|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

**О Т Ч Е Т**

по лабораторной работе №\_1\_

Вариант № 25

**Дисциплина**:\_ Технология разработки программных систем \_

**Название**: Исследование структур и методов обработки Данных \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-42б |  |  | И.С. Марчук |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Е.К. Пугачев |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |

*2021 г.*

**Цель работы –** исследование структур данных, методов их обработки и оценки.

**Задание:** Вариант 25

Задача 6 – Даны элементы: 130, 50, 120, 185, 27, 43, 913, 210, 5, 17, 245;

Структура данных – список;

Метод поиска – последовательный;

Метод упорядочивания – любой;

Метод корректировки – удаление записи;

**Основная часть**

**1. Исходные варианты структуры и методов ее обработки**

В качестве списка на языке java был разработан класс, реализующий структуру односвязного списка, представленную на рисунке 1.



Рисунок 1 – Структура односвязного списка

**1.1 Оценка памяти**

Объем памяти, отводимый на 1 запись:

Velement = Vnumber + Vint = 4 +4= 8 Байт

Объем памяти, занимаемый списком из N элементов

Vlist =Vfirstptr + Velement\*N=4 +8\*N Байт

Как видно из формулы, между памятью и количеством элементов – линейная зависимость.

**1.2 Анализ алгоритма поиска**

По заданию необходимо использовать последовательный поиск.

Последовательный поиск был реализован следующим образом (см. рисунок 2):

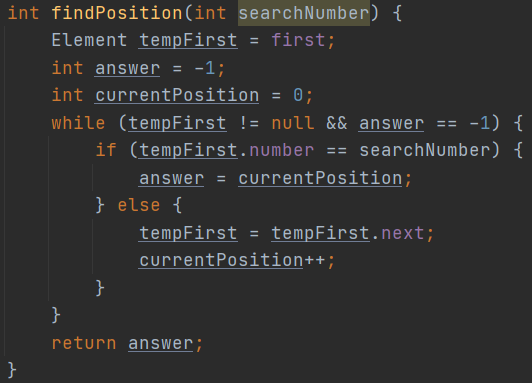


Рисунок 2 – Алгоритм последовательного поиска

Доступ к элементам в списке выполняется последовательно.

По таблице из методических указаний было определено время поиска, в тактах (рисунок 3):

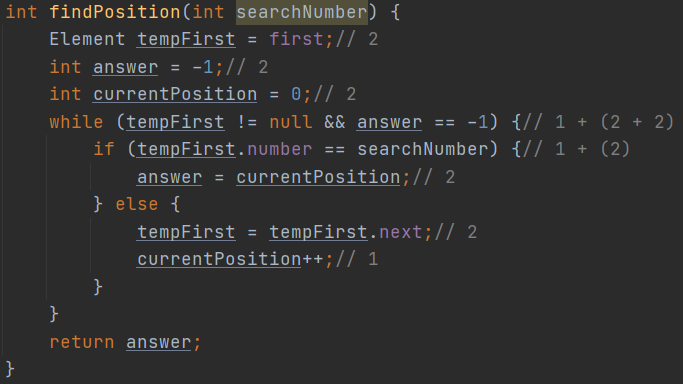


Рисунок 3 – Время поиска в тактах для каждой команды алгоритма

Введем условные обозначения:

“nach” = 2+2+2;

“while” = 1+2+2;

“if “= 1+2;

“yes” = 2;

“no” = 2+1;

t0 = nach+ (while+ if+yes+while)

t1 = nach+ (while+ if+no+while +if+yes+while)

t2 = nach+ (while+ if+no+while +if+no+while +if+yes+while)

В итоге получим время поиска n-го элемента:

tn = nach+ (while+ n\*(if+no+while) + if+yes+while);

tn = 2+2+2 + (1+2+2 +n\*(1+2+2+1+1+2+2)+1+2+2+1+2+2) =

=21 +11n,

где n – номер искомого элемента в списке (нумерация с 0).

Соответственно легко получить максимальное и минимальное время поиска в тактах:

tmax = 21+11n;

tmin = 21;

tср = (21+11n+21)/2 = 21+5.5n

Среднее число сравнений для реализации данного метода:

С = (N + 1) / 2.

**Анализ алгоритма упорядочения**

В задании не указан определенный алгоритм упорядочения. Я выбрал алгоритм сортировки bubble-sort.

На рисунке 4 представлена реализация алгоритма сортировки списка пузырьком на Java.

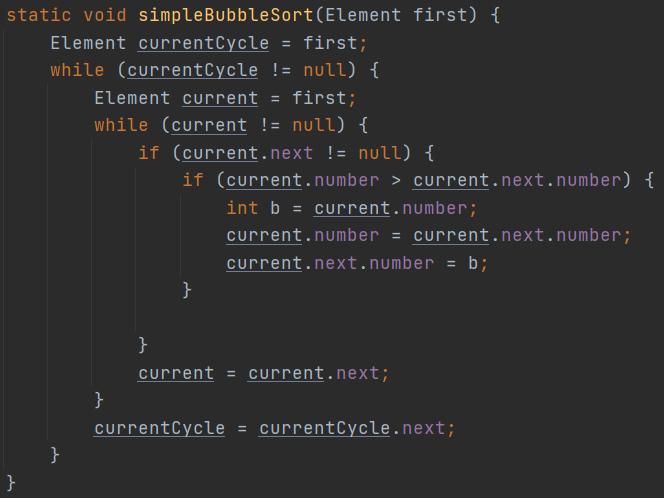


Рисунок 4 – Алгоритм bubble-sort

Параметр С – среднее количество сравнений для сортировки пузырьком

С = N \* ( N −1)

Исходя из этого, очевидно, что данная сортировка будет иметь квадратичную сложность.

**1.3 Анализ алгоритма корректировки**

В качестве корректировки требуется реализовать механизм удаления записи из списка. Удаление по условию необходимо реализовать путем уничтожения элемента. При этом, так как это список, необходимо переназначить указатель на следующий элемент у элемента, предшествующего удаляемому. А для первого элемента, еще и переназначить указатель на начало списка.



Рисунок 5 – Алгоритм удаления элемента

Оценка времени удаления:

Первый элемент: tdel = 1+(1)+1+(1)+2+1+(1)+1 = 9;

Элемент посередине: tdel = 1+(1)+1+(1)+2+(n-1)\*(1+(1+1) +2+1)+1+(1)+1+(1)+1+(1)+2 = 14+(n-1)\*(6) = 8+6\*n;

Последний элемент: tdel = 1+(1)+1+(1)+2+(n-1)\*(1+(1+1) +2+1)+1+(1)+1+(1)+1+(1)+2+1 = 15+(n-1)\*(6) = 9+6\*n,

где n – номер удаляемого элемента в списке (нумерация с 0).

Отсюда можно понять, что зависимость времени выполнения алгоритма корректировки от порядкового номера элемента – линейная.

**1.4 Вывод**

Выбранный алгоритм упорядочения не подходит для большого количества элементов, так как имеет квадратичную сложность.

Алгоритмы поиска, удаления являются оптимальными, удаление

выполняется быстро за счет отсутствия необходимости в сдвиге элементов.

Алгоритм упорядочения нуждается в доработке.

**2. Альтернативные варианты структуры и методов ее обработки.**

**2.1 Анализ улучшенного алгоритма упорядочения.**

При доработке алгоритма упорядочения, необходимо в первую очередь, ускорить его. Поэтому было решено использовать другой алгоритм упорядочения – merge sort, который работает быстрее на больших данных. И хорош если элементы можно получать только последовательно. Реализация алгоритма на Java представлена на рисунках 6, 7 и 8.

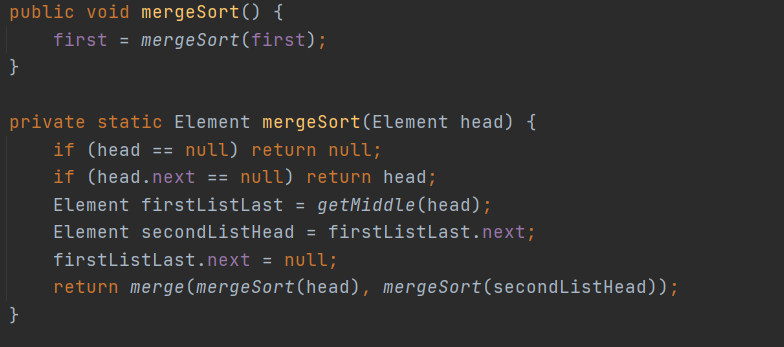


Рисунок 6 – Метод запуска сортировки слиянием и рекурсивный алгоритм разделения коллекции на две дочерние

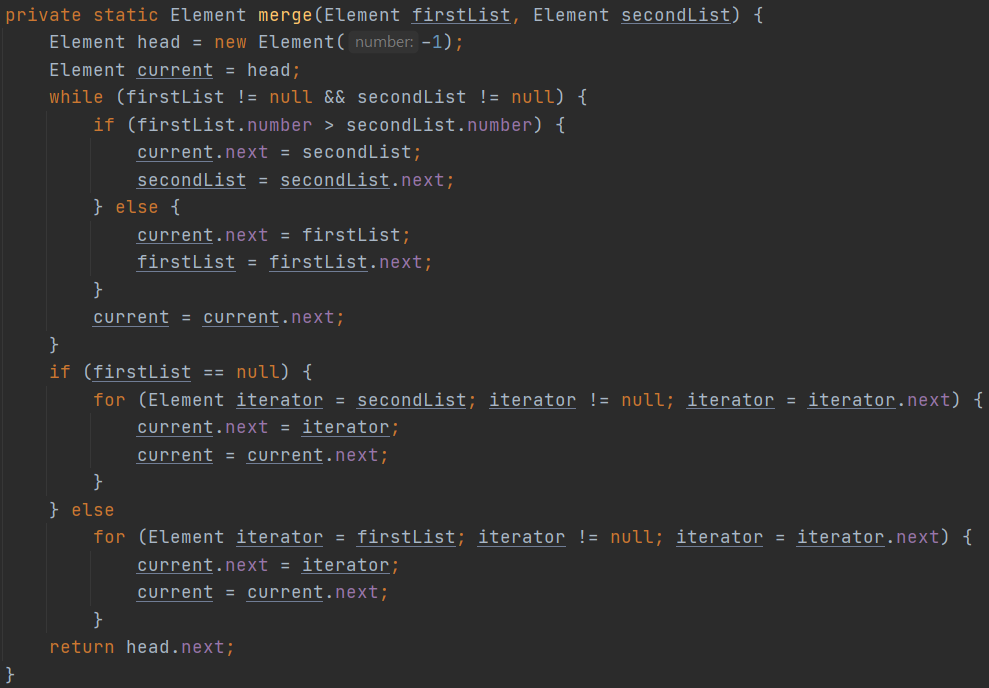


Рисунок 7 – Алгоритм слияния двух коллекций при сортировке слиянием

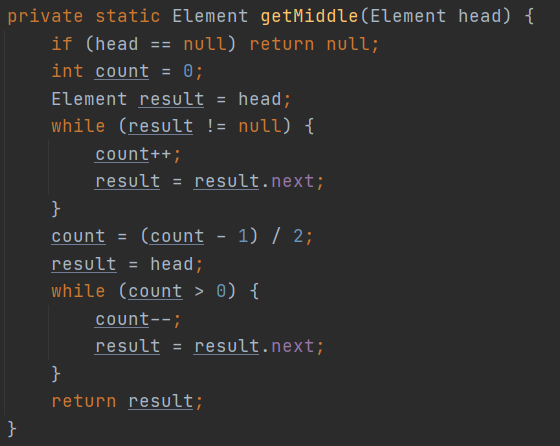


Рисунок 8 – Алгоритм получения среднего элемента в коллекции

Среднее количество сравнений: С= N\*Log2(N).

Недостатки:

- Большой стек вызовов, рекурсивной функции.

Преимущества:

- Более высокая скорость сортировки на больших данных, по сравнению с bubble-sort с квадратичной сложностью.

Сравнение времени работы merge-sort и bubble-sort на разных объемах данных (см. рисунок 9):

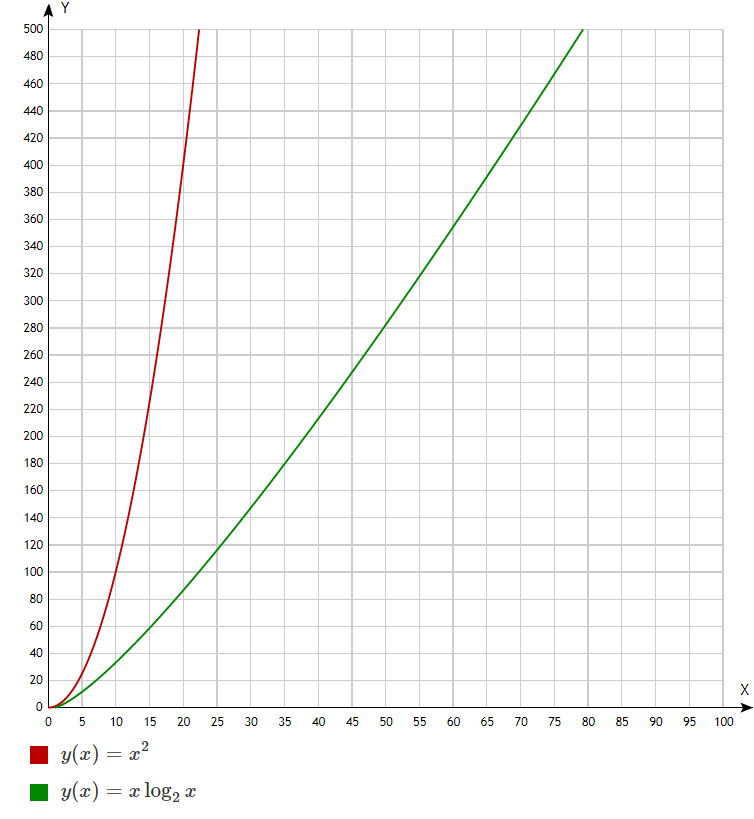


Рисунок 9 – Вычислительная сложность merge-sort (отмечена зеленым) и bubble-sort (отмечена красным)

Таблица 1 – Отладка программы

Последовательный поиск был реализован следующим образом (см. рисунок 2):

**2.2 Вывод**

По результатам доработки системы, все алгоритмы подходят для решения поставленной задачи.

Таблица 1 – Сравнение результатов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вариант | Структура | Поиск | Упорядочивание | Корректировка |
| Основной | Односвязный  Список  V=  4 +8\*N Байт | Последова тельный  C = (N+1)/2 | Пузырьком  C = N \* (N-1) | Удаление, путем  переназначения  указателя  Такты: 6 |
| Альтерна тивный | Односвязный  Список  V=  4 +8\*N Байт | Последова тельный  C = (N+1) / 2 | Слиянием  C = N\*log2(N) | Удаление, путем  переназначения  указателя  Такты: 6 |