|  |
| --- |
| 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 работы**

**«Реализация сортировки линейных структур данных»**

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

**Вариант № 35**

Выполнил: студент 2 курса

группы БББО-03-21

Кармазин В.С.

шифр 21Б-1335

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 № 35.**

**Реализация связи элементов линейного списка: Указатели**

**Способ организации линейного связанный список: Очередь**

**Алгоритм сортировки: Быстрая сортировка Хоару (с медианным (pivot) элементов)**

**Теория о сортировках.**

**Быстрая сортировка Хоару (с медианным (pivot) элементом).**

**Обзор быстрой сортировки**

Быстрая сортировка — эффективный алгоритм сортировки на месте, который обычно работает примерно в два-три раза быстрее, чем Сортировка слиянием а также сортировка кучей при хорошей реализации. Быстрая сортировка — это сортировка сравнением, то есть она может сортировать элементы любого типа, для которых *меньше, чем* отношение определено. В эффективных реализациях это обычно нестабильная сортировка.

Алгоритм быстрой сортировки является рекурсивным, поэтому для простоты процедура на вход будет принимать границы участка массива от l включительно и до r не включительно. Понятно, что для того, чтобы отсортировать весь массив, в качестве параметра l надо передать 0, а в качестве r — n, где по традиции n обозначает длину массива.  
 В основе алгоритма быстрой сортировке лежит процедура partition. Partition выбирает некоторый элемент массива и переставляет элементы участка массива таким образом, чтобы массив разбился на 2 части: левая часть содержит элементы, которые меньше этого элемента, а правая часть содержит элементы, которые больше или равны этого элемента. Такой разделяющий элемент называется пивотом.  
 Пивот в нашем случае берется как медиана. Такой алгоритм называется с медианным элементом. На самом деле пивот можно выбирать самым разным образом: либо брать случайный элемент, либо брать первый / последний элемент учаcтка, либо выбирать его каким-то «умным» образом. Выбор пивота является очень важным для итоговой сложности алгоритма сортировки, но об этом несколько позже. Сложность же процедуры partition — O(n), где n = r — l — длина участка.

**Анализ**  
 Предлагаю проанализировать данный алгоритм.  
 Временная сложность алгоритма выражается через нее же по формуле: T(n) = n + T(a \* n) + T((1 — a) \* n). Таким образом, когда мы вызываем сортировку массива из n элементов, тратится порядка n операций на выполнение partition'а и на выполнения себя же 2 раза с параметрами a \* n и (1 — a) \* n, потому что пивот разделил элемент на доли.  
 В нашем случае a = 1 / 2, то есть пивот каждый раз делит участок на две равные части (либо на две почти равные части, если у нас четное количество элементов). В таком случае: T(n) = n + 2 \* T(n / 2) = n + 2 \* (n / 2 + 2 \* T(n / 4)) = n + n + 4 \* T(n / 4) = n + n + 4 \* (n / 4 + 2 \* T(n / 8)) = n + n + n + 8 \* T(n / 8) =…. Итого будет log(n) слагаемых, потому как слагаемые появляются до тех пор, пока аргумент не уменьшится до 1. В результате T(n) = O(n \* log(n)).

Сортировка называется быстрой, потому что константа, которая скрывается под знаком O на практике оказывается достаточно небольшой, что привело к широкому распространению алгоритма на практике.

**Листинг программы с расчетами.**

class QueueWithOneHead **{**

public**:**

element**\*** head**;**

unsigned long int nop**;**

QueueWithOneHead**()** **{** //+2

head **=** 0**;**

nop **=** 0**;**

**}**

**~**QueueWithOneHead**()** **{**

**};**

void enQueue**(**int n**)** **{** // 2+3+1+4+(n-2)(1+2n+4)

nop **+=** 2**;**

**if** **(**head **==** 0**)** **{**

head **=** **new** element**;**

head**->**value **=** n**;**

head**->**next **=** 0**;**

nop **+=** 3**;**

**}**

**else** **if** **(**head**->**next **==** 0**)** **{**

element**\*** tmpElement **=** **new** element**;**

tmpElement**->**value **=** n**;**

tmpElement**->**next **=** 0**;**

head**->**next **=** tmpElement**;**

nop **+=** 4**;**

**}**

**else** **{**

element**\*** currentElement **=** head**->**next**;**

nop **+=** 1**;**

**while** **(**currentElement**->**next **!=** 0**)** **{**

currentElement **=** currentElement**->**next**;**

nop **+=** 2**;**

**}**

element**\*** tmpElement **=** **new** element**;**

tmpElement**->**value **=** n**;**

tmpElement**->**next **=** 0**;**

currentElement**->**next **=** tmpElement**;**

nop **+=** 4**;**

**}**

nop **+=** 1**;**

**}**

void showQueue**()** **{** //2+3n

element**\*** currentElement **=** head**;**

nop **+=** 1**;**

**while** **(**currentElement **!=** 0**)** **{**

cout **<<** currentElement**->**value **<<** " "**;**

currentElement **=** currentElement**->**next**;**

nop **+=** 3**;**

**}**

cout **<<** endl**;**

nop **+=** 1**;**

**}**

void indexing**()** **{** // 2+4n

element**\*** currentElement **=** head**;**

int i **=** 0**;**

nop **+=** 2**;**

**while** **(**currentElement **!=** 0**)** **{**

currentElement**->**index **=** i**;**

currentElement **=** currentElement**->**next**;**

i**++;**

nop **+=** 4**;**

**}**

**}**

void swapElements**(**int first\_ind**,** int second\_ind**)** **{** //4+n(3+1+n(3+5))

nop **+=** 2**;**

element**\*** currentElement **=** head**;**

nop **+=** 1**;**

**while** **(**currentElement **!=** 0**)** **{**

nop **+=** 3**;**

**if** **(**currentElement**->**index **==** first\_ind**)** **{**

element**\*** tmpElement **=** currentElement**;**

nop **+=** 1**;**

**while** **(**currentElement **!=** 0**)** **{**

nop **+=** 3**;**

**if** **(**currentElement**->**index **==** second\_ind**)** **{**

int tmp **=** 0**;**

tmp **=** tmpElement**->**value**;**

tmpElement**->**value **=** currentElement**->**value**;**

currentElement**->**value **=** tmp**;**

nop **+=** 5**;**

**return;**

**}**

currentElement **=** currentElement**->**next**;**

**}**

**}**

currentElement **=** currentElement**->**next**;**

**}**

cout **<<** "Элементов с такими индексами не существует"**;**

nop **+=** 1**;**

**}**

int findElement**(**int ind**)** **{** //2+4n

nop **+=** 1**;**

element**\*** currentElement **=** head**;**

nop **+=** 1**;**

**while** **(**currentElement **!=** 0**)** **{**

nop **+=** 3**;**

**if** **(**currentElement**->**index **==** ind**)** **{**

nop **+=** 1**;**

**return** currentElement**->**value**;**

**}**

currentElement **=** currentElement**->**next**;**

**}**

**}**

**};**

F(n)=

O(F(n))=

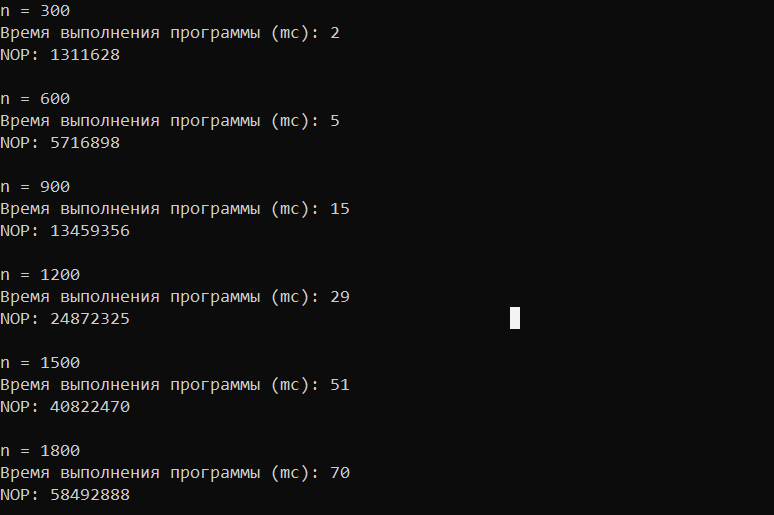
**Таблица результата экспериментов и графики зависимостей**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 300 | 907280 | 2469 | 0,002 | 1311628 |
| 600 | 3615148 | 5537 | 0,005 | 5716898 |
| 900 | 8123243 | 8832 | 0,015 | 13459356 |
| 1200 | 14431486 | 12275 | 0,029 | 24872325 |
| 1500 | 22539837 | 15826 | 0,051 | 40822470 |
| 1800 | 32448276 | 19465 | 0,07 | 58492888 |
| 2100 | 44156787 | 23176 | 0,097 | 79740439 |
| 2400 | 57665360 | 26949 | 0,13 | 107789566 |
| 2700 | 72973988 | 30777 | 0,187 | 139999201 |
| 3000 | 90082663 | 34652 | 0,215 | 176367244 |

**Коэффициенты**

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 453640000 | 1234500 | 0,691720518 | 0,001882393 |
| 723029600 | 1107400 | 0,632361816 | 0,000968532 |
| 541549533,3 | 588800 | 0,603538758 | 0,000656198 |
| 497637448,3 | 423275,8621 | 0,580222637 | 0,00049352 |
| 441957588,2 | 310313,7255 | 0,552142901 | 0,000387679 |
| 463546800 | 278071,4286 | 0,554738826 | 0,000332775 |
| 455224608,2 | 238927,8351 | 0,553756507 | 0,000290643 |
| 443579692,3 | 207300 | 0,534980909 | 0,000250015 |
| 390235229,9 | 164582,8877 | 0,521245746 | 0,000219837 |
| 418989130,2 | 161172,093 | 1,42948E-09 | 5,49878E-13 |

**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.**

// 35 вариант. Указатели, очередь с 1 головой, быстрая сортировка Хоару (с медианным (pivot) элементом)

#include <iostream>

#include <windows.h>

#include <time.h>

#include <math.h>

**using** **namespace** std**;**

struct element **{** //1+1+1

int value**;**

int index**;**

element**\*** next**;**

**};**

class QueueWithOneHead **{**

public**:**

element**\*** head**;**

unsigned long int nop**;**

QueueWithOneHead**()** **{** //+2

head **=** 0**;**

nop **=** 0**;**

**}**

**~**QueueWithOneHead**()** **{**

**};**

void enQueue**(**int n**)** **{** // 2+3+1+4+(n-2)(1+2n+4)

nop **+=** 2**;**

**if** **(**head **==** 0**)** **{**

head **=** **new** element**;**

head**->**value **=** n**;**

head**->**next **=** 0**;**

nop **+=** 3**;**

**}**

**else** **if** **(**head**->**next **==** 0**)** **{**

element**\*** tmpElement **=** **new** element**;**

tmpElement**->**value **=** n**;**

tmpElement**->**next **=** 0**;**

head**->**next **=** tmpElement**;**

nop **+=** 4**;**

**}**

**else** **{**

element**\*** currentElement **=** head**->**next**;**

nop **+=** 1**;**

**while** **(**currentElement**->**next **!=** 0**)** **{**

currentElement **=** currentElement**->**next**;**

nop **+=** 2**;**

**}**

element**\*** tmpElement **=** **new** element**;**

tmpElement**->**value **=** n**;**

tmpElement**->**next **=** 0**;**

currentElement**->**next **=** tmpElement**;**

nop **+=** 4**;**

**}**

nop **+=** 1**;**

**}**

void showQueue**()** **{** //2+3n

element**\*** currentElement **=** head**;**

nop **+=** 1**;**

**while** **(**currentElement **!=** 0**)** **{**

cout **<<** currentElement**->**value **<<** " "**;**

currentElement **=** currentElement**->**next**;**

nop **+=** 3**;**

**}**

cout **<<** endl**;**

nop **+=** 1**;**

**}**

void indexing**()** **{** // 2+4n

element**\*** currentElement **=** head**;**

int i **=** 0**;**

nop **+=** 2**;**

**while** **(**currentElement **!=** 0**)** **{**

currentElement**->**index **=** i**;**

currentElement **=** currentElement**->**next**;**

i**++;**

nop **+=** 4**;**

**}**

**}**

void swapElements**(**int first\_ind**,** int second\_ind**)** **{** //4+n(3+1+n(3+5))

nop **+=** 2**;**

element**\*** currentElement **=** head**;**

nop **+=** 1**;**

**while** **(**currentElement **!=** 0**)** **{**

nop **+=** 3**;**

**if** **(**currentElement**->**index **==** first\_ind**)** **{**

element**\*** tmpElement **=** currentElement**;**

nop **+=** 1**;**

**while** **(**currentElement **!=** 0**)** **{**

nop **+=** 3**;**

**if** **(**currentElement**->**index **==** second\_ind**)** **{**

int tmp **=** 0**;**

tmp **=** tmpElement**->**value**;**

tmpElement**->**value **=** currentElement**->**value**;**

currentElement**->**value **=** tmp**;**

nop **+=** 5**;**

**return;**

**}**

currentElement **=** currentElement**->**next**;**

**}**

**}**

currentElement **=** currentElement**->**next**;**

**}**

cout **<<** "Элементов с такими индексами не существует"**;**

nop **+=** 1**;**

**}**

int findElement**(**int ind**)** **{** //2+4n

nop **+=** 1**;**

element**\*** currentElement **=** head**;**

nop **+=** 1**;**

**while** **(**currentElement **!=** 0**)** **{**

nop **+=** 3**;**

**if** **(**currentElement**->**index **==** ind**)** **{**

nop **+=** 1**;**

**return** currentElement**->**value**;**

**}**

currentElement **=** currentElement**->**next**;**

**}**

**}**

**};**

class QuickSort **{** //n\*(log(n))

public**:**

unsigned int long nop**;**

void hoarasort**(**QueueWithOneHead**\*** Q**,** int first**,** int last**)** **{**

nop **+=** 5**;**

int i **=** first**,** j **=** last**;**

int temp **=** 0**;**

nop **+=** 3**;**

int pivot **=** Q**->**findElement**((**first **+** last**)** **/** 2**);**

nop **+=** 3**;**

**while** **(**i **<=** j**)** **{**

nop **+=** 1**;**

**while** **(**Q**->**findElement**(**i**)** **<** pivot**)** **{**

i**++;**

nop **+=** 2**;**

**}**

**while** **(**Q**->**findElement**(**j**)** **>** pivot**)** **{**

j**--;**

nop **+=** 2**;**

**}**

**if** **(**i **<=** j**)** **{**

nop **+=** 1**;**

**if** **(**i **<** j**)** **{**

Q**->**swapElements**(**i**,** j**);**

nop **+=** 3**;**

**}**

i**++;**

j**--;**

nop **+=** 2**;**

**}**

**}**

**if** **(**i **<** last**)** **{**

nop **+=** 1**;**

hoarasort**(**Q**,** i**,** last**);**

**}**

**if** **(**first **<** j**)** **{**

nop **+=** 1**;**

hoarasort**(**Q**,** first**,** j**);**

**}**

**}**

**};**

int main**()** **{**

SetConsoleCP**(**1251**);**

SetConsoleOutputCP**(**1251**);**

QuickSort**\*** Sort **=** **new** QuickSort**;**

int first **=** 0**;**

int last**;**

Sort**->**nop **=** 0**;**

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

QueueWithOneHead**\*** Queue **=** **new** QueueWithOneHead**;**

Queue**->**nop **=** 0**;**

**for** **(**int b **=** 0**;** b **<=** a**;** b**++)** **{**

Queue**->**enQueue**(**a **-** b**);**

**}**

unsigned int start\_time **=** 0**;**

unsigned int end\_time **=** 0**;**

unsigned int final\_time **=** 0**;**

int last **=** a**;**

Queue**->**indexing**();**

cout **<<** "n = " **<<** a **<<** endl**;**

start\_time **=** clock**();**

Sort**->**hoarasort**(**Queue**,** first**,** last**);**

end\_time **=** clock**();**

final\_time **=** end\_time **-** start\_time**;**

cout **<<** "Время выполнения программы (mc): " **<<** final\_time **<<** endl**;**

cout **<<** "NOP: " **<<** Queue**->**nop**+**Sort**->**nop**;**

cout **<<** endl **<<** endl**;**

Queue**->**nop **=** 0**;**

Sort**->**nop **=** 0**;**

**}**

**return** 0**;**

**}**