МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №2
по курсу «ЭВМ и периферийные устройства»

ИЗУЧЕНИЕ ОПТИМИЗИРУЮЩЕГО КОМПИЛЯТОРА

Выполнил: студент 2-го курса гр. 17208
Гафиятуллин А.Р.

Новосибирск, 2018

1. ЦЕЛИ РАБОТЫ:

1. Изучение основных функций оптимизирующего компилятора, и некоторых примеров оптимизирующих преобразований и уровней оптимизации;
2. Получение базовых навыков работы с компилятором GCC;
3. Исследование влияния оптимизационных настроек компилятора GCC на время исполнения программы.

2. ХОД РАБОТЫ:

Для достижения поставленных целей был выбран 7 вариант задания:

Алгоритм сортировки методом пузырька. Дан массив случайных чисел длины N. На первой итерации попарно упорядочиваются все соседние элементы; на второй – все элементы, кроме последнего элемента; на третьей – все элементы, кроме последнего элемента и предпоследнего элемента и т.п.

1. Написана программа на языке C++, которая реализует алгоритм сортировки методом пузырька;

   Исходный код программы:

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <sys/times.h>
#include <unistd.h>
using namespace std;

template<typename T>
void bubble_sort(vector<T> &v)
{
    for(auto out_iter = v.end() - 1; out_iter != v.begin(); out_iter--)
        for(auto in_iter = v.begin(); in_iter != out_iter; in_iter++)
```

```
                if(*in_iter > *(in_iter + 1))
                    swap(*in_iter, *(in_iter + 1));
}

int main()
{
    srand(time(NULL));
    long long int size;
    cin >> size;
    vector<int> v(size);
    for(auto &element : v)
        element = rand();

    struct tms start, finish;
    long clocks_per_sec = sysconf(_SC_CLK_TCK);
    times(&start);
    bubble_sort(v);
    times(&finish);
    double clocks = finish.tms_utime - start.tms_utime;
    cout << endl << "Total process time: " << (double)clocks / clocks_per_sec
        << "s" << endl;
    return 0;
}
```

Команда компиляции (без оптимизаций): g++ -std=c++11 main.cpp -o main

При компилировании с оптимизациями добавляется ключ -O{0, 1, 2, 3, s, fast, g), например g++ -O1 -std=c++11 main.cpp -o main

2. Проверена правильность работы программы на нескольких тестовых наборах входных данных:
   1. Вход: 81 78 76 35 29 81 52 93 22 67

Выход: 22 29 35 52 67 76 78 81 81 93

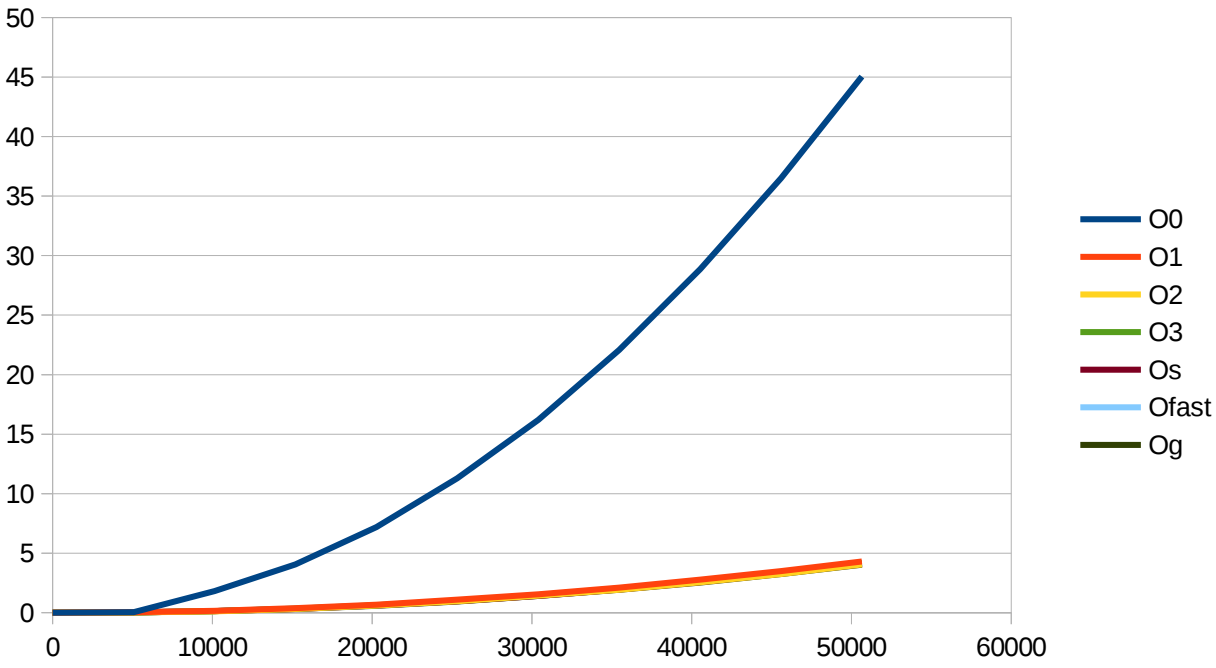
2. Вход: 620 776 746 849 149 75 58 920 349 561 882 302 612 157 773

   Выход: 58 75 149 157 302 349 561 612 620 746 773 776 849 882 920

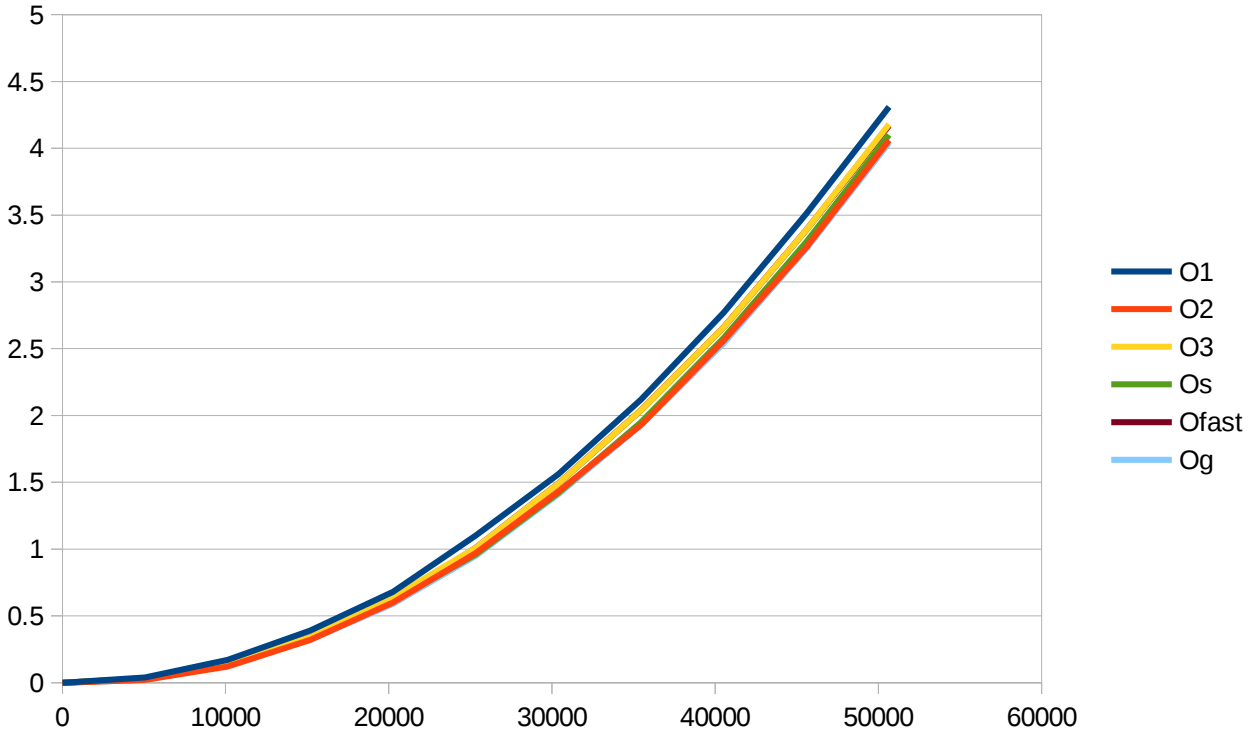
3. Вход: 9343 1879 3750 6152 4770 1050 3199 5646 2638 7844

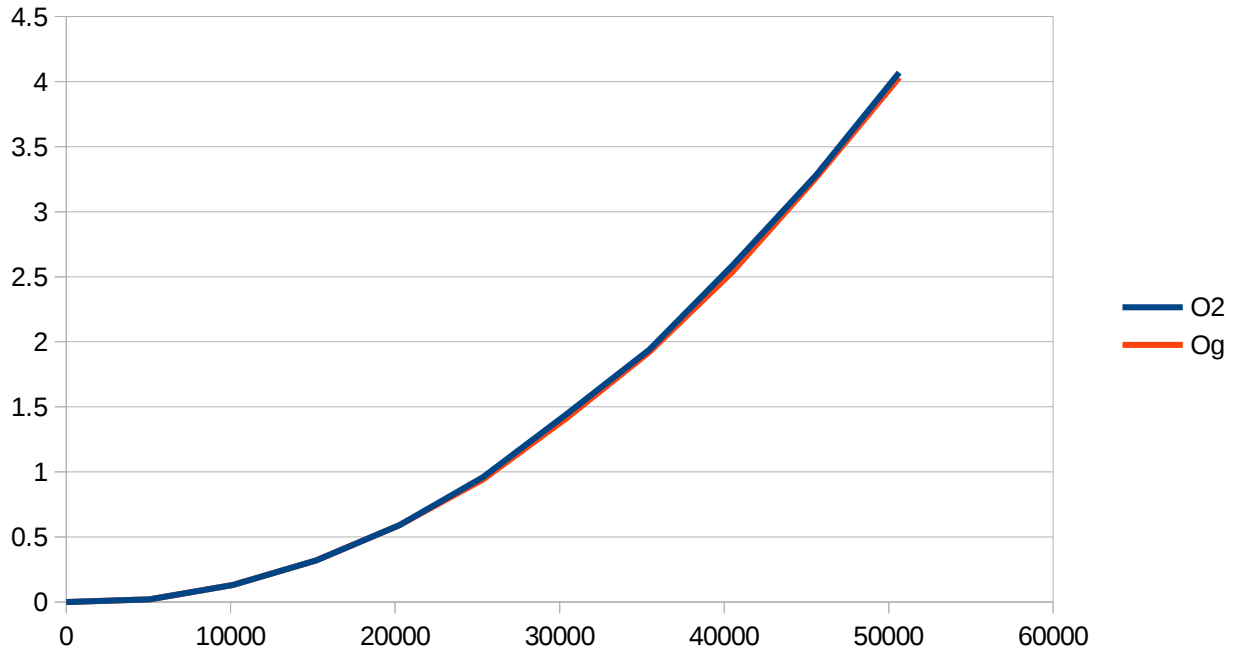
   Выход: 1050 1879 2638 3199 3750 4770 5646 6152 7844 9343

3. На момент тестирования времени работы программы в Linux-машине с Elementary OS(Linux kernel 4.15.0-33-generic, Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, оболочка Pantheon(X Windows System), performance говернер) было запущено около 230-240 процессов, а так же была выполнена команда sync. Для более точного измерения времени работы алгоритма (без части кода, где вводятся данные) в многозадачной ОС, был использован таймер времени работы процесса times(), обрамляющий вызов функции сортировки методом пузырька. Тестовым путем было выясненно, что при текущей конфигурации компьютера и ОС, время работы алгоритма составляет порядка 45 секунд при сортировке около 50650 элементов.
4. Графики зависимости времени выполнения программы с уровнями оптимизации -O0, -O1, -O2, -O3, -Os, -Ofast, -Og от параметра N:

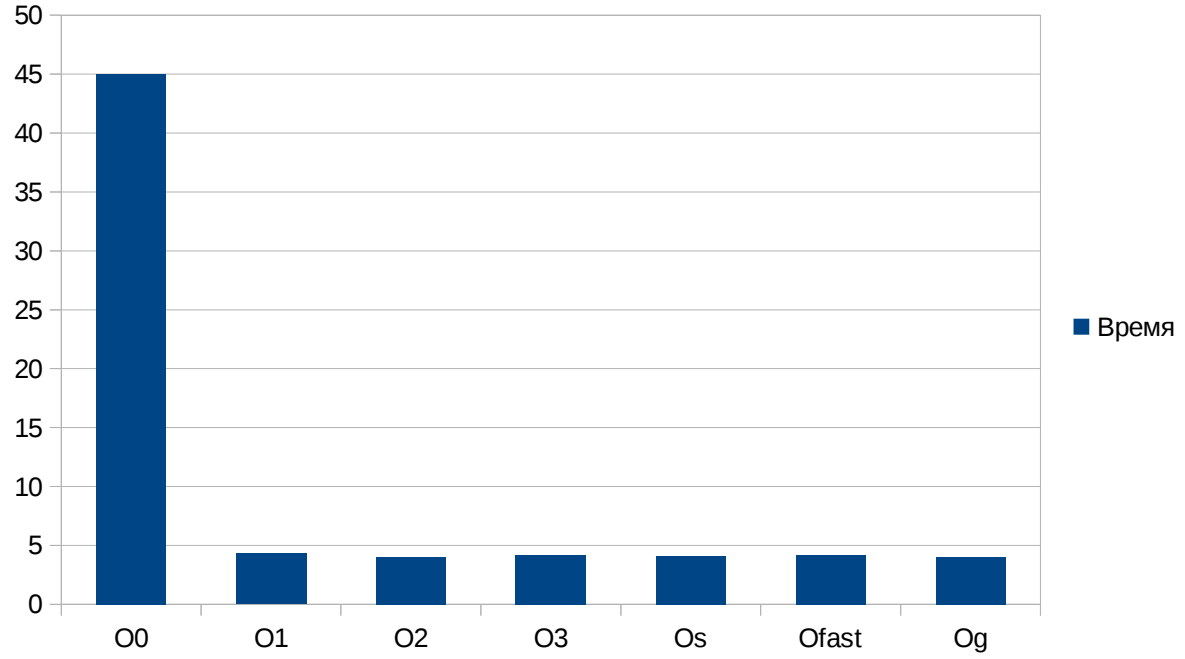


Из графика, приведенного выше, можно заметить, что уже даже первый уровень оптимизации алгоритма дает примерно 10-кратное ускорение выполнения алгоритма, но между самими уровнями оптимизации отличия при таком масштабе не так очевидны, поэтому рассмотрим их подробнее на графике ниже:





Показатели времени при N = 50650:



Проанализировав графики, можно сделать вывод, что лучшие показатели времени достигаются при оптимизациях уровня -O2 и -Og. -Og в этом тесте показал лучшие результаты, хотя эти показатели сильно зависят от самой тестируемой программы и конфигурации ОС и компьютера.

3. ВЫВОДЫ

1. Изучили основные функции оптимизирующего компилятора, и некоторых примеров оптимизирующих преобразований и уровней оптимизации;
2. Получили базовые навыков работы с компилятором GCC;
3. Исследовали влияние оптимизационных настроек компилятора GCC на время исполнения программы;
4. Выяснили, что уровня O2 достаточно для получения очень высоких результатов оптимизации, причем данная оптимизация не увеличивает размер кода.