Федеральное государственное бюджетное образовательное учреждение

высшего образования «Сибирский государственный университет телекоммуникаций и информатики»

Факультет ИВТ

Кафедра вычислительных систем

**Курсовая работа**

на тему «Обработка последовательной информации»

Вариант 1.2 «Разработка библиотеки сортировок:

сортировка вставками и пирамидальная сортировка»

Выполнил:студент гр. ИС-241Рыбалко В.А.

Проверил:старший преподаватель Кафедры ВС

Фульман В.О

Новосибирск, 2023

**ОГЛАВЛЕНИЕ**

ТЕМА3

ЗАДАНИЕ4

АНАЛИЗ ЗАДАЧИ5

ОБЩАЯ ИНФОРМАЦИЯ ПРО СОРТИРОВКИ5

ВЫПОЛНЕНИЕ РАБОТЫ6

ХОД РАБОТЫ ПРОГРАММЫ6

ПРОВЕРКА НА КОРРЕКТНОСТЬ6

АЛГОРИТМ РАБОТЫ СОРТИРОВОК6

ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ8

ПРИЛОЖЕНИЕ10

sort.c10

**ТЕМА**

ОБРАБОТКА ПОСЛЕДОВАТЕЛЬНОЙ ИНФОРМАЦИИ

В рамках данной главы рассматриваются различные задачи обработки последовательностей.

Получение информации через аргументы командной строки

В большинстве заданий входные данные должны передаваться через аргументы командной строки. Эта информация является частью «окружения» программы. Все аргументы командной строки передаются в виде строк и доступны через формальные параметры функции main. Прототип функции main в программе, использующей аргументы командной строки, выглядит следующим образом:

|  |
| --- |
| int main(int argc, char \*\*argv) |

где argc – количество строк-аргументов, а argv – массив указателей на сами строки-аргументы.

Рассмотрим следующий пример:

|  |
| --- |
| $ ./prog first\_arg “second arg” third arg |

На вход программе будет передана информация, показанная на рисунке:

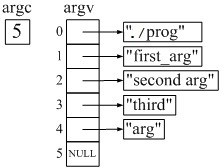


Рисунок 2: Аргументы командной строки

Обратите внимание, что в командной строке пробел является разделителем, поэтому строка third arg разбивается на два аргумента. Если требуется, чтобы входной аргумент содержал пробелы – строку необходимо заключать в кавычки, например, *"second arg"*. Первым элементом *argv* (*argv*[*0*]) всегда является имя исполняемого файла программы. Обработка входных аргументов может производиться напрямую через взаимодействие с массивом *argv*, однако существует ряд функций, позволяющих упростить процесс разбора входных параметров. Это функции *getopt* и *getopt\_long*. Более подробную информацию об использовании этих функций можно получить в справочном руководстве ОС GNU/Linux – "*man 3 getopt*".

# **ЗАДАНИЕ**

Реализовать динамическую библиотеку сортировок. Алгоритмы сортировок выбираются в соответствии с вариантом задания. Проанализировать эффективность алгоритмов сортировки. Разработать демонстрационную программу, использующую созданную библиотеку.

Критерии оценки

* **Оценка «удовлетворительно»:** алгоритмы реализованы в виде простой программы без применения библиотек, данные поступают на вход с клавиатуры. Тесты проведены только на небольших последовательностях, нет анализа и сравнения алгоритмов. Не предусмотрено динамическое выделение памяти под входные данные.
* **Оценка «хорошо»:** работа выполнена в полном соответствии с заданием. Обязательно динамическое выделение памяти под входные данные.
* **Оценка «отлично»:** помимо выполнения условий задания предусмотрена сортировка произвольных данных (по аналогии с функцией qsort библиотеки GNU C Library – GLibC). Обязательно динамическое выделение памяти под входные данные.

Указания к выполнению задания

Алгоритмы сортировки необходимо реализовать в подпрограммах. Подпрограммы выносятся в отдельную библиотеку, которая компилируется как динамическая. Информация о создании и использовании динамических библиотек может быть найдена на ресурсе FirstSteps: http://firststeps.ru/linux/general1.html.

Эффективность сортировок оценивать по времени работы алгоритмов. По полученным результатам сформулировать выводы о преимуществах и недостатках каждого алгоритма. Сравнить полученные результаты с теоретическими оценками вычислительной сложности реализованных алгоритмов.

Экспериментальные измерения необходимо провести как для упорядоченных данных (по возрастанию и по убыванию), так и случайных последовательностей, размер которых составляет 28 – 215 элементов (с некоторым шагом). Построить графики полученных зависимостей. В случае, если время работы одного из алгоритмов превышает 15 мин., прекратить измерения по данному алгоритму и строить график не на всем интервале.

|  |  |
| --- | --- |
| Вариант | Алгоритмы сортировок |
| 1.1 | Сортировка методом пузырька, быстрая сортировка |
| 1.2 | Сортировка вставками, пирамидальная сортировка |
| 1.3 | Сортировка Шелла, сортировка слиянием |

Таблица 1. Распределение вариантов задания по видам сортировок

# **АНАЛИЗ ЗАДАЧИ**

Курсовая работа выполняется на оценку «удовлетворительно», поэтому:

1. программа реализована внутри одного файла
2. алгоритмы сортировок не вынесены в библиотеку
3. тесты проводятся на небольших произвольных числовых последовательностях, поступающих на вход с клавиатуры
4. не происходит динамического выделения памяти под входные данные
5. отсутствует анализ и сравнение эффективности алгоритмов

ОБЩАЯ ИНФОРМАЦИЯ ПРО СОРТИРОВКИ

Проходим по массиву слева направо и обрабатываем по очереди каждый элемент. Слева от очередного элемента наращиваем отсортированную часть массива, справа по мере процесса потихоньку испаряется неотсортированная. В отсортированной части массива ищется точка вставки для очередного элемента. Сам элемент отправляется в буфер, в результате чего в массиве появляется свободная ячейка — это позволяет сдвинуть элементы и освободить точку вставки

Пирамидальная сортировка (или сортировка кучей) — это метод сортировки сравнением, основанный на такой структуре данных как двоичная куча. Она похожа на сортировку выбором, где мы сначала ищем максимальный элемент и помещаем его в конец. Далее мы повторяем ту же операцию для оставшихся элементов.

# **ВЫПОЛНЕНИЕ РАБОТЫ**

ХОД РАБОТЫ ПРОГРАММЫ

1. Введение пользователем длины массива
2. Проверка ввода пользователя на корректность
3. Создание целочисленного массива заданной длины
4. Введение пользователем массива
5. Проверка ввода пользователя на корректность
6. Вывод программой неотсортированного массива
7. Создание отдельных массивов под обе сортировки
8. Копирование начального массива в те, в которых будут происходить сортировки
9. Запуск сортировки вставками
10. Вывод результата работы сортировки вставками
11. Запуск пирамидальной сортировки
12. Вывод результата работы пирамидальной сортировки
13. Завершение работы программы

ПРОВЕРКА НА КОРРЕКТНОСТЬ

Проверка осуществляется с помощью конструкции if(scanf(“%d”, …) != 1). Функция scanf() возвращает количество полей, которым были действительно присвоены введённые значения. В данном случае функция при получении на ввод чего-либо, кроме целочисленного числа, не сможет присвоить введённое значение переменной, и вернёт ноль. Таким образом проходить проверку на корректность будут только целые числа.

АЛГОРИТМ РАБОТЫ СОРТИРОВОК

1. Сортировка вставками

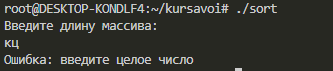
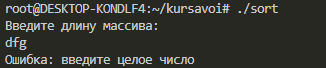
Функция insertion\_sort(int\* arr, int n) представляет собой алгоритм сортировки вставками для массива целых чисел. Алгоритм состоит в том, чтобы перебирать каждый элемент массива, начиная со второго (i=1), сравнивать его со всеми предыдущими элементами и вставлять его в нужное место в упорядоченном списке.  
  
На каждом шаге цикла (for) происходит выбор текущего элемента массива (key) и его сравнение с элементами, находящимися перед ним (while). Если находится элемент, который больше текущего, то он сдвигается вправо, а текущий элемент вставляется на его место.  
  
Алгоритм завершается, когда все элементы массива просмотрены и отсортированы в порядке возрастания.

1. Пирамидальная сортировка

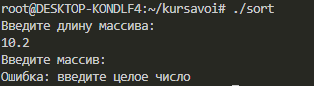
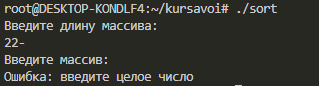
Это две функции для выполнения пирамидальной сортировки на массиве целых чисел. Пирамидальная сортировка - это алгоритм сортировки, который использует двоичное дерево, называемое кучей, чтобы отсортировать элементы.  
  
1. Функция heapify(int\* arr, int n, int i):  
Эта функция превращает поддерево с корнем в узле `i` в максимальную кучу, при условии что поддеревья с корнями l (left) и r (right) уже являются максимальными кучами. arr - это указатель на массив, который содержит элементы для сортировки, n - размер массива, а i - индекс узла, который должен быть преобразован в максимальную кучу. Функция выполняет проверку и свап элементов в массиве, чтобы сделать максимальную кучу.  
  
2. Функция heap\_sort(int\* arr, int n):  
Эта функция выполняет сортировку кучей для входного массива arr размера n. Вначале она создает максимальную кучу из исходного массива, вызывая функцию heapify для всех узлов, начиная с середины массива и заканчивая корнем кучи. Затем она перемещает максимальный элемент кучи (корневой узел) в конец отсортированной части массива, свапая его с последним элементом в неотсортированной части. Затем размер кучи уменьшается на 1, и вызывается функция heapify для корневого узла с обновленным размером кучи, и процесс повторяется до тех пор, пока вся куча не закончится, и массив будет отсортирован.

# **ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ**

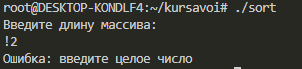
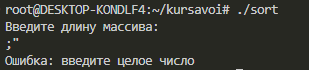
Реакция программы на ввод букв вместо длины массива:

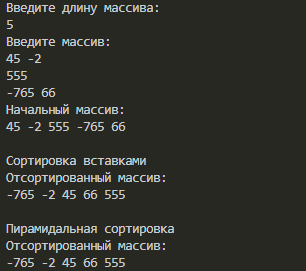
На нецелое число или другое некорректное число:

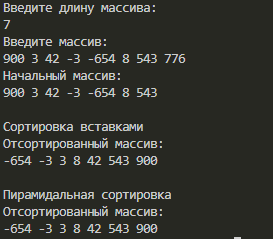
На знаки препинания:

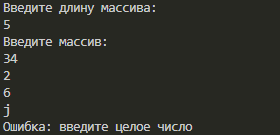
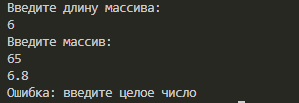
Поведение программы при корректном вводе:



Поведение при вводе большего количества элементов, чем названная ранее длина массива:



Некорректный ввод массива:

# **ПРИЛОЖЕНИЕ**

sort.c

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106 | *#include <stdio.h>*  *#include <stdlib.h>*  **void** heapify**(int\*** arr**,** **int** n**,** **int** i**)**  **{**  **int** largest **=** i**;**  **int** l **=** **2\***i **+** **1;**  **int** r **=** **2\***i **+** **2;**  **if** **(**l **<** n **&&** arr**[**l**]** **>** arr**[**largest**])** **{**  largest **=** l**;**  **}**    **if** **(**r **<** n **&&** arr**[**r**]** **>** arr**[**largest**])** **{**  largest **=** r**;**  **}**  **if** **(**largest **!=** i**)** **{**  **int** temp **=** arr**[**i**];**  arr**[**i**]** **=** arr**[**largest**];**  arr**[**largest**]** **=** temp**;**    heapify**(**arr**,** n**,** largest**);**  **}**  **}**  **void** heap\_sort**(int\*** arr**,** **int** n**)**  **{**  **for** **(int** i **=** n**/2** **-** **1;** i **>=** **0;** i**--)** **{**  heapify**(**arr**,** n**,** i**);**  **}**    **for** **(int** i **=** n **-** **1;** i **>=** **0;** i**--)** **{**  **int** temp **=** arr**[0];**  arr**[0]** **=** arr**[**i**];**  arr**[**i**]** **=** temp**;**    heapify**(**arr**,** i**,** **0);**  **}**  **}**  **void** insertion\_sort**(int\*** arr**,** **int** n**)**  **{**  **int** i**,** key**,** j**;**  **for** **(**i **=** **1;** i **<** n**;** i**++)** **{**  key **=** arr**[**i**];**  j **=** i **-** **1;**  **while** **(**j **>=** **0** **&&** arr**[**j**]** **>** key**)** **{**  arr**[**j **+** **1]** **=** arr**[**j**];**  j **=** j **-** **1;**  **}**  arr**[**j **+** **1]** **=** key**;**  **}**  **}**  **int** main**()**  **{**  **int** n**;**  printf**(**"Введите длину массива:\n"**);**  **if** **(**scanf**(**"%d"**,** **&**n**)** **!=** **1)** **{**  printf**(**"Ошибка: введите целое число\n"**);**  exit**(-1);**  **}**  **int** arr**[**n**];**  printf**(**"Введите массив:\n"**);**  **for** **(int** i **=** **0;** i **<** n**;** i**++){**  **if** **(**scanf**(**"%d"**,** **&**arr**[**i**])** **!=** **1)** **{**  printf**(**"Ошибка: введите целое число\n"**);**  exit**(-1);**  **}**  **}**      printf**(**"Начальный массив: \n"**);**  **for** **(int** i **=** **0;** i **<** n**;** i**++){**  printf**(**"%d "**,** arr**[**i**]);**  **}**  printf**(**"\n\n"**);**  **int** insertion**[**n**];**  **int** heap**[**n**];**  **for** **(int** i **=** **0;** i **<** n**;** i**++)** **{**  insertion**[**i**]** **=** arr**[**i**];**  heap**[**i**]** **=** arr**[**i**];**  **}**    printf**(**"Сортировка вставками\n"**);**  insertion\_sort**(**insertion**,** n**);**  printf**(**"Отсортированный массив: \n"**);**  **for** **(int** i **=** **0;** i **<** n**;** i**++){**  printf**(**"%d "**,** insertion**[**i**]);**  **}**  printf**(**"\n\n"**);**  printf**(**"Пирамидальная сортировка\n"**);**  heap\_sort**(**heap**,** n**);**  printf**(**"Отсортированный массив: \n"**);**  **for** **(int** i **=** **0;** i **<** n**;** i**++){**  printf**(**"%d "**,** heap**[**i**]);**  **}**  printf**(**"\n"**);**  **return** **0;**  **}** |