Lab3

Sheharyar Alam Khan

March 9, 2019

Contents

1 partition.cpp

Lab3 partition

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

    In this function we want to partition the array in a way so that all the numbers less
    than the pivot are on the left and all the values greater than the pivot are on the
    right side of the pivot.  If the value on the left is greater than the pivot then its
    swapped with the value on the right.
    @param arr[] :  this parameter takes in the array whose median is to be found and
    returned.
    @param pivotIndex :  this parameter takes in the pivot index.
    @param lower :  this parameter takes in the starting index from which the median is
    supposed to be found.
    @param upper :  this parameter takes in the last index upto which the median is
    supposed to be found.
int partition(int a[], int lower, int upper, int pivotIndex)
{
        int pivot = a[pivotIndex]; int left = lower; int right = upper-1;
        std::swap(a[pivotIndex], a[upper]);
        while(left <= right)
        {
                if(a[left] <= pivot)
                        left++;
                else {
                        std::swap(a[left],a[right]);
                        right--;
```

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


TEST_CASE("Test Partition")
{
        int a[] = {2,87,3,12,78,97,16,89,21};
        int b[] = {2,16,3,12,21,78,89,87,97};
        int n = (sizeof(a) / sizeof(a[0]));
        partition(a, 0, n - 1, n/2);
        for (int i=0; i < n; i++)
        {
                CHECK(a[i] == b[i]);
        }
}
```

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

-------------------------------------------------------------------------------
Test Partition
-------------------------------------------------------------------------------
partition.cpp:30
...............................................................................

partition.cpp:38:
PASSED:
```

```
  CHECK( a[i] == b[i] )
with expansion:
  2 == 2

partition.cpp:38:
PASSED:
  CHECK( a[i] == b[i] )
with expansion:
  16 == 16

partition.cpp:38:
PASSED:
  CHECK( a[i] == b[i] )
with expansion:
  3 == 3

partition.cpp:38:
PASSED:
  CHECK( a[i] == b[i] )
with expansion:
  12 == 12

partition.cpp:38:
PASSED:
  CHECK( a[i] == b[i] )
with expansion:
  21 == 21

partition.cpp:38:
PASSED:
  CHECK( a[i] == b[i] )
with expansion:
```

```
  78 == 78

partition.cpp:38:
PASSED:
  CHECK( a[i] == b[i] )
with expansion:
  89 == 89

partition.cpp:38:
PASSED:
  CHECK( a[i] == b[i] )
with expansion:
  87 == 87

partition.cpp:38:
PASSED:
  CHECK( a[i] == b[i] )
with expansion:
  97 == 97

===============================================================================
All tests passed (9 assertions in 1 test case)
```

In order to illustrate the changes occuring in memory i have made a table with records of all the changes made in the array during this function call and i have also included a GIF animation that shows memory changes step by step.

FIG.1 Array Change Table

Default	Change 1	Change 2	Change 3	Change 4	End
2	2	2	2	2	2
87	87	89	16	16	16
3	3	3	3	3	3
12	12	12	12	12	12
78	21	21	21	21	21
97	97	97	97	78	78
16	16	16	89	89	89
89	89	87	87	87	87
21	78	78	78	97	97

FIG.1 Step by Step by step gif

2 quicksort.cpp

```
#include <cstdlib>
#include <ctime>
#include <iostream>

using namespace std;

int partition(int a[], int from, int to)
{
    int pivot = a[from];
    int i = from - 1;    int j = to + 1;
    while (i < j)
    {
        i++; while (a[i] < pivot) { i++; }
        j--; while (a[j] > pivot) { j--; }
        if (i < j) { std::swap(a[i], a[j]); }
    }
    return j;
}


void quicksort(int a[], int from, int to)
{
    if (from >= to) { return; }
    int p = partition(a, from, to);
    quicksort(a, from, p);
    quicksort(a, p + 1, to);
}
```

3 select.cpp

Lab3 Select

```
#include <iostream>
#include <algorithm>
#include <stdlib.h>
#include <fstream>
#include "catch.hpp"
#include <chrono>


using namespace std;

int partition(int arr[], int lower, int upper, int pivotIndex);
void quicksort(int a[], int from, int to);
```

select

This is a recursive function that finds and returns the median value of the array. By definition, the median value is the middle value. this means it has an equal number of elements before it and after it in a sorted list. Even if the list is unsorted if a value in a list has an equal number of values greater than it and less than it, then that value is the median. You will se that by trying to find the median we actually end up sorting the array a little bit. not all the way sorted but it brings a quite a bit closer to sorting it.

@param arr[] : this parameter takes in the array whose median is to be found and returned.

@param pivotIndex : this parameter takes in the pivot index.

@param lower : this parameter takes in the starting index from which the median is supposed to be found.

```
    @param upper :  this parameter takes in the last index upto which the median is
    supposed to be found.

int select(int arr[], int pivotIndex, int lower, int upper)
{
        int p = partition(arr, lower, upper, pivotIndex);
        if (pivotIndex < p)
        {
                return select (arr, pivotIndex, 0, p);//recursive case
        }
        else if (pivotIndex > p)
        {
                return select(arr, pivotIndex, p+1, upper);//recursive case
        }
        else
        {
                return (arr[p]);//base case
        }
}


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

-------------------------------------------------------------------------------
Test Select
-------------------------------------------------------------------------------
select.cpp:84
...............................................................................

select.cpp:88:
```

```
PASSED:
  REQUIRE( select(a,n/2,0,n - 1) == 21 )
with expansion:
  21 == 21

select.cpp:89:
PASSED:
  REQUIRE( a[n/2] == 21 )
with expansion:
  21 == 21

===============================================================================
All tests passed (2 assertions in 1 test case)
```

Reading input from file

In order to read the input from a text file i used ifstream. ifstream is basically like cin but instead of taking input from the user it takes input from the opened text file, in this case shuffle.txt. It reads in the input and just stores everything in an array of size n (in this case 1,000,000). I'm using a large value for n so we can see the diference in times between our select function and the std libraries sort function and quicksort.

```
TEST_CASE("Select time")
{
        const int n = 1000000;

        //int a[] = {2,87,21,3,12,78,97,16,89,21};
        int a[n];

        ifstream file;
        file.open("shuffle.txt");
```

```
        for(int i = 0; i < n; i++)
        {
                file >> a[i];
        }
        select(a,n/2,0,n-1);
}

TEST_CASE("Sort time")
{
        const int n = 1000000;

        //int a[] = {2,87,21,3,12,78,97,16,89,21};
        int a[n];

        ifstream file;
        file.open("shuffle.txt");

        for(int i = 0; i < n; i++)
        {
                file >> a[i];
        }
        std::sort(a,a+n);
}

TEST_CASE("Quicksort time")
{
        const int n = 1000000;

        //int a[] = {2,87,21,3,12,78,97,16,89,21};
        int a[n];

        ifstream file;
```

```
        file.open("shuffle.txt");

        for(int i = 0; i < n; i++)
        {
                file >> a[i];
        }
        quicksort(a, 0, n-1);
}


TEST_CASE("Test Select")
{
        int a[] = {2,87,3,12,78,97,21,89,16};
        int n = (sizeof(a) / sizeof(a[0]));
        REQUIRE(select(a,n/2,0,n - 1) == 21);
        REQUIRE(a[n/2] == 21);
}



0.749 s: Select time
1.359 s: Sort time
0.951 s: Quicksort time
===============================================================================
test cases: 3 | 3 passed
assertions: - none -
```

Conclusion

You can see that in the test result selct is much faster than sort. almost twice as fast. It is also faster than quicksort but since the array is not fully sorted after select as it is after quicksort we cannot say that select if used in a sort function will be faster than quicksort. the sort in the std library is slow because it is not optimized. it is a an unoptimized version of quicksort. this a great example of how important optimizatation is. it reduced the time it took to sort by over 30%. this is the power of optimization.