МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра вычислительной техники

Отчет по лабораторной работе №8
по дисциплине: «Программирование»

Факультет: АВТФ

Группа: АВТ-143

Студент: Васютин А. М.

Преподаватель: Новицкая Ю. В.

Вариант: 4

Новосибирск, 2022 г.

ОГЛАВЛЕНИЕ

1. ЛАБОРАТОРНАЯ РАБОТА №8. БИБЛИОТЕКА ШАБЛОННЫХ КЛАССОВ STL (STANDART TEMPLATE LIBRARY)

1.1. Цель работы

Ознакомиться с шаблонными классами библиотеки STL. Изучить применение этих классов и их методов на практике.

1.2. Задание на лабораторную работу

Для встроенного типа (например, int или char) провести временной анализ заданных шаблонных классов на основных операциях: добавление, удаление, поиск, сортировка. Использовать итераторы для работы с контейнерами. Для получения времени выполнения операции засекать системное время начала и окончания операции и автоматически генерировать большое количество данных.

Структуры данных – Стек; множество с дубликатами.

1.3. Решение

В стандартной библиотеке шаблонных классов стек и множество с дубликатами представленны классами queue и multiset и подключаются заголовочными файлами queue и set соответственно. Для замера времени выполнения был реализован макрос, генерирующий функцию, замеряющую время выполнения кода и из-за того, что компилятор, согласно стандарту, не может оптимизировать порядок side-effect’ов, то эти ассемблерные вставки не позволяют ему оптимизировать выполнение кода.

```
% ~/c/nstu-labs  main *  term_3/lab_8/build  ./lab 1000000 10
Test: stack_push
repeat: 10
size: 1000000
Time(average ∓ σ): 27.6114 ∓ 2.60612 ms
Range(min … max):  26.2572 … 35.3759 ms
Test: stack_pop
repeat: 10
size: 1000000
Time(average ∓ σ): 7.01033 ∓ 0.0647625 ms
Range(min … max):  6.92195 … 7.11174 ms
Test: multiset_insert
repeat: 10
size: 1000000
Time(average ∓ σ): 486.55 ∓ 6.53687 ms
Range(min … max):  478.539 … 501.754 ms
Test: multiset_erase
repeat: 10
size: 1000000
Time(average ∓ σ): 103.197 ∓ 3.56755 ms
Range(min … max):  96.9154 … 109.112 ms
Test: multiset_find
repeat: 10
size: 1000000
Time(average ∓ σ): 1147.56 ∓ 53.5203 ms
Range(min … max):  1049.05 … 1239.25 ms
% ~/c/nstu-labs  main *  term_3/lab_8/build
```

Рисунок 1 – Вывод работы программы.

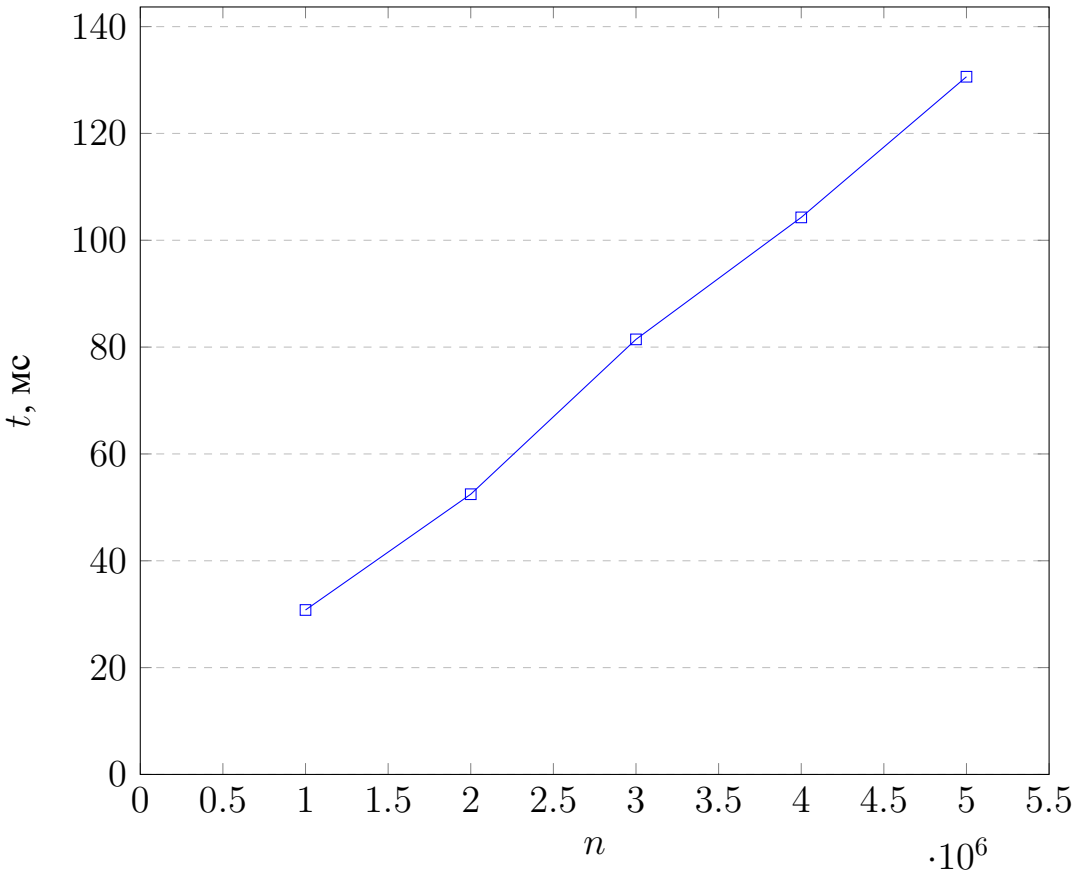

Рисунок 2 – *stack_push*

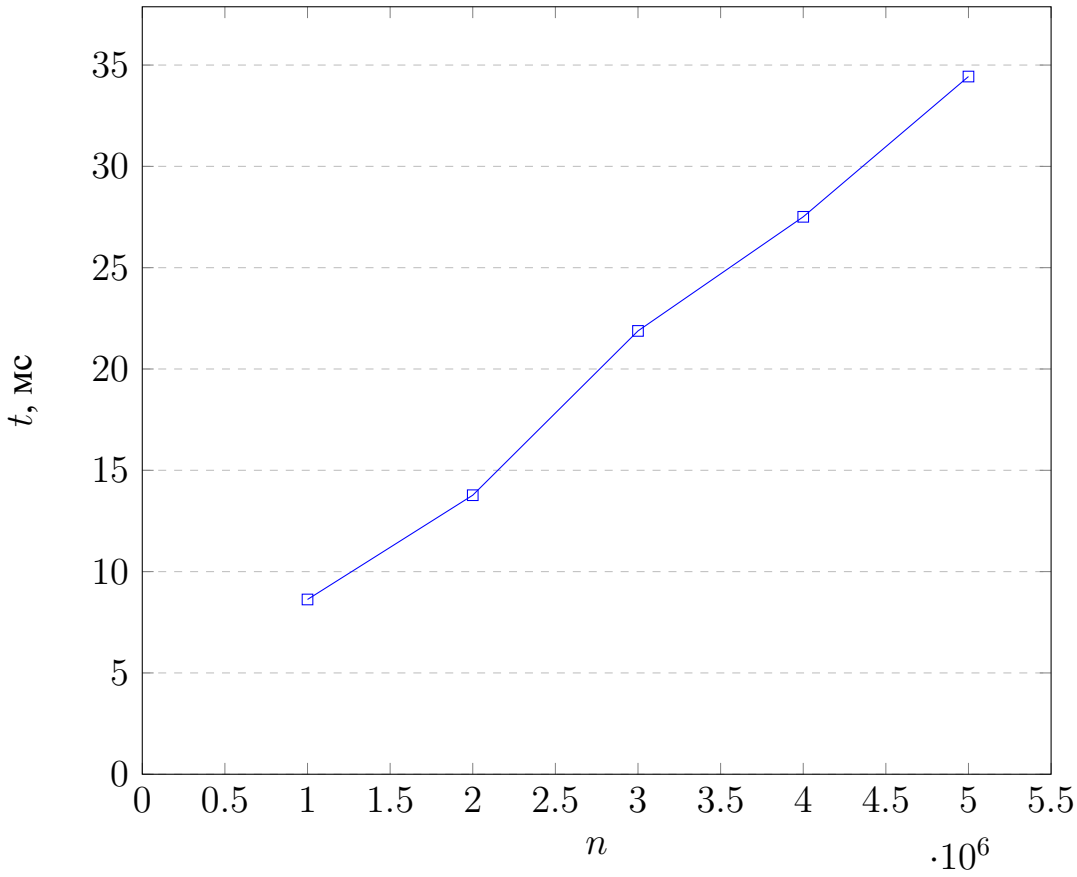

Рисунок 3 – *stack_pop*

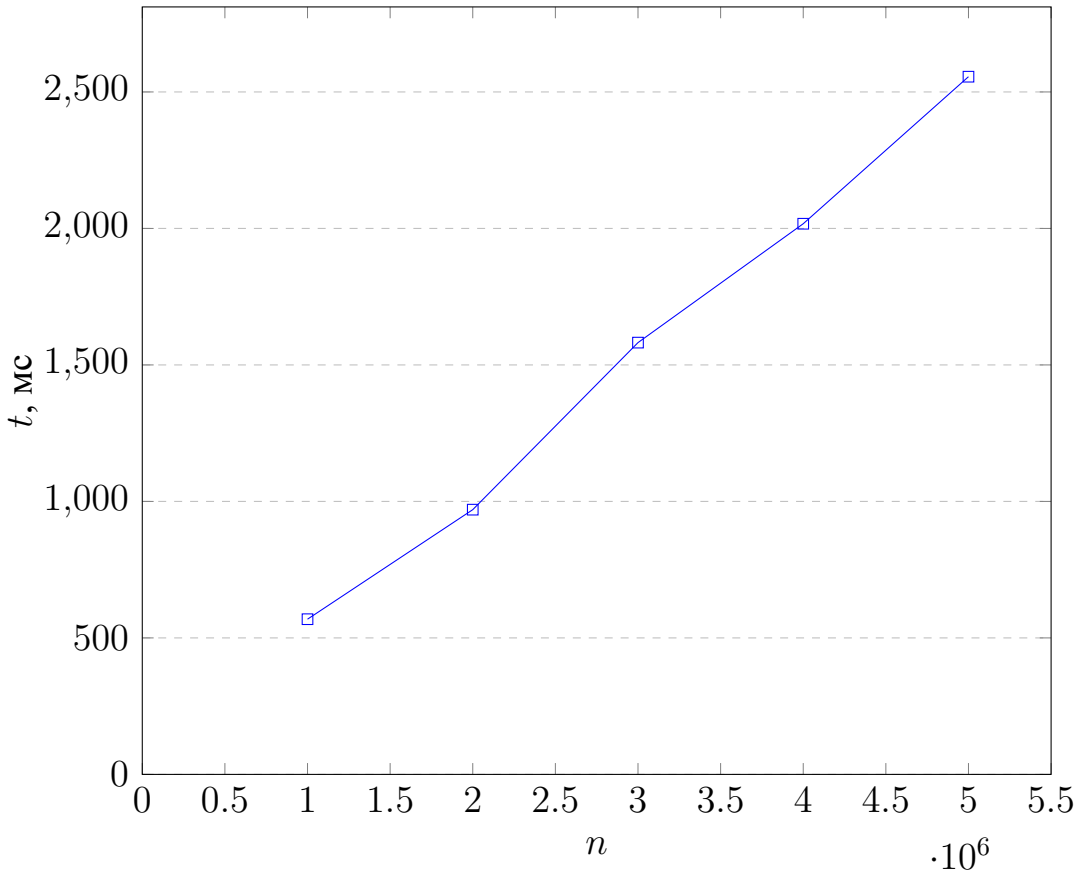

Рисунок 4 – *multiset_insert*

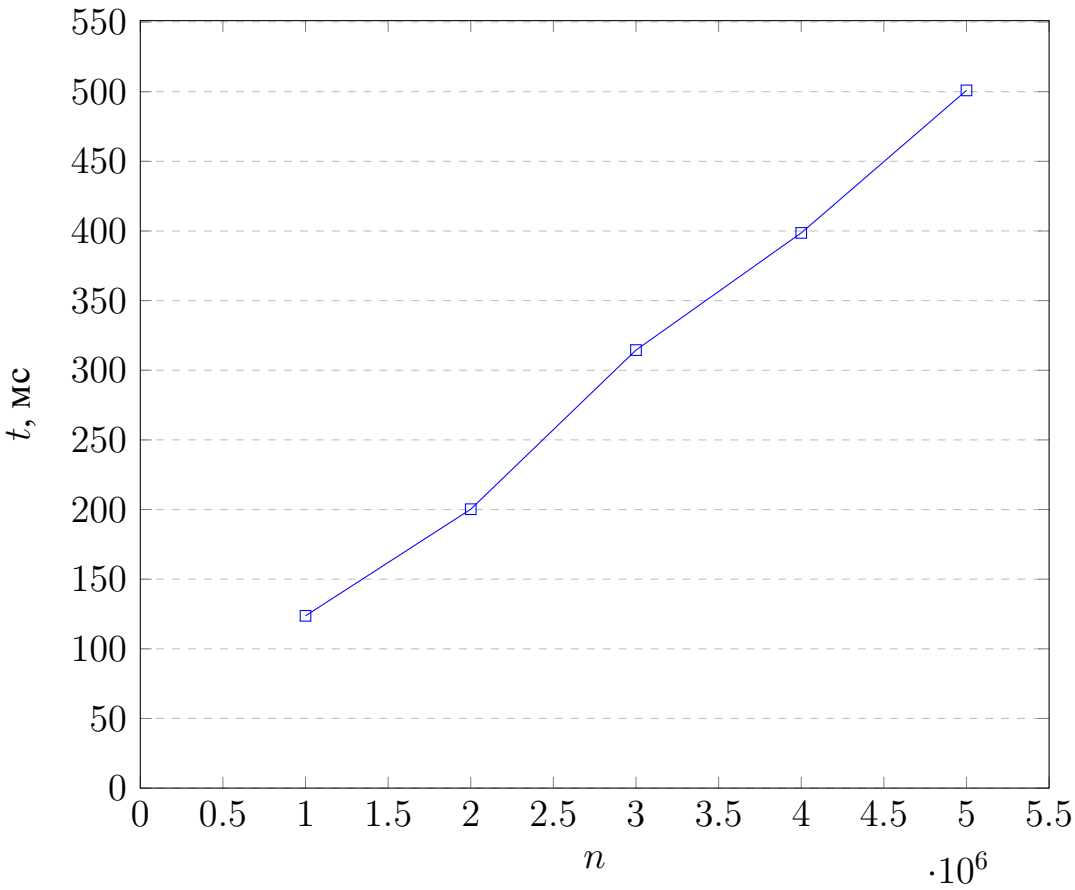

Рисунок 5 – *multiset_erase*

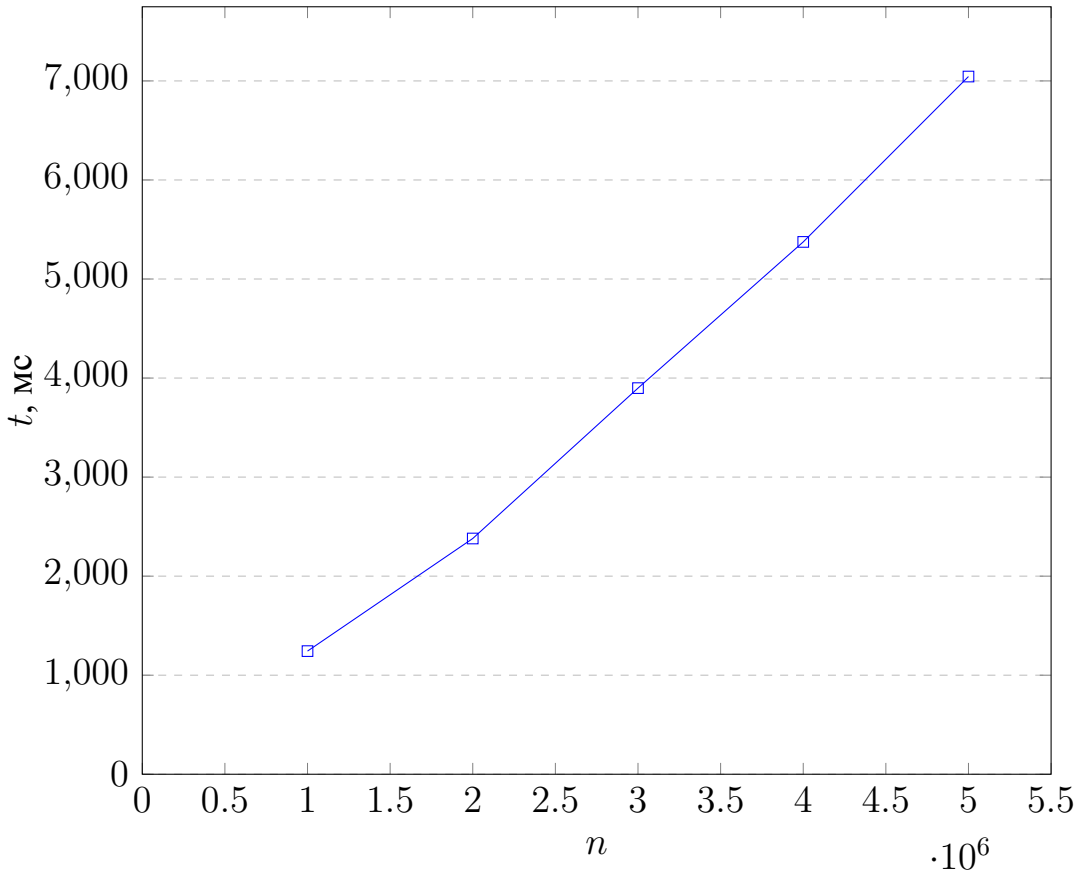

Рисунок 6 – *multiset_find*

1.4. Исходный код

Листинг 1 – main.cpp

```
#include <algorithm>
#include <chrono>
#include <cmath>
#include <cstdint>
#include <iostream>
#include <iterator>
#include <numeric>
#include <set>
#include <stack>
#include <string>
#include <vector>

#define MEM_DEFAULT 1
#define REPS_DEFAULT 1

#define BENCH(NAME, TYPE, ARG, TO_TEST)                                 \
  void bench_##NAME(TYPE ARG, int reps, int N) {                        \
    std::cout << "Test: " << #NAME << std::endl                         \
              << "repeat: " << reps << std::endl                        \
              << "size: " << N << std::endl;                            \
    std::vector<double> results(reps);                                  \
    TYPE ARG##_copy = ARG;                                              \
```

```
    for (int i = 0; i < reps; ++i) {                                          \
      ARG = ARG##_copy;                                                       \
      auto start = std::chrono::high_resolution_clock::now();                 \
      asm volatile("" ::"g"(start));                                          \
      for (int j = 0; j < N; ++j) {                                           \
        TO_TEST;                                                              \
      }                                                                       \
      auto end = std::chrono::high_resolution_clock::now();                   \
      asm volatile("" ::"g"(end));                                            \
      results[i] =                                                            \
          std::chrono::duration<double>(end - start).count() * 1000.0;        \
    }                                                                         \
    const auto average =                                                      \
        std::accumulate(results.begin(), results.end(), 0.0) / results.size(); \
    const auto sigma =                                                        \
        std::sqrt(std::accumulate(results.begin(), results.end(), 0.0,        \
                                  [&](const auto &i, const auto &j) {         \
                                    return i + std::pow(j - average, 2);      \
                                  }) /                                        \
                  results.size());                                            \
    std::cout << "Time(average \u2213 \u03c3): " << average << " \u2213 "     \
              << sigma << " ms" << std::endl                                  \
              << "Range(min \u2026 max):  "                                   \
              << *std::min_element(results.begin(), results.end())            \
              << " \u2026 "                                                   \
              << *std::max_element(results.begin(), results.end()) << " ms"   \
              << std::endl;                                                   \
  }

int maybe_readopt(int argc, char *argv[], int n, int def) {
  if (argc > n) {
    char *flag = nullptr;
    return std::strtoll(argv[n], &flag, 0);
  }
  return def;
}

BENCH(stack_push, std::stack<int>, stack, stack.push(1))
BENCH(stack_pop, std::stack<int>, stack, stack.pop())
BENCH(multiset_insert, std::multiset<int>, multiset, multiset.insert(1))
BENCH(multiset_erase, std::multiset<int>, multiset,
      multiset.erase(std::prev(multiset.end())))
BENCH(multiset_find, std::multiset<int>, multiset, multiset.find(std::rand()))

int main(int argc, char *argv[]) {
  std::srand(std::time(nullptr));

  int size = maybe_readopt(argc, argv, 1, MEM_DEFAULT);
  int reps = maybe_readopt(argc, argv, 2, REPS_DEFAULT);

  /* Для встроенного типа (например, int или char) провести временной анализ
   * заданных шаблонных классов на основных операциях: добавление, удаление,
   * поиск, сортировка. Использовать итераторы для работы с контейнерами.
   * Для получения времени выполнения операции засекать системное время
```

```
   * начала и окончания операции и автоматически генерировать большое
   * количество данных
   */
  {
    std::stack<int> stack;
    bench_stack_push(stack, reps, size);
  }
  {
    std::stack<int> stack;
    for (int i = 0; i < size; ++i)
      stack.push(std::rand());
    bench_stack_pop(stack, reps, size);
  }
  {
    std::multiset<int> multiset;
    bench_multiset_insert(multiset, reps, size);
  }
  {
    std::multiset<int> multiset;
    for (int i = 0; i < size; ++i)
      multiset.insert(std::rand());
    bench_multiset_erase(multiset, reps, size);
  }
  {
    std::multiset<int> multiset;
    for (int i = 0; i < size; ++i)
      multiset.insert(std::rand());
    bench_multiset_find(multiset, reps, size);
  }
  // {
  //   std::multiset<int> multiset;
  //   for (int i = 0; i < 10; ++i)
  //     multiset.insert(std::rand());
  //   for (const auto &i : multiset)
  //     std::cout << i << std::endl;
  // }
}
```

1.5. Выводы

В данной лабораторной работе была изучена стандартная библиотека шаблонов и шаблонные контейнеры, содержащиеся в ней (на примере очереди и множества с повторениями). По результатам проведённого анализа, можно сделать выводы, что некоторые операции лишены смысла для конкретного контейнера. Так, нет необходимости в сортировке множества с повторениями, так как оно сортируется автоматически, что немного замедляет все остальные операции. Следовательно, выбирать контейнеры из STL (или создавать свои) нужно исходя из поставленных задач.