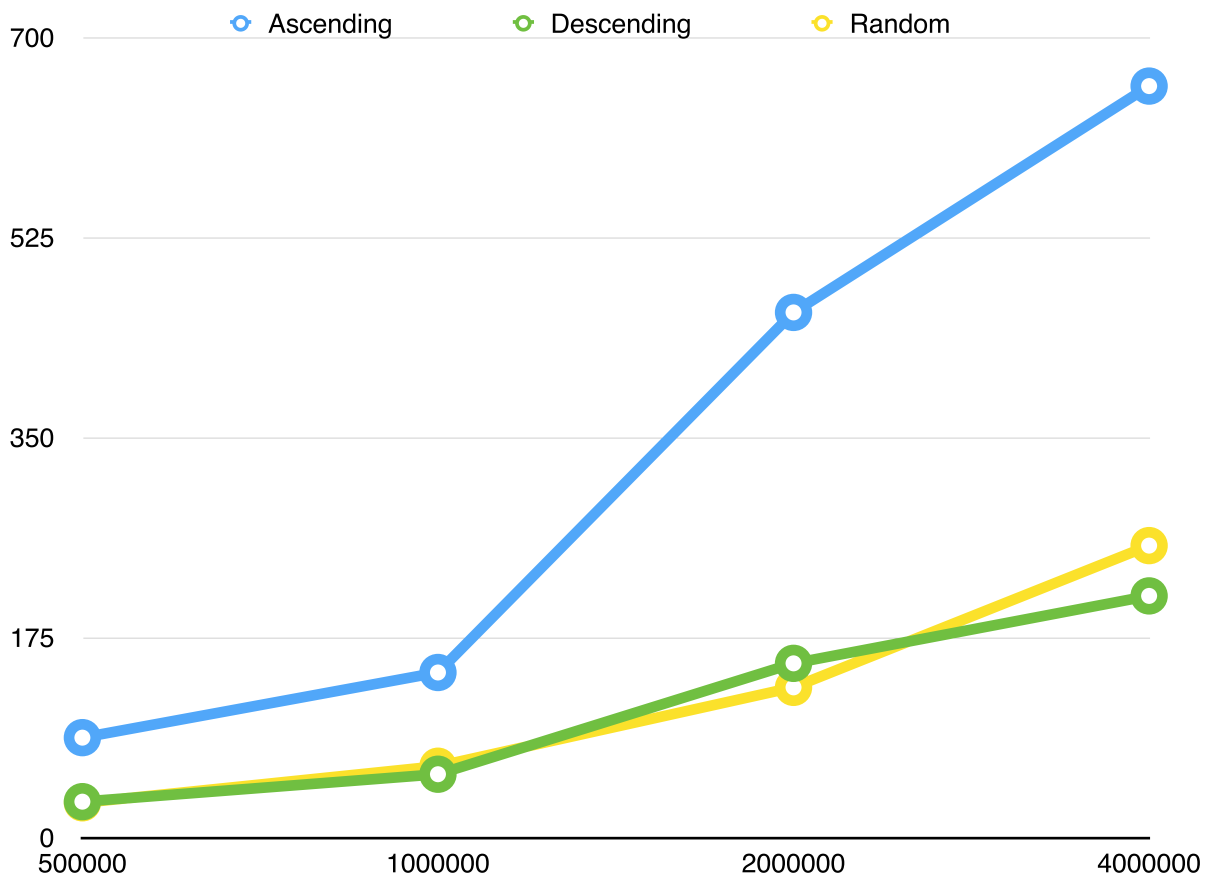
Test 1

		500000	1000000	2000000	4000000
🟦	Ascending	87	144	459	657
🟩	Descending	31	55	152	211
🟨	Random	30	62	131	255

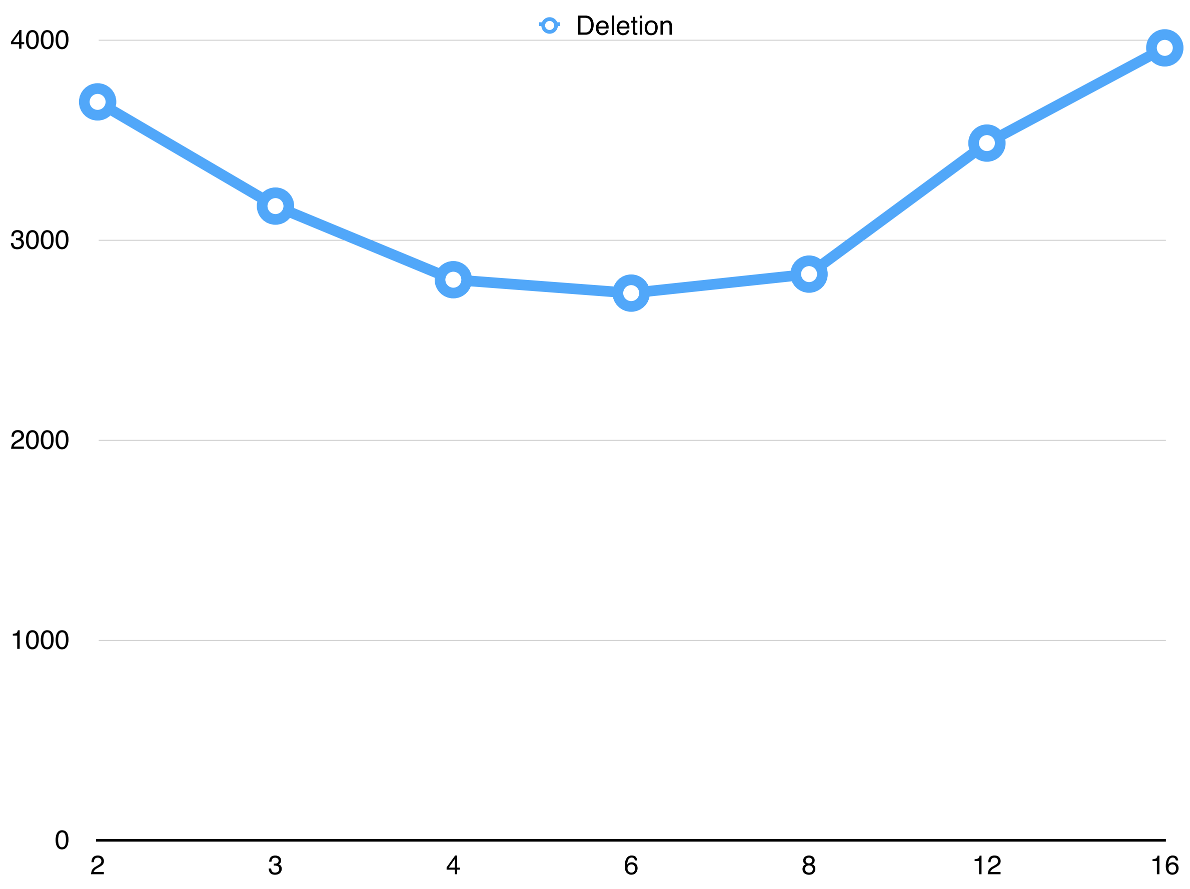


Which order (among ascending, descending and random) gives you the best performance? Why? Explain the reason using your runtime data (table or line graph) and your knowledge about their time complexity.

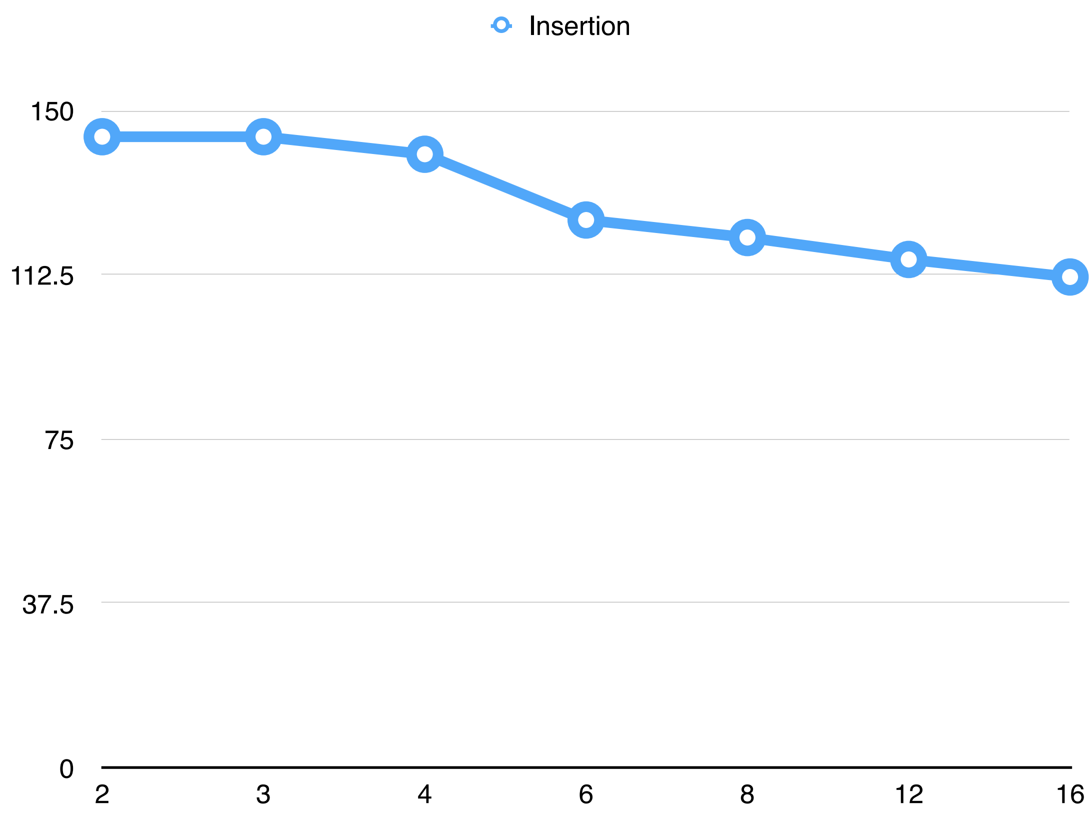
Ascending performs worse, while descending performs the best. Since this is a max heap, the larges elements should be stored in the root. In a descending order, the largest elements are stored first, and all remaining elements require no bubble upping when inserted, hence the faster significantly faster runtime compared to an ascending array.

Test 2

2	3	4	6	8	12	16
3685	3164	2796	2729	2824	3479	3955



2	3	4	6	8	12	16
144	144	140	125	121	116	112



Which branching factor gives you the best performance for insertion and deletion (one for each)? Why? Explain the reason using your runtime data (table or line graph) and your knowledge about their time complexity.

The branching factor of 16 gives the best runtime for insertion, while the branching factor of 6 gives the best runtime for deletion. Since insertion relies on bubbling up, the higher the branching factor, the less bubbling up an element has to do when being inserted, if any, lowering the runtime. On the other hand, for deletion, since the trickling down process relies on comparing the parent with each children, when increasing the branching factor to a certain point, the decreased trickling down provides a lower runtime, however, when the branching factor is too high, high amount to comparisons have to be made with the parent, increasing runtime and creating a parabolic shape.

Compare your outcome with the description in Part 2.1. Is there any discrepancy between the theoretical time complexity and the real empirical runtime?

The slower deletion times seem to match up with the description provided in 2.1, alongside with the faster insertion time provided with a larger branching factor.