Statistically Random Subsets

Jarred Parr

August 2018

1 Introduction

This project laid out a fictional scenario of the acquisition of medical records in a data set that may or may not be randomized. The goal was to get k randomly selected records from a data set of size n and then return the sorted list. For this project the approach I took was the most efficient given the artificial constraints. For instance, instead of modifying the provided list, n, I instead had to copy the records over to a new vector since doing any of the operations in place would have lead to lost records. To avoid this, I simply created a new list to copy the elements over as to not lose any from the original list. As a result, the implementation lacks some minor core efficiency gains but it keeps with the project description.

2 What I Learned

Here I didn't learn a ton of new things about C++ itself since conceptually things were quite straightforward, however, I did learn a great deal about sorting algorithms and some high performance tuning in C++. On top of that, I implemented a quicksort algorithm from scratch for fun and to learn the inner workings of it. One could argue the built in sorting systems are better and I would not dispute that, but it was fun to learn how it worked on a deeper level.

3 Code Performance

My code exhibited the following run time performance:

- With the built in $std :: sort$ function: $85ms$
- With my custom quick sort implementation: $115ms$

4 Code

My full source code is as follows:

```
#include "statistically_random_subsets.h"
#include <algorithm>
#include <chrono>
#include <cstdlib>
#include <iostream>
#include <iterator>
#include <random>

namespace stats {
int StatisticallyRandomSubsets::partition(std::vector<int> & arr,
int low, int high) {
  int pivot = arr[high];

  // Our artifical "wall"
  int i = low - 1;

  for (int j = low; j <= high - 1; ++j) {
    if (arr[j] <= pivot) {
      i++;
      std::iter_swap(arr.begin() + i, arr.begin() + j);
    }
  }

  std::iter_swap(arr.begin() + (i + 1), arr.begin() + high);

  return i + 1;
}

std::vector<int> StatisticallyRandomSubsets::sort(
std::vector<int> & unsorted_vector, int low, int high) {
  if (low < high) {
    int p = partition(unsorted_vector, low, high);

    sort(unsorted_vector, low, p - 1);
    sort(unsorted_vector, p + 1, high);
  }
```

```
  return unsorted_vector;
}

std::vector<int> StatisticallyRandomSubsets::generate(int k,
 const std::vector<int> & n) {
  std::vector<int> random_list(n);
  std::random_device rd;

  // Mersaine Twister Pseudo random number genrator
  // This is an optimized random number generator in the stl
  std::mt19937 g(rd());

  std::shuffle(random_list.begin(), random_list.end(), g);
  random_list.resize(k);

  return random_list;
}
} // namespace stats

int main() {
  stats::StatisticallyRandomSubsets srs;

  std::vector<int> n;
  n.reserve(500);
  int k = 50;
  for (int i = 0; i < 500; ++i) {
    n.push_back(i);
  }

  std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
  std::vector<int> output = srs.generate(k, n);
  // Custom sort takes ~30ms longer
  /* output = srs.sort(output, 0, output.size() - 1); */

  // Faster sort option
  std::sort(output.begin(), output.end());
  std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();

  std::copy(output.begin(), output.end(),
```

```
   std::ostream_iterator<int>(std::cout, " "));
  std::cout << "\n\nRunning Times" << std::endl;
  std::cout << "---------------------------------------" << std::endl;
  std::cout << "Time difference = " <<
   std::chrono::duration_cast<std::chrono::microseconds>(end - begin).count()
   << "ms" << std::endl;
  std::cout << "Time difference = " <<
   std::chrono::duration_cast<std::chrono::nanoseconds> (end - begin).count()
   << "ns" << std::endl;

  return 0;
}
```

5 Conclusion

This project showed me a great deal about the basics of run time performance and how to do performance tuning to squeeze every ounce of speed out of a particular set of functions I have written. I really enjoyed learning what I did on this lighter project and am eager to tackle new ones.