Writeup
Assignment 5

I ran the program with different array size values. I started with 8 and each subsequent run increased the size of the array by 4 times. Then I collected the resulting data into a table.

Table 1: moves and comparisons obtained on random data of different sizes

Sort	Bubble Sort		Shell Sort		Quick Sort		Heap Sort	
Size	moves	compares	moves	compares	moves	compares	moves	compares
8	39	27	21	53	21	38	51	27
32	768	495	150	1234	102	233	408	229
128	12432	8092	1038	22687	642	1426	2310	1389
512	186963	129735	5613	382238	3201	6629	12471	7641
2048	3181128	2095225	29904	6231387	15648	33302	61842	38661
4096	50624697	33549595	149631	100329955	74589	156696	297441	187637
32768	806836779	536843203	717399	1608463573	340353	710674	1386879	881901

By examining this data, I can find out that when the array increases in size by 4 times, the number of moves and comparisons increases by:

Table 2

Bubble Sort		Shell Sort		Quick Sort		Heap Sort	
moves	compares	moves	compares	moves	compares	moves	compares
15 - 20	16 - 18	5 - 7	23 - 16	4.5 - 5	4.5 - 6	4.5 - 8	4.5 - 8

From this table I can conclude that the growth of comparisons and moves for Bubble Sort corresponds to quadratic function $T = n^2$. For Shell Sort, the number of comparisons also grows according to quadratic function, but the number of moves grows much more slowly and corresponds to a function $T = n \lg n$. For both algorithms Quick and Heap Sort, the growth of both moves and comparisons corresponds to function $T = n \lg n$.

Here are two diagrams that I made in Excel based on this data. It shows that:

- Only Bubble sort has a sharp increase in the number of moves.
- The number of comparisons in Shell sort is growing even faster than in Bubble sort.
- The number of operations for Quick and Heap is invisible on the graph compared to the other two sorts.

Figure 1

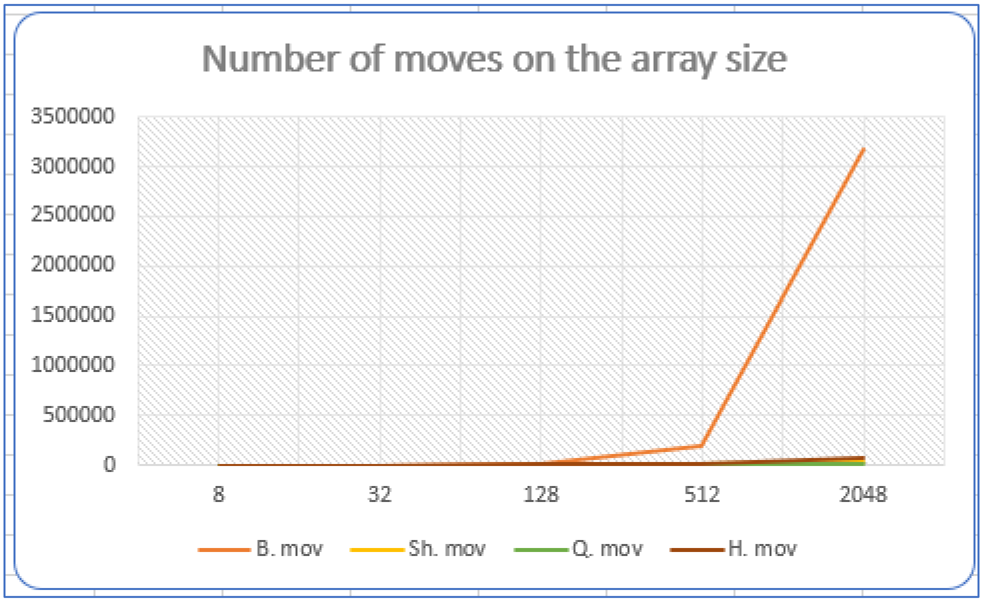
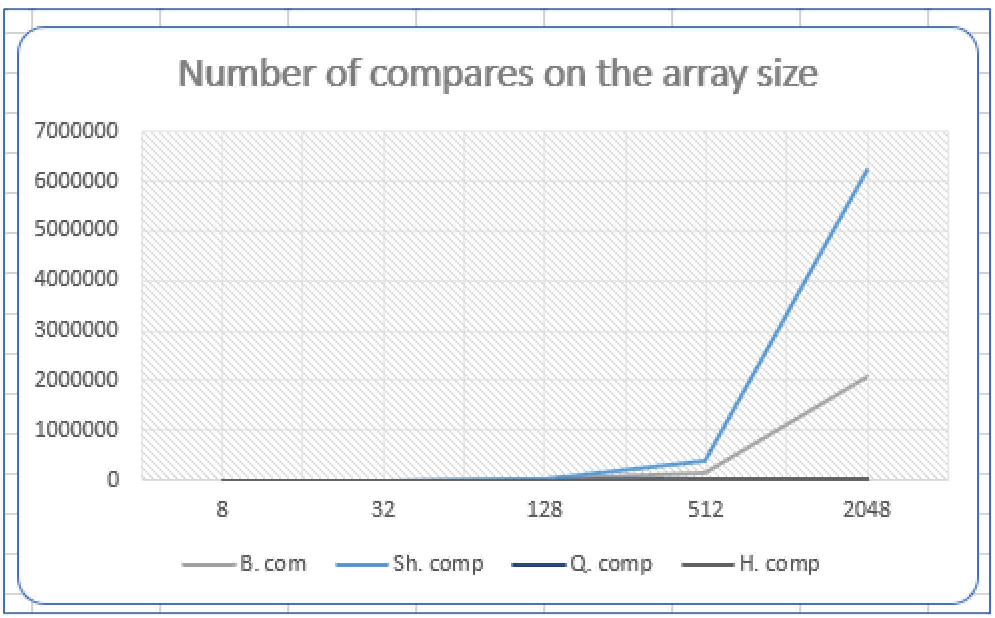


Figure 2



After testing the random data, I tested three more kinds of data — ascending and descending, and data in which all elements are the same. I have implemented three additional options in the *sorting* program to create and test such arrays. These are "-O", "-R" and "-E", respectively.

Table 3: moves and comparisons obtained on ordered data of different sizes (ascending order)

Sort	Bubble Sort		Shell Sort		Quick Sort		Heap Sort	
Size	moves	compares	moves	compares	moves	compares	moves	compares
8	0	7	0	53	0	31	66	27
128	0	127	0	22687	0	1023	2550	1459
2048	0	2047	0	6231387	0	24575	65592	40204
32768	0	32767	0	1608463573	0	524287	1450884	908636

Ascending data is the best case for all algorithms except Heap sort. For Heap sort, this turned out to be the worst case.

Table 4: moves and comparisons obtained on ordered data of different sizes (descending order)

Sort	Bubble Sort		Shell Sort		Quick Sort		Heap Sort	
Size	moves	compares	moves	compares	moves	compares	moves	compares
8	84	28	18	53	12	32	48	24
128	24384	8128	444	22687	192	1024	2106	1294
2048	6288384	2096128	9516	6231387	3072	24576	57762	36973
32768	1610563584	536854528	187794	1608463573	49152	524288	1319508	854980

Descending data is the worst case for Bubble Sort.

Table 5: moves and comparisons obtained on data where all elements are the same

Sort	Bubble Sort		Shell Sort		Quick Sort		Heap Sort	
Size	moves	compares	moves	compares	moves	compares	moves	compares
8	0	7	0	53	36	38	21	18
128	0	127	0	22687	1344	1150	381	378
2048	0	2047	0	6231387	33792	26622	6141	6138
32768	0	32767	0	1608463573	737280	557054	98301	98298

This is also an interesting table. It turns out to be the worst case for Quicksort and the best case for Heap sort.