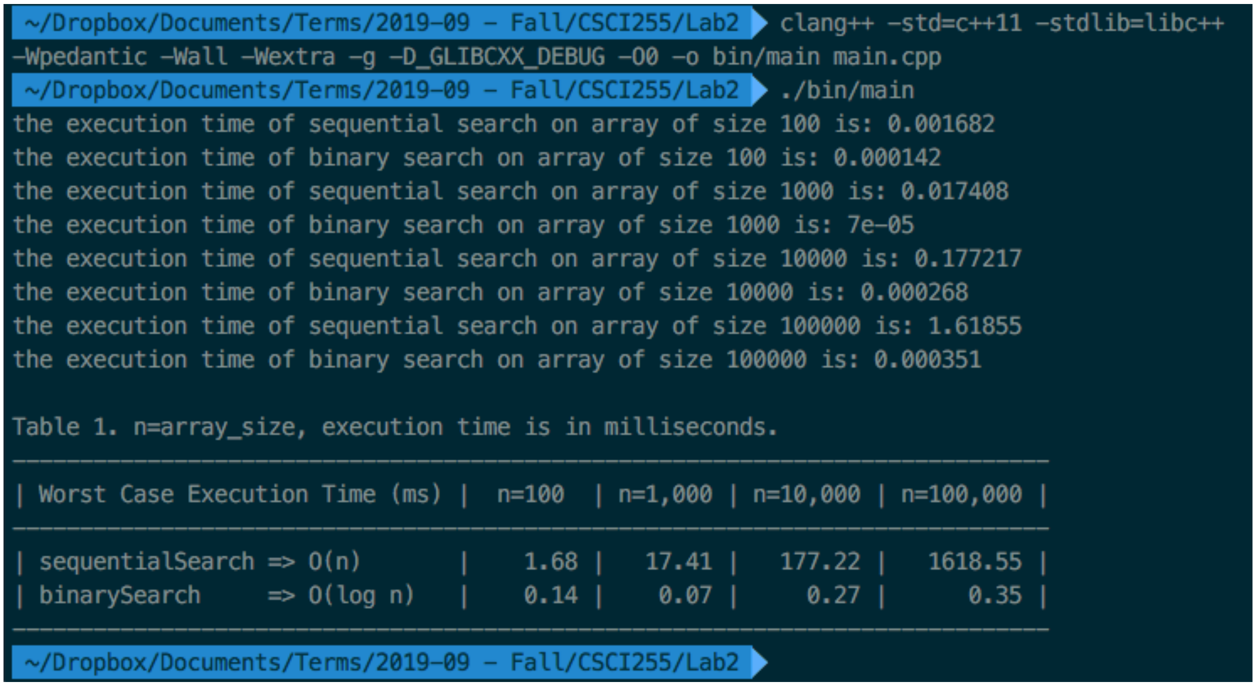
CSCI 255 — Lab 2: Time Performance of Search Algorithms

CONSOLE OUTPUT & TABLE

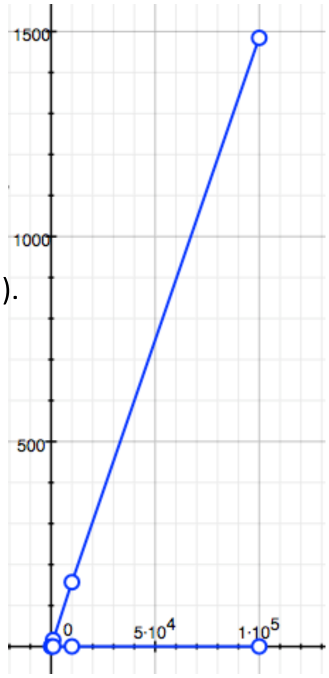


```
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255/Lab2 clang++ -std=c++11 -stdlib=libc++
-Wpedantic -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0 -o bin/main main.cpp
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255/Lab2 ./bin/main
the execution time of sequential search on array of size 100 is: 0.001682
the execution time of binary search on array of size 100 is: 0.000142
the execution time of sequential search on array of size 1000 is: 0.017408
the execution time of binary search on array of size 1000 is: 7e-05
the execution time of sequential search on array of size 10000 is: 0.177217
the execution time of binary search on array of size 10000 is: 0.000268
the execution time of sequential search on array of size 100000 is: 1.61855
the execution time of binary search on array of size 100000 is: 0.000351

Table 1. n=array_size, execution time is in milliseconds.
-------------------------------------------------------------------------------
| Worst Case Execution Time (ms) |  n=100   | n=1,000 | n=10,000 | n=100,000 |
-------------------------------------------------------------------------------
| sequentialSearch => O(n)       |    1.68 |   17.41 |   177.22 |   1618.55 |
| binarySearch     => O(log n)   |    0.14 |    0.07 |     0.27 |      0.35 |
-------------------------------------------------------------------------------
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255/Lab2
```

Q. *How well do the measured execution times of the C++ functions agree with the Big-Oh of the sequential and binary search algorithms?*

Calculations from a different instance but scaling still holds. Also very obvious if you look at a graph of those numbers.



SelectionSearch execution time scaled linearly with n as expected with O(n).

For example, going from n=1,000 to n=100,000:

$$0.021281 * 100 = 2.121$$
$$\cong 2.39$$

BinarySearch execution time scales logarithmic with n as expected with O(log n).

For example, lets break down n=1,000:

$$c * \log(1000) = 0.000282$$
$$c * \log(10^3) = 0.000282$$
$$3c * log(10) = 0.000282$$

Now to predict Log scaling for n = 100,000 multiply both sides by 5/3

$$\left(\frac{5}{3}\right) 3\mathrm{c} * \log(10) = \left(\frac{5}{3}\right) * 0.000282$$
$$5\mathrm{c} * \log(10) = 0.00047$$
$$\mathrm{c} * \log(100{,}000) = 0.00047$$
$$\cong 0.000486$$

CODE

```
/* main.cpp
 * --------
 * Authors: Darwin Jacob Groskleg, Man Lin
 * Date:    Tuesday, September 17th, 2019
 * CSCI 255
 * Lab 2 — Time Performance of Search Algorithms
 */
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <thread>
#include <chrono>
#include <tuple>
#include <VariadicTable.h> // installed library

using namespace std;

int sequentialSearch(int array[], int, int);
int binarySearch(int array[], int, int);

int main() {
  double start;
  double finish;
  double elapsed;
  int where;
  // Generate sorted arrays of different sizes
  int array1[100];
  for(int i=0; i<100; i++)
    array1[i] = i;
  int array2[1000];
  for(int i=0; i<1000; i++)
    array2[i] = i;
  int array3[10000];
  for(int i=0; i<10000; i++)
    array3[i] = i;
  int array4[100000];
  for(int i=0; i<100000; i++)
    array4[i] = i;

  VariadicTable<string, double, double, double, double> vt(
    {"Worst Case Execution Time (ms)",
     "n=100", "n=1,000", "n=10,000", "n=100,000"
    }, 7);
  tuple<string, double, double, double, double>
    seq_row("sequentialSearch => O(n)"),
    bin_row("binarySearch     => O(log n)");
  vt.setColumnFormat({VariadicTableColumnFormat::AUTO,
                     VariadicTableColumnFormat::FIXED,
                     VariadicTableColumnFormat::FIXED,
                     VariadicTableColumnFormat::FIXED,
                     VariadicTableColumnFormat::FIXED});
  vt.setColumnPrecision({1, 2, 2, 2, 2});


  // n=100
  start = double(clock()) / CLOCKS_PER_SEC; // start time
  where = sequentialSearch(array1, 100, 99);
  finish = double(clock()) / CLOCKS_PER_SEC; // end time
  elapsed = finish - start;
  cout << "the execution time of sequential search on array of size 100 is: "
      << elapsed << endl;
  get<1>(seq_row) = elapsed*1000;

  start = double(clock()) / CLOCKS_PER_SEC; // start time
  where = binarySearch(array1, 100, 99);
  finish = double(clock()) / CLOCKS_PER_SEC; // end time
  elapsed = finish - start;
  cout << "the execution time of binary search on array of size 100 is: "
      << elapsed << endl;
  get<1>(bin_row) = elapsed*1000;


  // n=1,000
  start = double(clock()) / CLOCKS_PER_SEC; // start time
  where = sequentialSearch(array2, 1000, 999);
  finish = double(clock()) / CLOCKS_PER_SEC; // end time
  elapsed = finish - start;
  cout << "the execution time of sequential search on array of size 1000 is: "
      << elapsed << endl;
  get<2>(seq_row) = elapsed*1000;

  start = double(clock()) / CLOCKS_PER_SEC; // start time
  where = binarySearch(array2, 1000, 999);
  finish = double(clock()) / CLOCKS_PER_SEC; // end time
  elapsed = finish - start;
  cout << "the execution time of binary search on array of size 1000 is: "
```

```
      << elapsed << endl;
  get<2>(bin_row) = elapsed*1000;

  // n=10,000
  start = double(clock()) / CLOCKS_PER_SEC; // start time
  where = sequentialSearch(array3, 10000, 9999);
  finish = double(clock()) / CLOCKS_PER_SEC; // end time
  elapsed = finish - start;
  cout << "the execution time of sequential search on array of size 10000 is: "
      << elapsed << endl;
  get<3>(seq_row) = elapsed*1000;

  start = double(clock()) / CLOCKS_PER_SEC; // start time
  where = binarySearch(array3, 10000, 9999);
  finish = double(clock()) / CLOCKS_PER_SEC; // end time
  elapsed = finish - start;
  cout << "the execution time of binary search on array of size 10000 is: "
      << elapsed << endl;
  get<3>(bin_row) = elapsed*1000;

  // n=100,000
  start = double(clock()) / CLOCKS_PER_SEC; // start time
  where = sequentialSearch(array4, 100000, 99999);
  finish = double(clock()) / CLOCKS_PER_SEC; // end time
  elapsed = finish - start;
  cout << "the execution time of sequential search on array of size 100000 is: "
      << elapsed << endl;
  get<4>(seq_row) = elapsed*1000;

  start = double(clock()) / CLOCKS_PER_SEC; // start time
  where = binarySearch(array4, 100000, 99999);
  finish = double(clock()) / CLOCKS_PER_SEC; // end time
  elapsed = finish - start;
  cout << "the execution time of binary search on array of size 100000 is: "
      << elapsed << endl;
  get<4>(bin_row) = elapsed*1000;

  vt.addRow(seq_row);
  vt.addRow(bin_row);
  cout << "\nTable 1. n=array_size, execution time is in milliseconds.\n";
  vt.print(cout);

  return 0;
}
```

```
// An implementation of the sequential search algorithm
int sequentialSearch(int array[], int size, int searchKey)
{
   for(int i=0; i<size;i++)
   {
      this_thread::sleep_for(chrono::milliseconds(1));
      if(searchKey==array[i])
      {
        return i; // found
      }
   }
    return -1; // not found
}

// An implementation of the binary search algorithm
int binarySearch(int array[], int size, int searchKey)
{
   int left = 0;
   int right = size-1;
   int mid;
   while (left <= right)
   {
      this_thread::sleep_for(chrono::milliseconds(1));
      mid = (int) ((left + right) / 2);
      if (searchKey == array[mid])
      {
         return mid; // found
      }
      else if (searchKey > array[mid])
      {
         left = mid + 1;
      } else {
         right = mid - 1;
      }
}
   return -1; // not found
}
```