**Министерство образования и науки РФ.**

**ФГБОУ ВПО**

**«Московский государственный университет приборостроения и**

**информатики»**

Факультет ИТ направление 231000.62 «Программная инженерия»

Кафедра ИТ – 6 «Управление и моделирование систем»

Дисциплина « Структуры и алгоритмы обработки данных »

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе на тему:**

Программная реализация абстрактного типа данных

Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ .06.2013\_\_\_\_\_\_\_\_\_\_\_\_Дмитриев Д.В.\_\_

подпись, дата инициалы и фамилия

Группа ИТ6-1101 шифр 110088

Обозначение курсовой работы КР-2068752-231000.62-ИТ6-13

Работа защищена на оценку\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Филатов В.В.\_\_\_\_

подпись, дата инициалы и фамилия

Член комиссии\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата инициалы и фамилия

МОСКВА 2013г.

**Министерство образования и науки РФ.**

**ФГБОУ ВПО**

**«Московский государственный университет приборостроения и**

**информатики»**

Факультет ИТ направление 231000.62 «Программная инженерия»

Кафедра ИТ – 6 «Управление и моделирование систем»

Дисциплина «Структуры и алгоритмы обработки данных»

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Студент Дмитриев Д.В. шифр\_\_110088\_\_\_\_\_\_\_\_группа\_\_\_ИТ6-1101\_\_\_\_\_\_\_\_\_\_\_\_\_

1 Тема:\_\_ Программная реализация абстрактного типа данных\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2 Срок предоставления проекта (работы к защите)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2013г

3 Исходные данные для разработки – \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4 Содержание пояснительной записки:

Титульный лист

Задание

Содержание

Введение

Теоретические сведения

Текст программы

Распечатка выходной информации

Получение аналитической оценки трудоемкости работы алгоритма сортировки

Таблица значений n, T(n), f(n), O(f(n)), Nop, C1, C2, C3, C4

Заключение

Список использованных источников

5 Перечень графического материала:\_\_\_\_\_графики зависимости: С1 С2 С3 С4 \_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель проекта (работы)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Филатов В.В. \_\_\_\_

подпись, дата инициалы и фамилия

Задание принял к исполнению\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата инициалы и фамилия

*Оглавление*

[Введение 4](#_Toc359371519)

[1 Задание 4](#_Toc359371520)

[1.1 Постановка задачи 4](#_Toc359371521)

[1.2 Задание в соответствии с вариантом 4](#_Toc359371522)

[2 Теоретические сведения 5](#_Toc359371523)

[3 Текст программы 8](#_Toc359371524)

[3.1 MyStack.h 8](#_Toc359371525)

[3.2 MyStack.cpp 8](#_Toc359371526)

[3.3 Main.cpp 9](#_Toc359371527)

[4 Распечатка выходной информации 10](#_Toc359371528)

[5 Получение аналитической оценки трудоемкости работы алгоритма сортировки. 11](#_Toc359371529)

[6 Таблица значений n, T(n), f(n), O(f(n)), Nop, C1, C2, C3, C4. 12](#_Toc359371530)

[Заключение 15](#_Toc359371531)

[Список использованных источников 16](#_Toc359371532)

# Введение

# Задание

## Постановка задачи

В рамках курсовой работы по дисциплине «Структуры и алгоритмы обработки данных» требуется программно реализовать абстрактный тип данных (АТД) в соответствии с заданием. Абстрактный тип данных должен позволять осуществлять только операции, присущие типу линейного связанного списка: получить значение первого элемента (на выходе), добавить элемент (в конец списка), удалить элемент из списка (на выходе), проверить – список пуст, обнулить (проинициализировать) список. Используя разработанный АТД и указанный набор операций, необходимо реализовать заданный алгоритм сортировки последовательности динамических элементов. Реализацию алгоритма осуществить на языке Паскаль или Си.

На основе предложенной в рамках курсовой работы программы получить аналитическую оценку трудоемкости работы алгоритма сортировки, используя О-символику.

## Задание в соответствии с вариантом

Вариант 44.

|  |  |  |  |
| --- | --- | --- | --- |
| **№ варианта** |  | | **Алгоритм сортировки** |
| **Реализация связи элементов линейного списка** | **Способ организации линейного связанный список** |
| 44. | **Указатели** | **Стек** | **Фиксированное двухпутевое слияние** |

# Теоретические сведения

Абстрактный тип данных – это математическая модель с совокупностью операторов или операций определенных в рамках данной модели. В языке программирования под типом данных понимают как понятие, определяющее диапазон значений, который может принимать данная переменная , которому соответствует достаточный объем памяти для хранения, выделяемый компилятором.

Для представления АТД используются структуры данных*,* которые представляют собой набор переменных, возможно, различных типов данных, объединенных определенным образом. Под структурой данных в общем случае понимают множество элементов данных и множество связей между ними. Такое определение охватывает все возможные подходы к структуризации данных, но в каждой конкретной задаче используются те или иные его аспекты.

Важный признак составной структуры данных – характер упорядоченности ее составных частей. По этому признаку структуры можно делить на линейные и нелинейные структуры.

Весьма важный признак структуры данных – ее изменчивость, то есть изменение числа элементов и/или связей между составными частями структуры. В определении изменчивости структуры не отражен факт изменения значений элементов данных, поскольку в этом случае все структуры данных имели бы свойство изменчивости. По признаку изменчивости различают структуры статические и динамические.

В процессе решения прикладных задач выбор подходящего алгоритма вызывает определенные трудности. Прежде чем принимать решение об использовании того или иного алгоритма, необходимо оценить сложность и эффективность этого алгоритма. Сложность алгоритма – это величина, отражающая порядок величины требуемого ресурса (времени или дополнительной памяти) в зависимости от размерности задачи.

Различается время выполнения программы, которое можно измерить в секундах (миллисекундах, аппаратных тактах центрального процессора) и время выполнения соответствующего ей алгоритма, которое измеряется числом инструкций, которые необходимо выполнить для получения требуемого результата.

Временная сложность алгоритма зависит от количества входных данных. Обычно говорят, что временная сложность алгоритма имеет порядок *T*(*n*) от входных данных размера *n*. Точно определить величину *T*(*n*) на практике представляется довольно трудно. Поэтому прибегают к асимптотическим отношениям с использованием *O*-символики, которая дает приемлемую оценку времени выполнения алгоритма для не бесконечно больших и не бесконечно малых значений *n*. Для предварительной оценки трудоемкости алгоритмов используется нотация «большое О» . Основная идея применения нотации O заключается в том, что необходимо определить, как будет вести себя алгоритм при росте размера n. O-символика дает аппроксимированную верхнюю границу трудоемкости алгоритма при неограниченном росте n.

Теоретическая оценка временной сложности алгоритма осуществляется с использованием следующих базовых принципов оценки сложности выполнения основных алгоритмических конструкций:

Время выполнения операторов присваивания, чтения и записи обычно имеет порядок О(1).

Время выполнения последовательности операторов определяется с помощью правила сумм.

Время выполнения условных операторов состоит из времени выполнения условно исполняемых операторов и времени вычисления самого логического выражения.

Время выполнения цикла является суммой времени всех исполняемых итераций цикла, в свою очередь состоящих из времени выполнения операторов тела цикла и времени вычисления условия прекращения цикла.

Для программ, содержащих несколько процедур (функций) (среди которых нет рекурсивных), можно подсчитать общее время выполнения программы путем последовательного нахождения времени выполнения процедур, начиная с той, которая не имеет вызовов других процедур.

Если есть рекурсивные процедуры, то в этом случае с каждой рекурсивной процедурой нужно связать временн*у*ю- функцию *Т*(*п*)*,* где *п* определяет объем аргументов процедуры. Затем получить рекуррентное соотношениедля *Т*(*п*)*.*

Сортировка слиянием - алгоритм сортировки, который упорядочивает списки (или другие структуры данных, доступ к элементам которых можно получать только последовательно, например — потоки) в определённом порядке. Эта сортировка — хороший пример использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получается решение исходной задачи.

Пример сортировки слиянием. Сначала делим список на кусочки (по 1 элементу), затем сравниваем каждый элемент с соседним, сортируем и объединяем. В итоге, все элементы отсортированы и объединены вместе.

Для решения задачи сортировки эти три этапа выглядят так:

* Сортируемый массив разбивается на две части примерно одинакового размера;
* Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
* Два упорядоченных массива половинного размера соединяются в один.

Алгоритм был изобретён Джоном фон Нейманом в 1945 году.

# Текст программы

## MyStack.h

#ifndef MYSTACK\_H

#define MYSTACK\_H

struct Node{

int data;

Node \*next;

};

class MyStack{

Node \*Head;

int count;

public:

MyStack();

void Push(int val);

int Pop();

bool Empty();

int GetTop();

void Init(int n);

int GetCount();

};

#endif // MYSTACK\_H

## MyStack.cpp

#include "mystack.h"

#include<cstdlib>

#include <time.h>

#include <iostream>

using namespace std;

MyStack::MyStack(){

Head = NULL;

count = 0;

}

void MyStack::Push(int val){

Node \*node = new Node;

node->data = val;

node->next = Head;

Head = node;

count ++;

}

int MyStack::Pop(){

int val = Head->data;

Node \*node = Head;

Head = node->next;

delete node;

count --;

return val;

}

int MyStack::GetTop(){

if (!Empty())

return Head->data;

cout << "стек пуст!"<<endl;

return 0;

}

bool MyStack::Empty(){

return Head == NULL;

}

void MyStack::Init(int n){

srand (time(NULL));

for(int i = 0 ; i < n ; i++)

Push(rand()%90);

}

int MyStack::GetCount(){

return count;

}

## Main.cpp

#include <iostream>

#include "mystack.h"

#include <time.h>

using namespace std;

MyStack Merge(MyStack A,MyStack B){

MyStack C,Reversed;

while (!A.Empty()||!B.Empty())

if (B.Empty()||(!A.Empty()&&A.GetTop()<=B.GetTop()))

Reversed.Push(A.Pop());

else

Reversed.Push(B.Pop());

while (!Reversed.Empty())

C.Push(Reversed.Pop());

return C;

}

MyStack MergeSort(MyStack A){

int n = A.GetCount();

if (n == 1)

return A;

MyStack left,right;

for (int i = 0 ; i < n/2 ; i++)

left.Push(A.Pop());

for (int i = n/2 ; i < n ; i++)

right.Push(A.Pop());

left = MergeSort(left);

right = MergeSort(right);

return Merge(left,right);

}

int main()

{

MyStack a,b;

a.Init(50);

while (!a.Empty()){

b.Push(a.GetTop());

cout << a.Pop() <<" ";

}

cout << endl;

b = MergeSort(b);

while (!b.Empty())

cout << b.Pop() <<" ";

cout << endl;

return 0;

}

# Распечатка выходной информации

Исходная последовательность (50 элементов):

83 84 53 23 48 86 84 60 74 15 60 11 68 4 83 68 73 71 40 67 19 61 65 79 16 17 60 17 58 38 81 44 88 71 27 72 64 64 60 5 45 14 6 54 8 85 9 51 70 76

Отсортированная последовательность:

4 5 6 8 9 11 14 15 16 17 17 19 23 27 38 40 44 45 48 51 53 54 58 60 60 60 60 61 64 64 65 67 68 68 70 71 71 72 73 74 76 79 81 83 83 84 84 85 86 88

Исходная последовательность (10 элементов):

44 78 16 6 70 58 14 4 87 23

Отсортированная последовательность:

4 6 14 16 23 44 58 70 78 87

Исходная последовательность (32 элемента):

37 42 68 12 6 50 33 87 1 33 28 87 58 57 1 51 31 35 62 25 87 36 73 89 32 21 39 10 53 9 54 4

Отсортированная последовательность:

1 1 4 6 9 10 12 21 25 28 31 32 33 33 35 36 37 39 42 50 51 53 54 57 58 62 68 73 87 87 87 89

# Получение аналитической оценки трудоемкости работы алгоритма сортировки.

Трудоемкость слияния

Fmerge(m) = 6 + 28m (1)

Где m – размер слитой последовательности

Алгоритм, получая на входе стек размером n элементов, делит его пополам на 2 других стека при первом вызове, поэтому рассмотрим случай, когда n=http://th-algoritmov.narod.ru/img/2k.gif, k =http://th-algoritmov.narod.ru/img/log2n.gifВ этом случае мы имеем полное дерево рекурсивных вызовов глубиной k, содержащее n листьев, фрагмент дерева показан на рисунке 1.

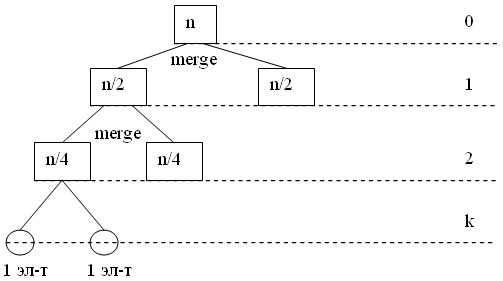


Рисунок 1 Фрагмент рекурсивного дерева при сортировке слиянием

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 вызов процедуры MergeSort c вычислением размера стека n,с проверкой условия if (n == 1) и с возвратом стека в вызывающую функцию для слияния, что даёт:

Fmergesort1(n) = 6n (2)

Для n-1 рекурсивных вершин выполняется проверка длины переданного стека, разбиение стека на 2 части( 2 других стека), рекурсивный вызов функций MergeSort для них, и возврат их слияния. С учетом (1) получаем:

Fmergesort(n) = 7\*(n-1) + 14m= 7n+14m-7 (3)

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:

f(n) = 6\*(n - 1) + 28n + 7n - 7 +14n+6n =42 n + 19n -13

O(f(n)) = n

# Таблица значений n, T(n), f(n), O(f(n)), Nop, C1, C2, C3, C4.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | F(n) | O(f(n)) | T(n) | N\_op | C1 | C2 | C3 | C4 |
| 1000 | 437549,94 | 9965,78428 | 0,003 | 437979 | 145849980 | 3321928,095 | 0,999020364 | 0,022754023 |
| 2000 | 959112,8799 | 21931,5686 | 0,005 | 959971 | 191822576 | 4386313,714 | 0,999106098 | 0,022846074 |
| 3000 | 1512381,095 | 34652,2404 | 0,009 | 1522955 | 168042343,9 | 3850248,928 | 0,993056981 | 0,022753292 |
| 4000 | 2086238,76 | 47863,1371 | 0,012 | 2087955 | 173853230 | 3988594,762 | 0,999178028 | 0,022923452 |
| 5000 | 2675406,6 | 61438,5619 | 0,016 | 2690923 | 167212912,5 | 3839910,119 | 0,9942338 | 0,02283178 |
| 6000 | 3276775,19 | 75304,4807 | 0,02 | 3297923 | 163838759,5 | 3765224,036 | 0,993587537 | 0,022833911 |
| 7000 | 3888289,927 | 89411,9744 | 0,023 | 3904923 | 169056083,8 | 3887477,15 | 0,995740486 | 0,022897244 |
| 8000 | 4508490,52 | 103726,274 | 0,027 | 4511923 | 166981130,4 | 3841713,862 | 0,999239242 | 0,022989372 |
| 9000 | 5136285,11 | 118221,384 | 0,03 | 5152859 | 171209503,7 | 3940712,786 | 0,996783555 | 0,022942872 |
| 10000 | 5770826,199 | 132877,124 | 0,034 | 5801859 | 169730182,3 | 3908150,7 | 0,994651232 | 0,022902508 |

На основе полученных данных построены графики зависимости:

C1= (f(n)/T(n)) – график отражает отношение функции роста f(n) ко времени T(n)

C2 =(O(f(n))/T(n)) - график отражает отношение порядка роста O(f(n)) ко времени T(n)

C3 =(f(n)/Nop) - график отражает отношение функции роста f(n) к количеству выполненных операций

C4 (O(f(n))/Nop) - график отражает отношение порядка роста O(f(n)) к количеству выполненных операций.

# Заключение

В результате выполнения данного задания была реализована сортировка стека алгоритмом фиксированного двухпутевого слияния. Был проведен анализ алгоритма. Данный алгоритм имеет множество преимуществ. Он легок в реализации и в то же время является одним из самых быстрых алгоритмов сортировки. Благодаря тому, что в алгоритме используется последовательный доступ к элементам, он имеет широкий спектр типов входных данных. Единственным недостатком алгоритма слияния является чрезмерное выделение дополнительной памяти.

# 

# Список использованных источников

1. Алгоритмы: построение и анализ. Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М.: МЦНМО, 2002.

2. Кнут Д. Э. Искусство программирования: В 3-х т. /Пер. с англ.-3-е изд.-М.: Вильямс.- (труд). Т. 1: Основные алгоритмы.-2004.-712 с.

3. Кнут Д. Э. Искусство программирования: В 3-х т. /Пер. с англ.-3-е изд.-М.: Вильямс.- (труд). Т. 2: Получисленные алгоритмы.-2004.-828 с.

4. Кнут Д. Э. Искусство программирования: В 3-х т. /Пер. с англ.-2-е изд.-М.: Вильямс.- (труд). Т. 3: Сортировка и поиск.-2004.-822 с.: ил.

5. Макконелл Дж. Основы современных алгоритмов: Учеб. пособие для вузов/Пер. с англ.-2-е изд., доп.-М.: Техносфера, 2004.-366 с.: ил.- (программирования).

6. Структуры данных и алгоритмы. Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман. – М.: Издательский дом «Вильямс», 2006.-384 с.

7. Левитин А.В.Алгоритмы: введение в разработку и анализ.: Пер. С англ.- М.: Издательский дом «Вильямс»,2006.-576 с.