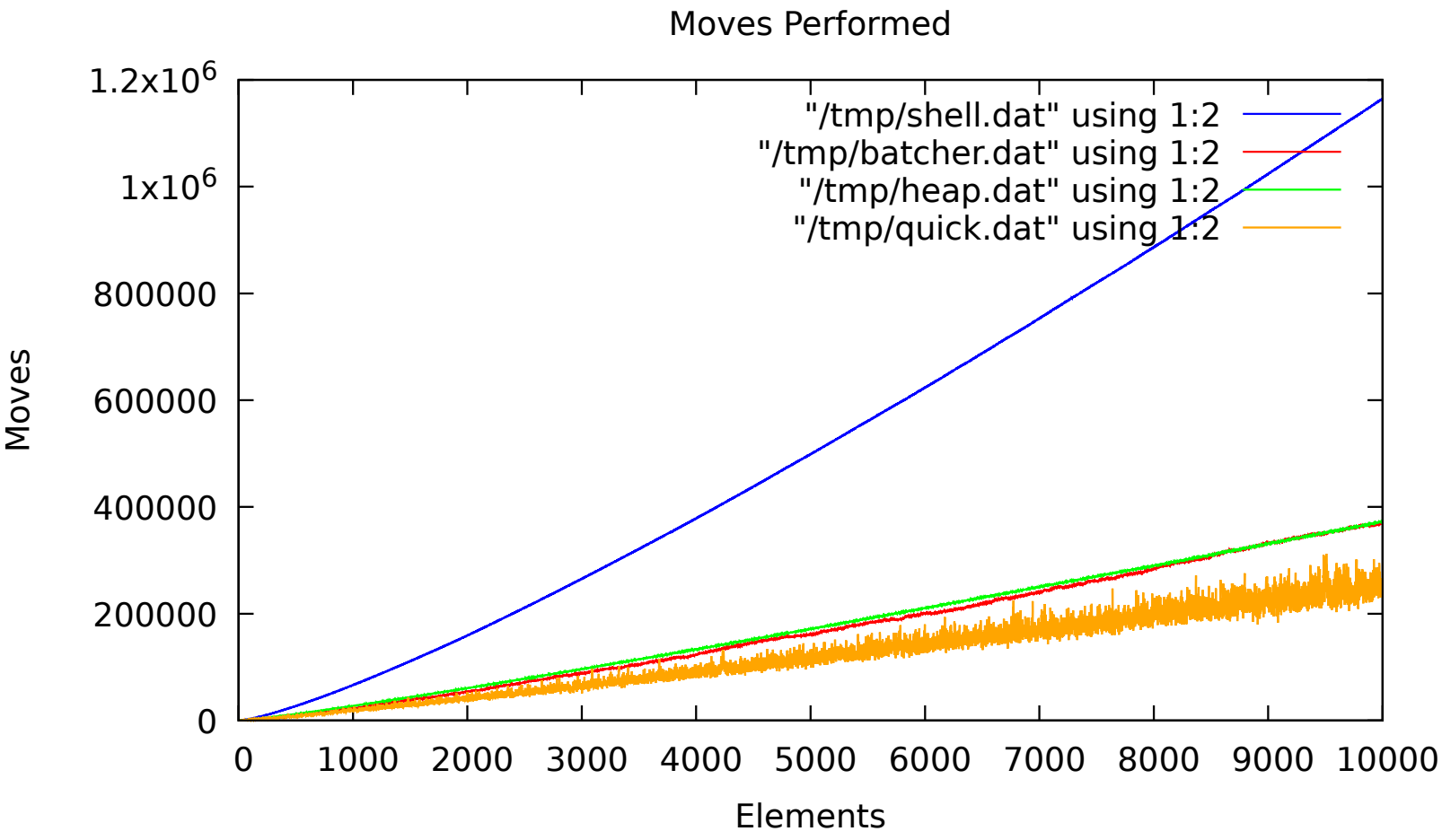
WRITEUP - ASGN3

Viveka Agrawal

CSE13S, Winter 2023

In this assignment, I learned how to code and implement different sorting algorithms These algorithms include Shell sort, Batcher sort, Heap sort, and Quick sort.

Here is my Sorting pdf (next page):


Moves Performed
Moves
Elements
"/tmp/shell.dat" using 1:2
"/tmp/batcher.dat" using 1:2
"/tmp/heap.dat" using 1:2
"/tmp/quick.dat" using 1:2
1.2x10^6
1x10^6
800000
600000
400000
200000
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

The blue line represents Shell sort, the red line represents Batcher sort, the green line represents Heap sort, and the orange line represents Quick sort.

For Shell sort, as the number of elements increases, the number of moves increases much more rapidly. This is because the time complexity of Shell sort depends on the gap sequence given in gaps.h. The best-case time complexity for Shell sort is O(nlog(n)). Based on the graph generated by my program, for an increasing array size from 0 to 10,000 elements, Shell sort requires the largest number of moves to sort the array compared to the other 3 sorting algorithms. Shell sort is also the only graph to have a relatively smooth line. This is due to the fact that as the number of elements increases, the number of moves also mostly increases. When the number of moves decrease when the array size increases, the difference is not by much as shown in my example below. In other words, the number of moves it takes to sort the array does not fluctuate/oscillate as much as the other sorting methods do as the number of elements increases. The number of moves in Shell sort also seems to be increasing exponentially as the size of the array to sort increases. For Shell sort, I observed the following from running my program:

Shell Sort:

Size of the array: 10, Number of Moves Required: 85. Size of the array: 11, Number of Moves Required: 99. (I observed that there is an increase in 14 moves when the array size increases by 1). Size of the array: 12, Number of Moves Required: 113. (I observed that there is an increase in 14 moves when the array size increases by 1). Size of the array: 13, Number of Moves Required: 132. (I observed that there is an increase in 19 moves when the array size increases by 1). Size of the array: 14, Number of Moves Required: 149. (I observed that there is an increase in 17 moves when the array size increases by 1). Size of the array: 15, Number of Moves Required: 170. (I observed that there is an increase in 21 moves when the array size increases by 1).

As shown in the graph, Batcher sort and Heap sort mainly take the same number of moves to sort the elements in a given array. The number of moves in Batcher sort and Heap sort seem to be increasing linearly as the size of the array to sort increases. Compared to Shell sort, both Batcher sort and Heap sort are much more efficient as they require less moves to sort the array. Unlike Shell sort, their lines are not smooth but rather fluctuate/oscillate, but they do not do so as much as Quick sort's line does. The time complexity for both sorts are the same, O(nlog(n)). For Batcher sort, I observed the following from running my program:

Batcher Sort:

Size of the array: 10, Number of Moves Required: 39. Size of the array: 11, Number of Moves Required: 51 (Here I observed that there is an increase in 12 moves when the array size increases by 1). Size of the array: 12, Number of Moves Required: 51 (Here I observed that the number of moves required to sort an array of size 11 is the same as the number of moves required to sort an array of size 12). Size of the array: 13, Number of Moves Required: 60. (Here I observed that there is an increase in 9 moves when the array size increases by 1). Size of the array: 14, Number of Moves Required: 69. (Here I observed that there is an increase in 9 moves when the array size increases by 1). Size of the array: 15, Number of Moves Required: 90 (Here I observed that there is a sudden jump in the number of moves required to sort an array of size 15 compared to an array of size 14).

Heap Sort:

Size of the array: 10, Number of Moves Required: 93. Size of the array: 11, Number of Moves Required: 99 (Here I observed that there is an increase in 6 moves when the array size increases by 1). Size of the array: 12, Number of Moves Required: 111 (Here I observed that there is an increase in 12 moves when the array size increases by 1). Size of the array: 13, Number of Moves Required: 120 (Here I observed that there is an increase in 9 moves when the array size increases by 1; 9 moves is less than 12, which explains why the Heap sort graph fluctuates). Size of the array: 14, Number of Moves Required: 144 (Here I observed that the number of moves required is increasing again as the size of the array increases). Size of the array: 15, Number of Moves Required: 90 (Here I observed that the number of moves required has decreased by 54 moves, which explains why the Heap sort graph fluctuates).

Quick sort is the most efficient sorting method based on the graph generated from running my program. It also has the largest fluctuations in the number of moves required as the size of the array increases. Also, I noticed that for an already almost sorted array, Quick sort performs the worst in the sense that it takes the most moves to sort the array compared to the other 3 sorting methods. Quick sort also has the worst time complexity, O(n squared). For Quick sort, I observed the following from running my program:

Quick Sort:

Size of the array: 10, Number of Moves Required: 51. Size of the array: 11, Number of Moves Required: 72 (Here I observed that there is an increase in 21 moves when the array size increases by 1). Size of the array: 12, Number of Moves Required: 108 (Here I observed that there is an increase in 36 moves when the array size increases by 1). Size of the array: 13, Number of Moves Required: 102 (Here I observed that there a decrease in 6 moves when the array size increases by 1). Size of the array: 14, Number of Moves Required: 102

(Here I observed that the number of moves required to sort an array of size 13 is the same as the number of moves required to sort an array of size 14). Size of the array: 15, Number of Moves Required: 135 (Here I observed that there is a sudden jump in the number of moves required to sort an array of size 15 compared to an array of size 14).