|  |
| --- |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА – Российский технологический университет"**  **РТУ МИРЭА** |

Институт кибербезопасности и цифровых технологий

Кафедра КБ-2 «Институт кибербезопасности и цифровых технологий»

**ОТЧЕТ   
о выполнении домашней работе**

**«Реализация сортировки линейных структур данных»**

**по дисциплине   
«Технологии и методы программирования»**

**Вариант № 46**

Выполнил: студент 2 курса

группы БББО-01-21

Мысливец Л.В.

шифр 21Б0746

Москва 2022 г.

**Задание на № 1.**

В рамках домашней работы №1 требуется программно реализовать (с помощью массива (однонаправленных/двунаправленный динамический линейный связанный список, массива или используя стандартный контейнер библиотеки STL “stack” или «queue» - по варианту) абстрактный тип данных (АТД) в соответствии с заданием (стек, дек, очередь с одной головой, очередь с головой и хвостом).

Абстрактный тип данных должен позволять осуществлять только операции, присущие типу линейного связанного списка:

* получить значение первого элемента (на выходе),
* добавить элемент (на вход),
* удалить элемент из списка (на выходе),
* проверить – список пуст,
* обнулить (проинициализировать) список (конструктор, при необходимости).
* деструктор (при необходимости)

Используя разработанный АТД и указанный набор операций, необходимо реализовать заданный алгоритм сортировки последовательности элементов заданного типа, при этом следует учитывать, что разрешен доступ (чтение/извлечение) только к элементу на выходе.

На основе исходного текста программы получить аналитическую оценку трудоемкости работы алгоритма сортировки, используя О-символику для каждого реализованного метода АТД и сортировки в целом.

**Вариант № 46.**

**Реализация связи элементов линейного списка: Массив**

**Способ организации линейного связанный список: Стэк**

**Алгоритм сортировки: Естественное двухпутевое слияние**

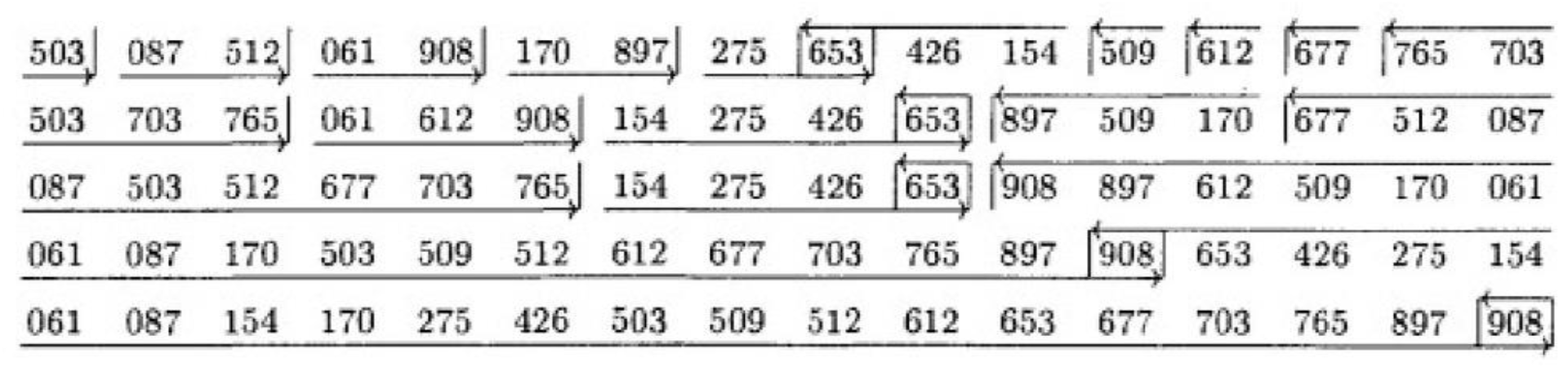
**Теория о сортировках.**

Естественное слияние – это сортировка, при которой всегда сливаются две самые длинные из возможных серий.

**Естественное двухпутевое слияние**

Этот алгоритм ищет упорядоченные отрезки с двух концов файла и переписывает их по очереди также в оба конца

Ниже приведена эта сортировка в действии:



Основное отличие простого (фиксированного) слияния от естественного заключается в следующем: в случае простого слияния частичная упорядоченность сортируемых данных не дает никакого преимущества. Это объясняется тем, что на каждом проходе сливаются серии фиксированной длины. При естественном слиянии длина серий не ограничивается, а определяется количеством элементов в уже упорядоченных подпоследовательностях, выделяемых на каждом проходе.

**Оценка трудоемкости алгоритма**

Все внутренние вершины порождают рекурсию, количество таких вершин = n-1, остальные n вершин – это вершины в которых рассматривается только один элемент стека, что приводит к прерыванию рекурсии.

Таким образом, для n листьев дерева выполняется сортировка слиянием c вычислением размера массива n, с проверкой условия, если n=1 и с возвратом массива в вызывающую функцию для слияния.

Для n-1 рекурсивных вершин выполняется проверка длины переданного массива, разбиение его на 2 части( 2 других массива), рекурсивный вызов функций слияния для них, и возврат их слияния.

Поскольку слияние вызывается n-1 раз с размерами стека равными n, n/2, n/4, ..., причем 2 раза с длиной n/2, 4 раза с длиной n/4, учитывая, что таким образом обрабатывается k-1 уровней, имеем:

O(f(n)) = n\*log2 𝑛. Трудоемкость естественного двухпутевого слияния аналогична.

Простыми словами Слияние работает за O(n), уровней всего log2 𝑛, поэтому асимптотика

O(n\*log2 𝑛). Стоит также помнить, что алгоритм требует дополнительно O(n) дополнительных затрат памяти для хранения копий массивов, перед их объединением.

**Листинг программы с расчетами.**

**public** **class** SortStack {

**static** Global global = **new** Global();

**public** **static** void main(**String** args[]) {

**for**(int i=300;i<=3000;i+=300) {

long start = **System**.currentTimeMillis();

**Stack** stack = **new** **Stack**();

**for** (int j = 0; j <= i; j++) {

stack.push(i - j);

}

sort(stack);

long finish = **System**.currentTimeMillis();

long elapsed = finish - start;

**System**.out.println("Прошло времени, мс: " + elapsed);

**System**.out.println("NOP="+global.nop);

global.nop = 0;

}

}

//1+4+2+2+n(2+3+8+3+2)+n+89n+nlogn+blogn=9+n(18)+n\*log(n)+n\*log(n)+89n=2nlogn+107n+9

**private** **static** void sort(**Stack** stack) {//1+4+2+

global.nop+=1;

**Stack** s1 = **new** **Stack**();

**Stack** s2 = **new** **Stack**();

global.nop+=4;

global.nop += 2;

**while** (stack.size() != 0) { // + 2 + n(2 + 3 + 8 + 3 + 2)

global.nop+=2;

global.nop += 3;

**if** (stack.size() % 2 == 0) {

s1.push(stack.pop());

global.nop+=2;

} **else** {

s2.push(stack.pop());

global.nop+=2;

}

}

global.nop += 2;

**if** (s1.size() > 1) {//n\*log(n)

global.nop+=1;

sort(s1);

}

global.nop += 2;

**if** (s2.size() > 1) {//n\*log(n)

global.nop += 1;

sort(s2);

}

global.nop += 4;

merge(s1, s2, stack);//+89n

}

**private** **static** void merge(**Stack** s1, **Stack** s2, **Stack** stack) {

global.nop+=3;

**Stack** mergedStack = **new** **Stack**();

global.nop+=2;

global.nop+=4;

**while** (!s1.isEmpty() && !s2.isEmpty()) {//2n(4+3

global.nop+=4;

global.nop+=3;

**if** ((**Integer**) s1.peek() < (**Integer**) s2.peek()) {//+5+3+8

global.nop+=3;

global.nop+=2;

mergedStack.push(s2.pop());

} **else** {

global.nop+=2;

mergedStack.push(s1.pop());

}

}

global.nop+=2;

**while** (!s1.isEmpty()) {//+n(+4+3+8)

global.nop+=2;

global.nop+=2;

mergedStack.push(s1.pop());

}

global.nop+=2;

**while** (!s2.isEmpty()) {//+n(+4+3+8)

global.nop+=2;

mergedStack.push(s2.pop());

}

global.nop+=2;

**while** (!mergedStack.isEmpty()) {//+n(2+3+8)

global.nop+=2;

stack.push(mergedStack.pop());

}

}

}

F(n)=

O(F(n))=

**Таблица результата экспериментов и графики зависимостей**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 300 | 37046 | 2469 | 0,004 | 169654 |
| 600 | 75284 | 5537 | 0,002 | 378946 |
| 900 | 113974 | 8832 | 0,003 | 601414 |
| 1200 | 152958 | 12275 | 0,003 | 836357 |
| 1500 | 192161 | 15826 | 0,002 | 1081246 |
| 1800 | 231539 | 19465 | 0,002 | 1324073 |
| 2100 | 271061 | 23176 | 0,086 | 1571333 |
| 2400 | 310707 | 26949 | 0,002 | 1832921 |
| 2700 | 350462 | 30777 | 0,003 | 2094328 |
| 3000 | 390313 | 34652 | 0,002 | 2355228 |

**Коэффициенты**

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 9261500 | 617250 | 0,218362078 | 0,014553149 |
| 37642000 | 2768500 | 0,198666829 | 0,014611581 |
| 37991333,33 | 2944000 | 0,189510055 | 0,014685391 |
| 50986000 | 4091666,667 | 0,182886016 | 0,014676747 |
| 96080500 | 7913000 | 0,177721814 | 0,014636817 |
| 115769500 | 9732500 | 0,174868757 | 0,014700851 |
| 3151872,093 | 269488,3721 | 0,172503855 | 0,014749261 |
| 155353500 | 13474500 | 0,169514671 | 0,014702761 |
| 116820666,7 | 10259000 | 0,16733864 | 0,014695406 |
| 195156500 | 17326000 | 0,16572196 | 0,014712801 |

**Скриншот работы программы:**



**Выводы.**

По результатам экспериментов было установлено, что графики C1, C2, C3, C4 от N имеют не линейную зависимость от количества элементов.

**Литература:**

1. Ахо А. В. Структуры данных и алгоритмы. – Издательский дом Вильямс, 2000.

2. Кнут Д. Э. Искусство программирования: Сортировка и поиск. – Издательский дом Вильямс, 2000. – Т. 3.

**Приложение 1. Применение счетчика операций N\_op.**

**Приложение 1.1 (класс сортировки слиянием)**

**public** **class** SortStack {

**static** Global global = **new** Global();

**public** **static** void main(**String** args[]) {

**for**(int i=300;i<=3000;i+=300) {

long start = **System**.currentTimeMillis();

**Stack** stack = **new** **Stack**();

**for** (int j = 0; j <= i; j++) {

stack.push(i - j);

}

sort(stack);

long finish = **System**.currentTimeMillis();

long elapsed = finish - start;

**System**.out.println("Прошло времени, мс: " + elapsed);

**System**.out.println("NOP="+global.nop);

global.nop = 0;

}

}

//1+4+2+2+n(2+3+8+3+2)+n+89n+nlogn+blogn=9+n(18)+n\*log(n)+n\*log(n)+89n=2nlogn+107n+9

**private** **static** void sort(**Stack** stack) {//1+4+2+

global.nop+=1;

**Stack** s1 = **new** **Stack**();

**Stack** s2 = **new** **Stack**();

global.nop+=4;

global.nop += 2;

**while** (stack.size() != 0) { // + 2 + n(2 + 3 + 8 + 3 + 2)

global.nop+=2;

global.nop += 3;

**if** (stack.size() % 2 == 0) {

s1.push(stack.pop());

global.nop+=2;

} **else** {

s2.push(stack.pop());

global.nop+=2;

}

}

global.nop += 2;

**if** (s1.size() > 1) {//n\*log(n)

global.nop+=1;

sort(s1);

}

global.nop += 2;

**if** (s2.size() > 1) {//n\*log(n)

global.nop += 1;

sort(s2);

}

global.nop += 4;

merge(s1, s2, stack);//+89n

}

**private** **static** void merge(**Stack** s1, **Stack** s2, **Stack** stack) {

global.nop+=3;

**Stack** mergedStack = **new** **Stack**();

global.nop+=2;

global.nop+=4;

**while** (!s1.isEmpty() && !s2.isEmpty()) {//2n(4+3

global.nop+=4;

global.nop+=3;

**if** ((**Integer**) s1.peek() < (**Integer**) s2.peek()) {//+5+3+8

global.nop+=3;

global.nop+=2;

mergedStack.push(s2.pop());

} **else** {

global.nop+=2;

mergedStack.push(s1.pop());

}

}

global.nop+=2;

**while** (!s1.isEmpty()) {//+n(+4+3+8)

global.nop+=2;

global.nop+=2;

mergedStack.push(s1.pop());

}

global.nop+=2;

**while** (!s2.isEmpty()) {//+n(+4+3+8)

global.nop+=2;

mergedStack.push(s2.pop());

}

global.nop+=2;

**while** (!mergedStack.isEmpty()) {//+n(2+3+8)

global.nop+=2;

stack.push(mergedStack.pop());

}

}

}

**Приложение 1.2 (класс реализации абстрактного типа данных “стэк”)**

**import** java.util.ArrayList;

**import** java.util.EmptyStackException;

**public** **class** Stack<T> **extends** ArrayList<T> {

Global global = **new** Global();

**public** void push(T t){ // 3

add(t);

global.nop=global.nop+3;

}

**public** T pop(){ // 8

int currentSize = size();

T t = get(currentSize - 1);

remove(currentSize-1);

global.nop=global.nop+8;

**return** t;

}

**public** T peek(){ // 7

int currentSize = size();

T t = get(currentSize - 1);

global.nop=global.nop+7;

**return** t;

}

**public** boolean empty() { // 3

global.nop=global.nop+3;

**return** size() == 0;

}

}

**Приложение 1.3(класс глобальной переменной)**

**public** **class** Global {

**public** **static** int nop;

}