставками(изображена черным цветом)

2 – Сортировка слиянием(изображена красным цветов)

3 – Сортировка пузырьком(изображена синим цветом)

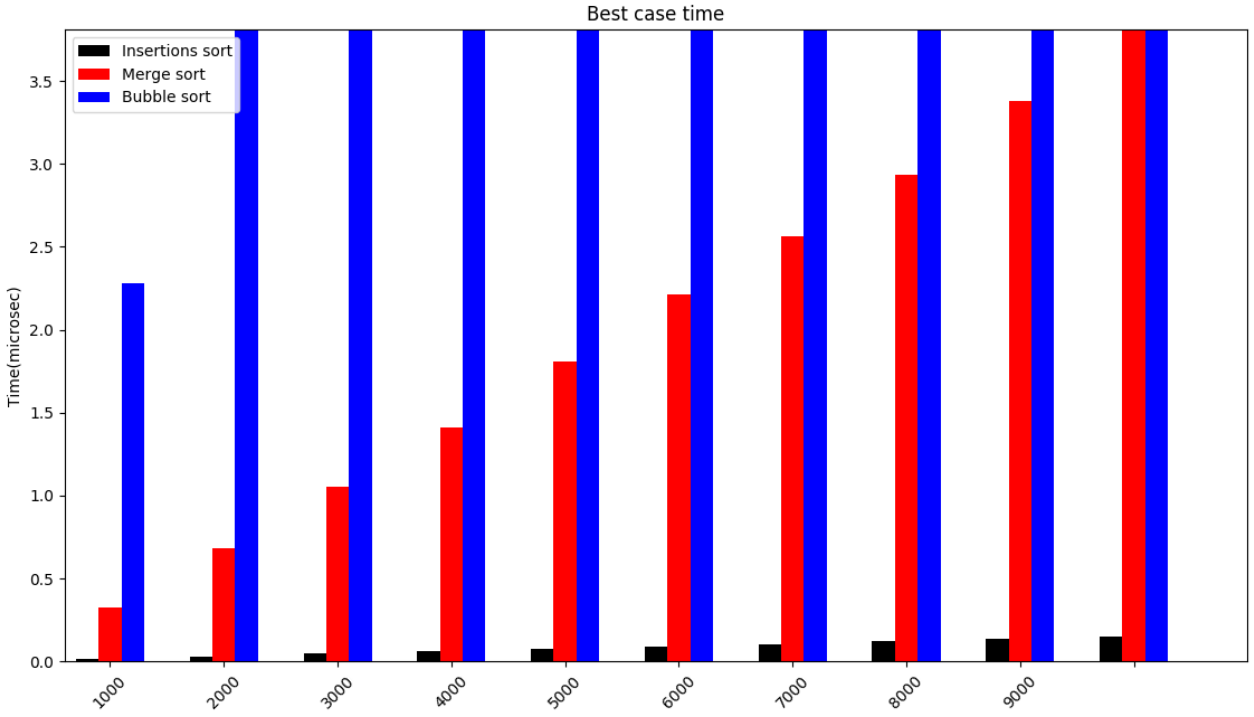

Рисунок 4.1 — График, показывающий временные замеры алгоритмов сортировки для лучшего случая

По полученным данным можно сделать вывод, что стандартный алгоритм перемножения матриц работает медленнее, чем классический алгоритм Винограда и улучшенный алгоритм Винограда. Улучшенный алгоритм работает быстрее по времени, чем классический. Если обратиться к рассчитанным трудоемкостям алгоритмов, можно заметить, что стандартный алгоритм менее трудоемкий, чем два других, но время, затрачиваемое на рассчет матрицы стандартным методом все равно очень большое. Так как это зависит от модели вычислений, где умножение и обращения к индексам и сложение были взяты за 1, можно сказать, что следует учитывать некоторые ньюансы при выборе весов операций, чтобы вычесленная трудоемкость была более точной.

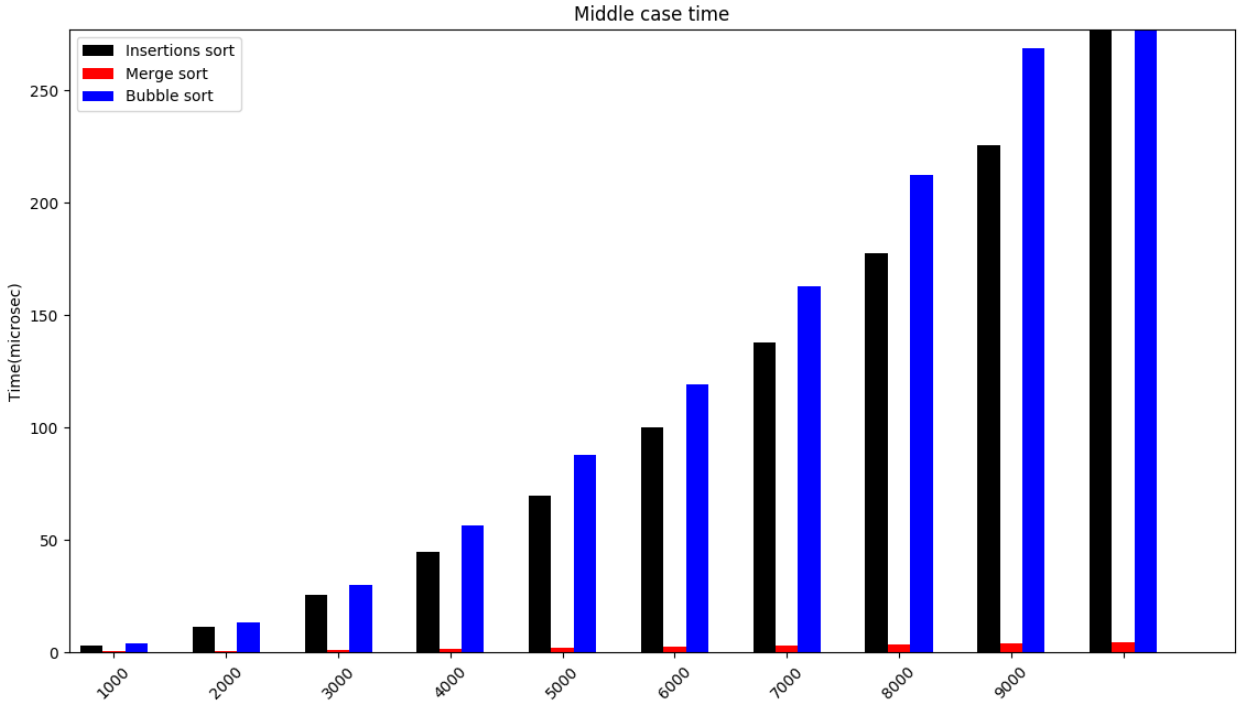

Рисунок 4.2 — График, показывающий временные замеры алгоритмов сортировки для произвольного случая

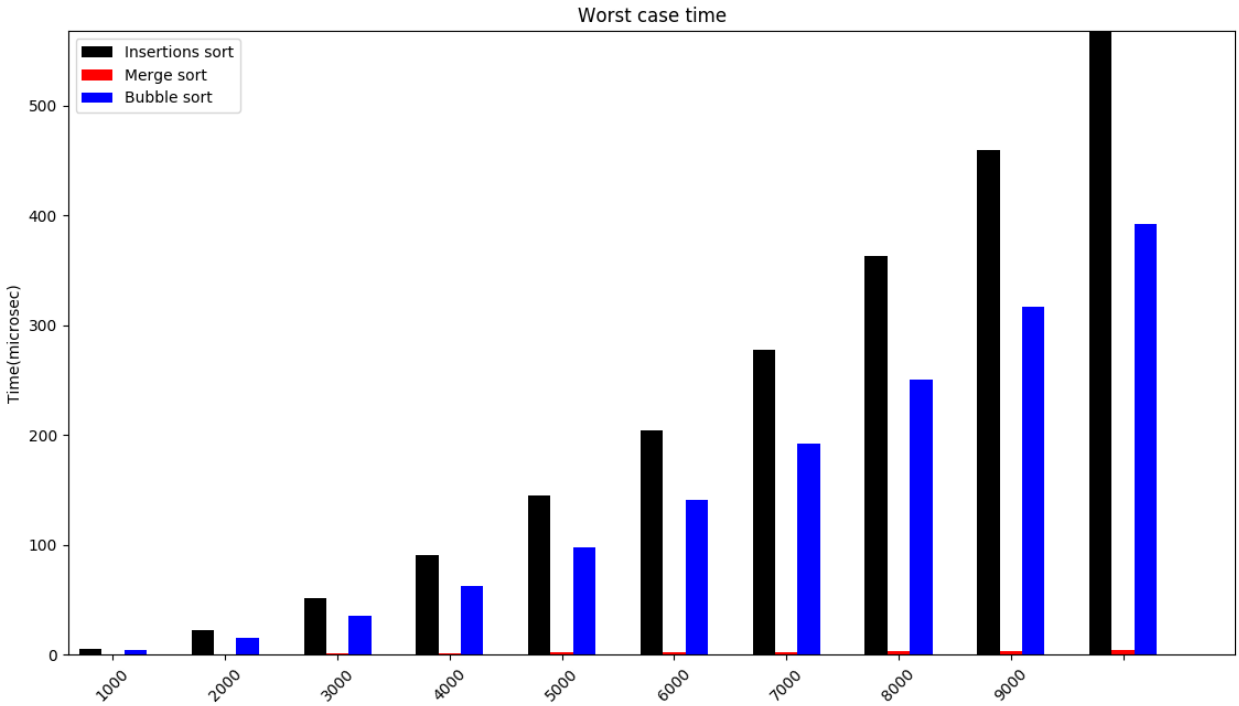

Рисунок 4.3 — График, показывающий временные замеры алгоритмов сортировки для худшего случая

Заключение

В результате выполнения лабораторной работы были получены следующие основные навыки:

а) изучены теоритеческие понятия в алгоритмах перемножения матриц;

б) рассмотрены различные способы оптимизации алгоритмов;

в) проведено сравнение четырех реализаций заданного алгоритма Винограда, выявлены их слабые места;

Список использованных источников

1. *Е.В., Пышкин.* Структуры данных и алгоритмы: реализация на C/C++ / Пышкин Е.В. — ФТК СПБГПУ, 2009.