# lab3x

Bhupinder Singh

March 8, 2019

## Contents

# 1 partition.cpp

```
Purpose - The purpose of this lab is to use the previous
lab and improve upon the documentation. In order to increase the quality
of the documentation the students are advised t ouse some sort of visual
aid, for my visual aid, I chose to use the "gdb" debugger.Lab 2 dove
into the topic of data structures and algorithms by introducing concepts
revolving around how much time an algorithm takes compareed to another.
After making changes to Lab2 and improving it, we will now compare the
time taken by select, quicksort and the built in sort function.

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 fixing both the functions, the student is expected to compare
the algorithm by a built in sorting function in the algorithm library and
the quicksort algorithm provided in the book.

The comparision should demonstrate which algorithm took the most time.
```

---

## 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 left = lower; int right = upper-2;

        int pivot = a[pivot_idx];
        we want the pivot o be 16, so we set the pivot to a[7]
        std::swap(a[pivot_idx], a[upper-1]);
        swap the pivot value and the end of the array to make it easierfor comparsion
```

```
        while(left <= right)
        {
                if(a[left] <= pivot) left++;
                else {std::swap(a[left], a[right]); right--;}
                putting all the values less than the pivot to the left of thearray and
                    the higher values to the right.
        }
        std::swap(a[left], a[upper-1]);
        now we know where to put the pivot value so we swap it with thevalue at "left"
            index.
        return left;
}
```

## 2 select.cpp

Lab2 Select Function

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);



In the previous lab, documentation for select was not as
good as it should be. So, before moving on to the function itself i would
like to share some visual aid that could help understand the select function
better.In the previous lab, our select was not working due to having extra
variables that were just not being used. After getting rid of them, we are
```

only left with p, k and b.

```
Breakpoint 1, select (arr=0x7fffffffd110, k=5, a=0, b=10) at select.cpp:38
38              if(k < p) select(arr, k, 0, p);  /*\textcolor{red}{recursive step}*/
1: p = 8
2: k = 5
3: b = 10
(gdb) n
```

At the begining, these are the current values of p, k and b.
K < P = true so, it should call select again recursively.

```
Breakpoint 1, select (arr=0x7fffffffd110, k=5, a=0, b=8) at select.cpp:38
38              if(k < p) select(arr, k, 0, p);  /*\textcolor{red}{recursive step}*/
1: p = 6
2: k = 5
3: b = 8
```

K < P = true so, it should call select again recursively.

```
Breakpoint 2, select (arr=0x7fffffffd110, k=5, a=0, b=6) at select.cpp:39
39              else if(k > p) select(arr, k, p+1, b);
1: p = 4
2: k = 5
3: b = 6
(gdb)
```

K < P = false, so it should go to the next line which is
else if(k > p) select(arr, k, p+1, b);

```
Breakpoint 3, select (arr=0x7fffffffd110, k=5, a=5, b=6) at select.cpp:40
40              else return arr[p]; //base case
1: p = 5
2: k = 5
3: b = 6
(gdb)
```

It keeps on looping through those lines until it reaches base case,
when it reaches base case, its eventually going to return arr[p].
I did not demonstrate all the screenshots for all the operations as that
would be too much. This screen shot is that when it reaches base case. After
it keeps reaching base from every call made the previous stack, eventually it
will return the median.

```
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);
        if(k < p) select(arr, k, 0, p);  recursive step
        else if(k > p) select(arr, k, p+1, b);
        else return arr[p]; //base case
}
```

Following is the code that the quicksort algorithm uses. For this lab,no documentation is required for the quicksort function.

```
int quick_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 = quick_partition(a, from, to);
    quicksort(a, from, p);
    quicksort(a, p + 1, to);
}
```

```
#ifdef UNIT_TEST
Test case "Select." This test case calls the selet function and checksif the values are
    accurate and if select actually calculated the median.
TEST_CASE("Select")
{
        const int n = 1000000;
        int a[n];
        for(int i = 0; i < n; i++)
        {
                a[i] = i;
        }
        std::random_shuffle(a, a+n);


        REQUIRE(select(a,n/2,0,n)==n/2);
}
#endif
```

```
Test case "quick." This test case calls the quicksortfunction given to us by the
    instructor.

TEST_CASE("quick")
{
        const int f = 1000000;
        int a[f];
        for(int i = 0; i < f; i++)
        {
                a[i] = i;
        }
        std::random_shuffle(a, a+f);


        quicksort(a,0,f-1);
        REQUIRE(a[f/2]==f/2);

}
```

Test case "sort." This test case calls the built in sortfunction.

```
TEST_CASE("sort")
{
        const int h = 1000000;
        int c[h];
        for(int i = 0; i < h; i++)
        {
                c[i] = i;
        }
        std::random_shuffle(c, c+h);


        std::sort(c, c+h);
        REQUIRE(c[h/2]==h/2);

}
```

This is the main and last part of our program. Our main goal in this program was to test the speed of the three algorithms. The following pictures show how the built-in sort() is s lower than the quicksort algorithm. But the fastest one is still select. Even though select is the fastest algorith in here I am still not sure if it is a good comparison because the other two algorithms are sorting a list while select is just finding the median of a list, so I am not sure if this is a valid comparison.

QUICKSORT TIME

```
PASSED:
  REQUIRE( a[f/2]==f/2 )
with expansion:
  500000 (0x7a120) == 500000 (0x7a120)

0.481 s: quick
===============================================
All tests passed (1 assertion in 1 test case)
```

BUILT-IN SORT FUNCTION TIME

```
PASSED:
  REQUIRE( c[h/2]==h/2 )
with expansion:
  500000 (0x7a120) == 500000 (0x7a120)

0.518 s: sort
===============================================
All tests passed (1 assertion in 1 test case)
```

SELECT TIME

```
PASSED:
  REQUIRE( select(a,n/2,0,n)==n/2 )
with expansion:
  500000 (0x7a120) == 500000 (0x7a120)

0.259 s: Select
===============================================
All tests passed (1 assertion in 1 test case)
```

Also, I have a request for the next lab. I personally feel that the next lab should not be based upon this lab at all. I feel that is a better way for students to comprehend the code better. We are moving onto new topics and that should reflect in our codes.