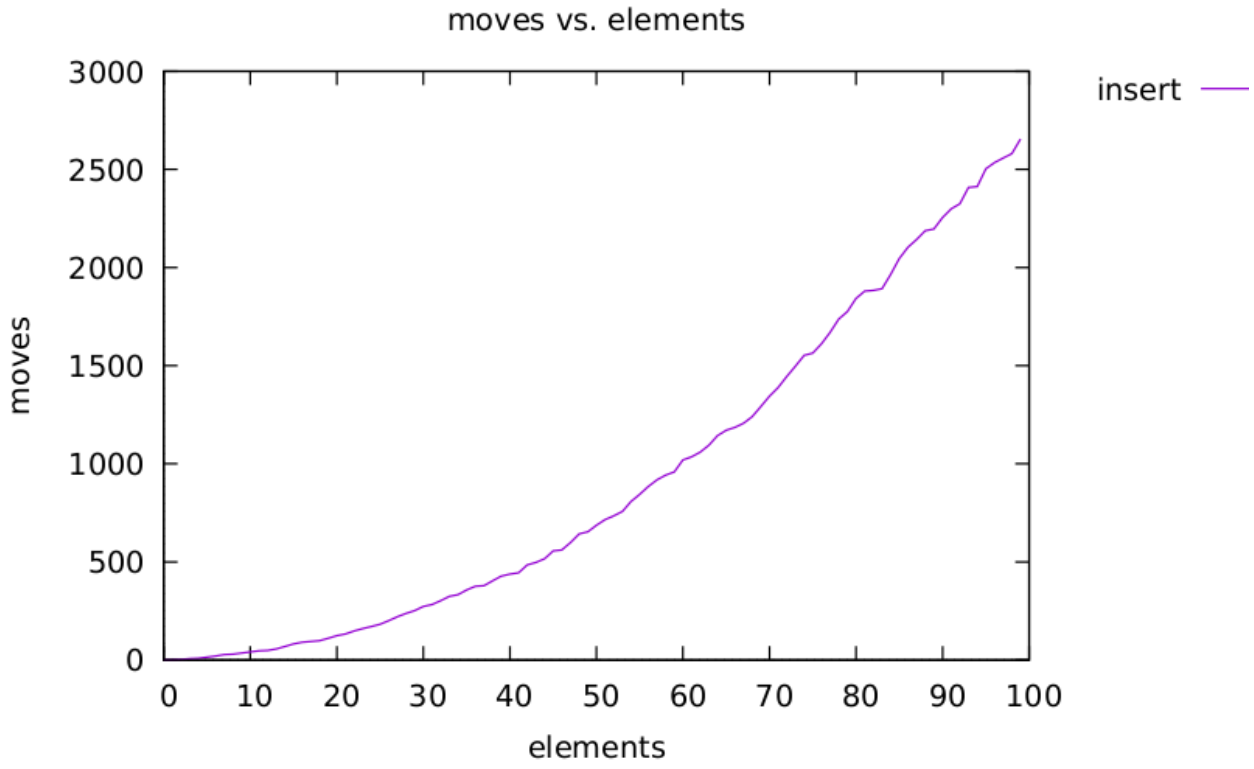
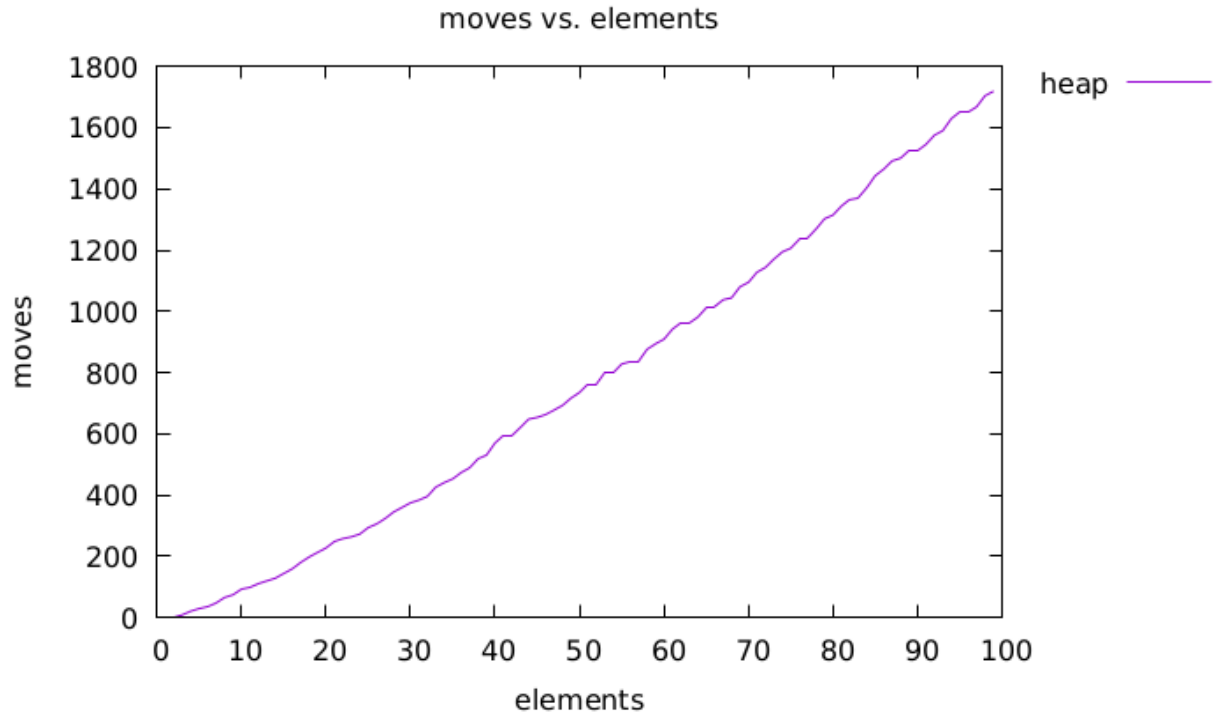
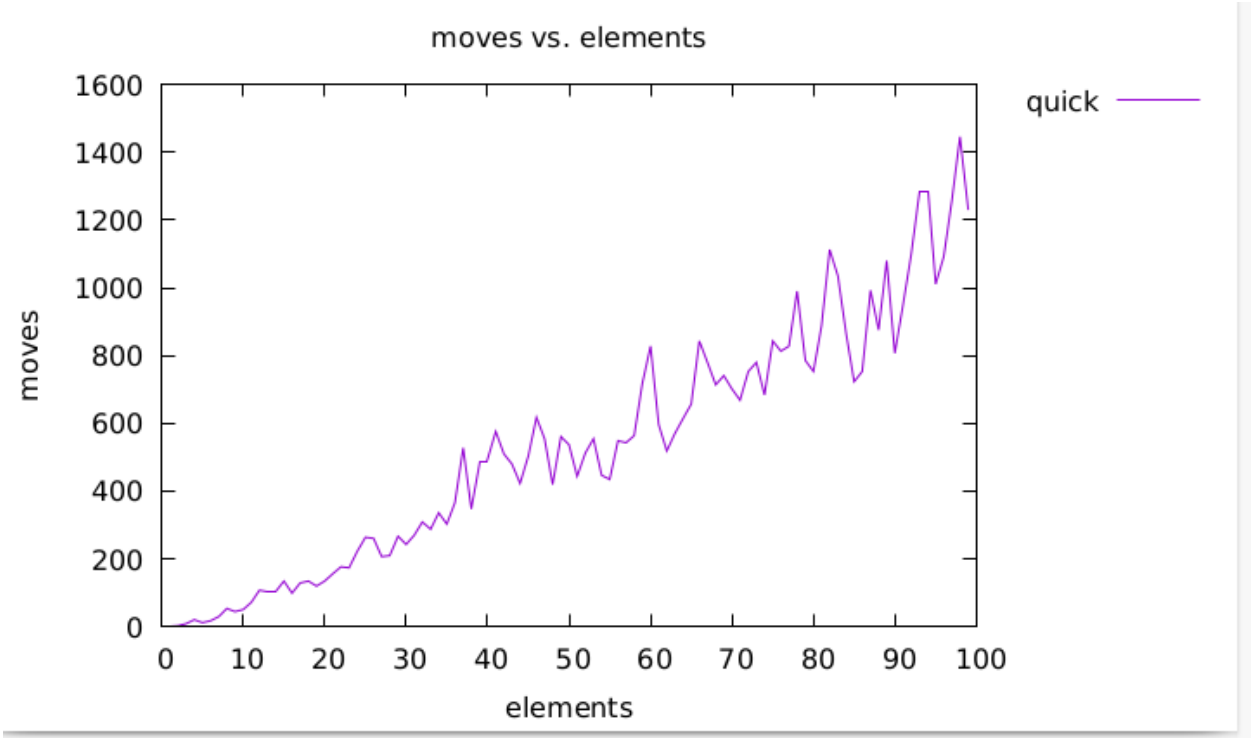

moves vs. elements
moves
elements
insert
0
500
1000
1500
2000
2500
3000
0
10
20
30
40
50
60
70
80
90
100


moves vs. elements
moves
elements
heap
0
200
400
600
800
1000
1200
1400
1600
1800
0
10
20
30
40
50
60
70
80
90
100




```
wichapas@wichapas-VirtualBox:~/wpichetp/asgn3$ ./sorting -a -p 0
Heap Sort, 100 elements, 1755 moves, 1029 compares
Shell Sort, 100 elements, 1183 moves, 801 compares
Insertion Sort, 100 elements, 2741 moves, 2638 compares
quick Sort, 100 elements, 1053 moves, 640 compares
```

```
wichapas@wichapas-VirtualBox:~/wpichetp/asgn3$ ./sorting -a -p 0 -n 1000
Heap Sort, 1000 elements, 27225 moves, 16818 compares
Shell Sort, 1000 elements, 19323 moves, 13394 compares
Insertion Sort, 1000 elements, 254769 moves, 253765 compares
quick Sort, 1000 elements, 18642 moves, 10531 compares
```

```
wichapas@wichapas-VirtualBox:~/wpichetp/asgn3$ ./sorting -a -p 0 -n 10000
Heap Sort, 10000 elements, 372558 moves, 235318 compares
Shell Sort, 10000 elements, 313890 moves, 234418 compares
Insertion Sort, 10000 elements, 24901706 moves, 24891699 compares
quick Sort, 10000 elements, 256734 moves, 149913 compares
wichapas@wichapas-VirtualBox:~/wpichetp/asgn3$
```

So in this lab I have learned a lot of things. Starting off, the use of atoi() is somewhat handy for getting number inputs after writing something like -n. I believe this will be helpful in the future labs. Next, I have learned that the sorting algorithms and their efficiency, complexity totally determines how long the sorting process takes. At first you might not notice the difference, but as the number of

elements in an array grows you will start to see that the sorting algorithm will also vary.

As I suspected, the insertion sort algorithm over the span of 100 elements has shown a poor performance of requiring about 2600 moves. If you observe the shape of the line from the first graph, you can see that it started off performing better with a smaller amount of elements. But gradually, the number of moves increased exponentially at around 50 elements. The spike in sudden increase in number of moves tells us that as the number of element increases, this will become an algorithm not suited for sorting large numbers of elements. My theory is further proven by the last screen shot shown where element was increased to 1000. Insertion sort is move count is blown out of proportion.

Next, if we take a look at the second photo, we can see that heap sort has steady growth, unlike insertion sort. What this tells us right off the bat is that while increasing the number of elements, the moves won't increase exponentially but rather at a steady rate. Also, at 100 elements heap sort was only around 1700 moves compared to insertion 2700 moves. The shape of the graph tells us a lot about the behavior of how the number of moves may increase varying for each algorithm. Also observe how at 10000 elements, insertion was at 2500k moves while heap is 370k.

Quick sort, famously known for being an extremely fast sorting algorithm. However, this graph is very different compared to others. As you can see from the third photo, we can see a lot of fluctuations. With increasing number of elements, for some reason it might take more moves or it can also take less amount of moves. I suspect that this is because due to the nature of quick sort, some pattern of arrays might allow the quick sort to easily sort than other patterns thus causing the fluctuations. Because of this fluctuation, this makes quick sort worst than the other type of sorts in the early elements. However, when it reaches ridiculous number of elements, quick sort is the definite winner as seen in the last photo.