Yash Sharma
CSE13S, Winter 2021

Assignment 3 - WRITEUP.pdf

Introduction:

This assignment requires us to implement sorting algorithms like Insertion Sort, Batcher Sort, Heap Sort, and recursive Quicksort for the purpose of us getting fully familiarized with each sorting algorithm. Through this assignment, the program will also sort data using comparisons based on the number of elements needed to be sorted. For example, if the set of elements to be sorted is small, a counting sort is rather to be implemented as it would count the number of occurrences of each element in an array. In addition, by developing this program we will get a feel for computational complexity.

The Insertion Sort function is a sorting algorithm that considers elements one at a time, placing them in their correct ordered position. This algorithm is quite basic and utilizes an array in increasing order to check if A[k] is in the correct order by comparing it with the element A[k-1]. To create this function in C, I created a for loop that takes in three parameters: Statistics, Array[A], and length of the array. This will check if the array is in increasing order along with sorting the array based on the range between 1 and the length of the array. Once it is completed with sorting, the array is then printed with the statistics and temporary value resulting from the program.

The Batcher Sort function is a sorting network built with a fixed number of wires, one for each input to sort, and is connected using comparators. This sorting network will sort any input using a fixed number of comparisons using Batcher's method. His sorting method involves k-sorting the even and odd subsequence referring to the sequence of values. For example, in an array of 16 elements, Batcher's method first 8-sorts the even and odd subsequences, then 4-sorts them, then 2-sorts them, and finally 1-sorts them, finally merging the sorted even and odd sequences. In the code, we created multiple functions for batcher sorts that would work in parallel to the main program: bitlength, comparator, and batcher sort. The bitlength function counts the number of bits being processed through a while loop that checks the length of the array. The comparator function does a similar task, but instead, it swaps the places of the arrays if one is

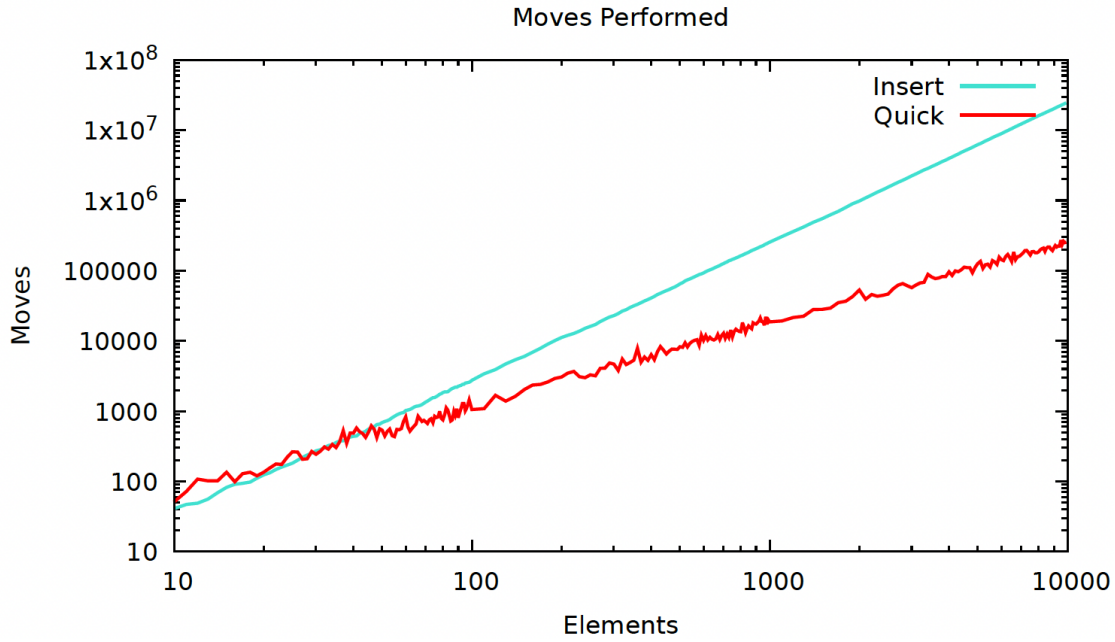
greater than the other. Finally, the batcher sort function is the main function that calls the bitlength and comparator functions to sort the data.

The Heap Sort function is implemented in this program as an array, in which for any index k, the index of its left child is 2k and the index of its right child is 2k + 1. Therefore, the parent index of any index k should be k/2. In this program, we utilize the max heap component as any parent node within a max heap must have a value that is greater than or equal to the values of its children. First, we utilize the ordering of the array elements for our heap sort so then the root of the heap becomes the first element of the array. In the code, the functions used for this were: max child, fix heap, build heap, and heap sort. The max child function sorts and sets the left and right variables to equal max child. In the fix heap function, the while loop checks the array and swaps mother and great indexes as necessary. The build heap function is designed to call fix heap within a for loop, and for the heapsort function, this is the main function that builds/fixes the heap.

The Quick Sort function is the fastest known algorithm that sorts using comparisons versus its competitors: merge and heapsort. Although this is a fast sorting algorithm, it is also a divide and conquers type of algorithm. The quicksort algorithm is designed to partition arrays into two sub-arrays by selecting an element from the array and designating it as a pivot. The elements in the array that are less than the pivot go to the left sub-array and elements in the array that are greater than or equal to the pivot go to the right sub-array.

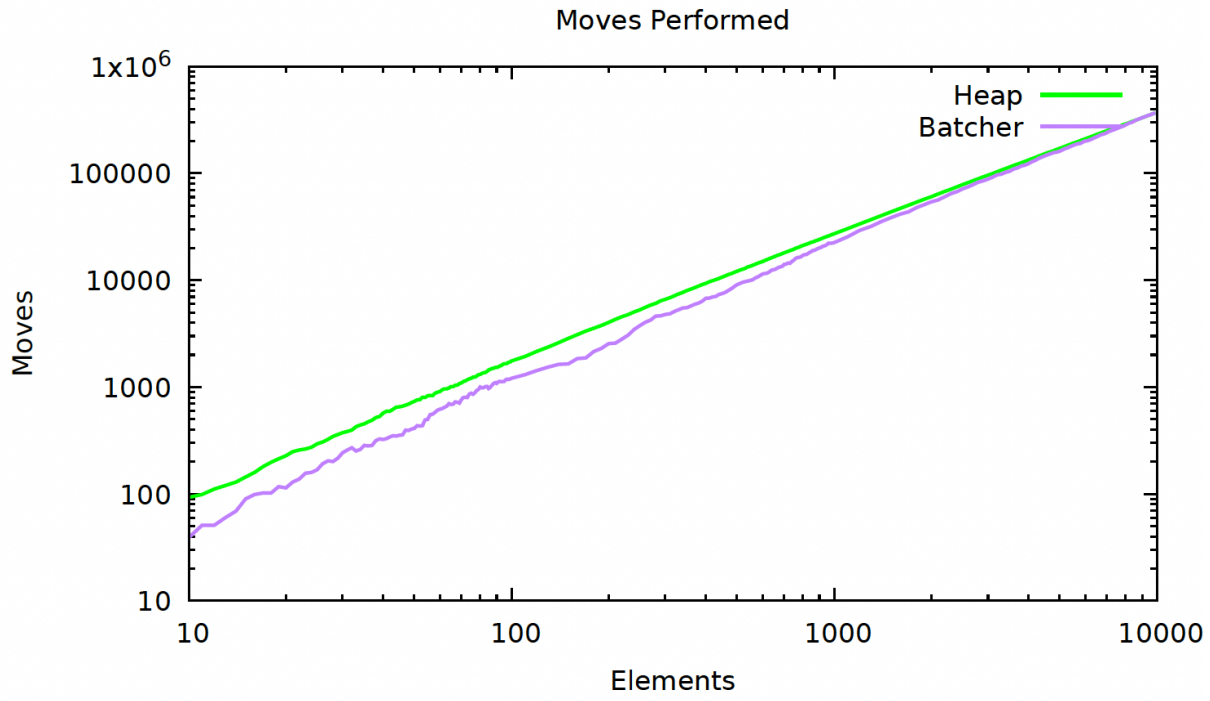
Sorting Graphs:

Insertion Sort vs Quick Sort



This was the data produced by the correlation between the elements and moves of the quick & insertion sort of a random set of data. To produce this graph, I wrote the functions for a quick and insertion sort. For quick sort, I wrote a function called partition that could contain multiple parameters like stats, array, low, and high. I also wrote two other functions called quick_sorter and quick_sort that could sort the partition values and call the function to do sorting. At the end, all this data is sent to a separate output file which then GNUPlot uses to create a graph. GNU Plot takes in the xlabel, ylabel and plots the graph according to the output file (sorting.pdf). From this graph, we can conclude that the jagged lines produced by the quick graph state there is some inconsistency in the graph. Along with this, the straight line produced by the insertion sort shows there are consistent values being generated.

Heap Sort vs Batcher Sort:



This was the data produced by the correlation between moves and elements for the heap and batcher sort. To produce this graph, I created multiple functions within heap sort such as max_child, fix_heap, build_heap, and heap_sort. Each of these functions helped generate the heap graph as they were used to sort through the first and last variables along with used to build a heap sort. For the batcher graph, I utilized bit length, comparator, and batcher__sort for my functions. The bit length function counts the amount of bits being processed through an array. In addition, the comparator and batcher_sort functions work hand in hand as the comparator one swaps arrays while the batcher_sort function contains all the sorting algorithms. After this, the data is then sent through GNUPlot which plots this data based on the xlabel, ylabel, and a yrange [0:10000] in the data file. From this graph, we can conclude that the batcher sort function is more inconsistent in resulting values versus the heap sort function due to the jagged lines produced from the moves vs elements comparison.

Conclusion:

From the data shown here, we can draw 3 major conclusions:

1. Quick sort is the fastest and most inconsistent sorting algorithm
2. Heap function is a bit slow, but fairly consistent sorting algorithm
3. Batcher is another fast but inconsistent sorting algorithm
4. Insert is the slowest but most consistent sorting function