lab2

Bhupinder Singh

February 22, 2019

Contents

1 partition.cpp

```
Purpose - The purpose of this lab is to get the studenta familiar
with some more advanced concepts regarding data structures. Lab 1
introduced them to the tools that they wer going to be using throughout
this course, this lab however, dives into the topic of data structures
and algorithms by introducing concepts revolving around how much time
an algorithm takes compareed to another.

The students were provided the logic by the professor. Students and the
professor followed the directions giiven by the author regarding the
partition and select function.

After making both the functions, the student is expected to compare
the algorithm by a built in sorting function in the algorithm library.

The comparision should demonstrate which algorithm sorted the array faster.
```

Lab2 Partition

Partition Function
This is the begining of partition function. The logic followed by the
function is the one provided to us by the instructor.

The function takes in four parameters.
@param --> array on integers
@param --> lower value [left most value in the begining.]
@param --> upper value [right most value in the begining.]
@param --> pivotidx [the position of our pivot.]

```
#include <iostream>
#include "catch.hpp"
#include <algorithm>


int partition(int a[], int lower, int upper, int pivot_idx)
{
        int pivot = a[pivot_idx]; int left = lower; int right = upper-2;
        std::swap(a[pivot_idx], a[upper-1]);

        while(left <= right)
        {
                if(a[left] <= pivot) left++;
                else {std::swap(a[left], a[right]); right--;}
```

```
        }
        std::swap(a[left], a[upper-1]);
        return left;
}
```

2 select.cpp

Lab2 Select Function

The median m of a sequence of n elements is the element that would fall in the middle if the sequence was sorted. That is, $e \leq m$ for half the elements, and $m \leq e$ for the others. Clearly, one can obtain the median by sorting the sequence, but one can do quite a bit better with the following algorithm that finds the kth element of a sequence between a (inclusive) and b (exclusive). (For the median, use $k = n/2$, $a = 0$, and $b = n$.)

```
select(k, a, b)
Pick a pivot p in the subsequence between a and b.
Partition the subsequence elements into three subsequences: the elements <p, =p, >p
Let n1, n2, n3 be the sizes of each of these subsequences.
if k < n1
    return select(k, 0, n1).
else if (k > n1 + n2)
    return select(k, n1 + n2, n).
else
    return p.
```

Implement this algorithm and measure how much faster it is for computing the median of a random large sequence, when compared to sorting the sequence and taking the middle element.

The select function was also developed in class amongst all other students and the professor. This function calls the previously mentioned partition function and uses it to provide the desired results.

```
#include <chrono>
#include <ctime>
#include <iostream>  required libraries
#include <algorithm>
#ifdef UNIT_TEST
#include "catch.hpp"
#endif
using namespace std::chrono;
```

```
Function declaration for partition.
int partition(int a[], int lower, int upper, int pivot_idx);



int select(int arr[], int k, int a, int b)
{
        int p = partition(arr,a,b,k);
        int n1 = p - a;
        int n2 = 1;
        int n3 = b - (n1 + n2);
        CHECK(n1+n2+n3 == b);
        if(k < n1) select(arr, k, 0, n1);  recursive step
        else if(k > n1 + n2) select(arr, k, n1+n2, b);
        else return arr[p]; //base case
}
#ifdef UNIT_TEST

TEST_CASE("Sort")
{
        const int n = 10;
        int a[n] = {2, 87, 21, 3, 12, 78, 97, 16, 89, 21};
         2, 12, 3, 16, 78, 97, 21, 89, 87, 21 == 21
        std::sort(a,a+n); //range [0,n)
        CHECK(a[n/2] == 21);
}


TEST_CASE("hrs")
{
```

```
const int size = 10;
int pivot_index = size / 2;
 int array[size], brray[size];
int c[size] = {2, 87, 21, 3, 12, 78, 97, 16, 89, 21};
int d[size] = {2, 12, 3, 16, 78, 97, 21, 89, 87, 21};
 median of 2, 3, 12, 16, 21, 78, 89, 97 == 21
int N = sizeof(c)/sizeof(int);
int lower = 0;
int upper = N;

high_resolution_clock::time_point t1 = high_resolution_clock::now();

REQUIRE(select(c, N/2, 0, N) == 21); return the median
high_resolution_clock::time_point t2 = high_resolution_clock::now();


duration<double> selectTimeSpan = duration_cast<duration<double>>(t2
     - t1);

std::cout << "select() took " << selectTimeSpan.count() << " seconds
    .";
std::cout << std::endl;


 use sort()
high_resolution_clock::time_point t3 = high_resolution_clock::now();

std::sort(d,d+N);  range [0,n]
REQUIRE(d[N/2] == 21);
high_resolution_clock::time_point t4 = high_resolution_clock::now();

duration<double> sortTimeSpan = duration_cast<duration<double>>(t4 -
```

```
            t3);

        std::cout << "sort() took " << sortTimeSpan.count() << " seconds.";
        std::cout << std::endl;

        std::cout << "selectTimeSpan - sortTimeSpan = "
                  << (selectTimeSpan.count() - sortTimeSpan.count())
                      << " seconds.";
        std::cout << std::endl;

}
```

This the Catch2 for our comparison. The catch test casesmake sure that the values resulting are right before they get passed ontothe next lines.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
select is a Catch v2.0.1 host application.
Run with -? for options

-------------------------------------------------------------------------------
hrs
-------------------------------------------------------------------------------
select.cpp:49
...............................................................................

select.cpp:32:
PASSED:
  CHECK( n1+n2+n3 == b )
with expansion:
  10 == 10

select.cpp:32:
PASSED:
  CHECK( n1+n2+n3 == b )
with expansion:
  6 == 6

select.cpp:63:
PASSED:
  REQUIRE( select(array, N/2, 0, N) == 21 )
with expansion:
  21 == 21

select() took 0.000159971 seconds.
```

```
select.cpp:78:
PASSED:
  REQUIRE( brray[N/2] == 21 )
with expansion:
  21 == 21

sort() took 2.4478e-05 seconds.
selectTimeSpan - sortTimeSpan = 0.000135493 seconds.
0.000 s: hrs
===============================================================================
All tests passed (4 assertions in 1 test case)
```

```
#endif
```

This is the main and last part of our program. Our main
goal in this program was to test the speed of two algorithms. The following
picture shows how the built-in sort() is faster than the algorithm our
class came up with. Hence, the lab is complete.

```
sort() took 0.000273575 seconds.
selectTimeSpan - sortTimeSpan = 0.0010991 seconds.
```