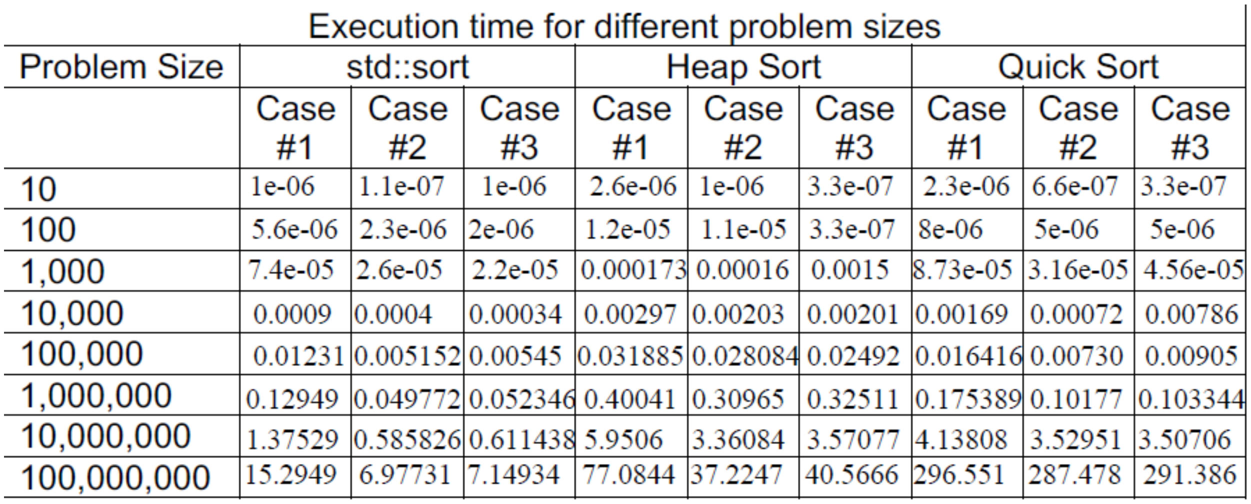
Based on the execution time in the table above, answer the following questions and include them in the report along with the table above:

a. For each of the three sort methods, what pattern do you observe in the execution time as the problem size is increased for different test cases (in other words, if you look at each column in the table above, do you see any patterns in execution time as related to the problem size)? Did you expect to see this pattern? If so, why? If not, then what did you expect to see?
    a. The execution time for std::sort increases roughly in a linear fashion as the problem size grows. This is expected, as std::sort is implemented using efficient sorting algorithms, such as quicksort, which have average-case time complexities of O(n log n). Therefore, we expect the execution time to grow proportionally to the problem size.
    b. Heap sort shows a clear pattern of increasing execution time with larger problem sizes. However, the increase in execution time is steeper compared to std::sort. Heap sort has a general time complexity of O(n log n), which explains the growth in execution time. The steeper increase may be due to the additional overhead of maintaining the heap structure.
    c. Quick sort also exhibits a pattern of increasing execution time as the problem size grows. However, quick sort seems to perform better than heap sort for the most of the problem sizes. Quick sort has an average-case time complexity of O(n log n) and often performs well in practice due to its good cache locality and low constant factors. I did not expect quick sort to take longer when introduced to 100 million inputs.
    d. These patterns align with expectations based on the time complexities of the sorting algorithms, except for how long it takes quick sort to sort 100 million inputs in all cases. I expected to take roughly the same time as std::sort.

b. How does the execution time of heap sort compare with execution time of quick sort for different problem sizes and different test cases? Explain the difference in execution time.
    a. For smaller problem sizes, heap sort and quick sort have similar execution times, with quick sort sometimes being slightly faster. As the problem size increases, quick sort starts to outperform heap sort significantly. This is because quick sort has better average-case performance compared to heap sort, especially when the data is partially sorted or randomly distributed. For the largest problem size (100,000,000), the performance difference between quick sort and heap sort is substantial, with quick sort taking much longer. This is not consistent with the quick sort. Quick sort is often much quicker than heapsort, according to empirical studies. As a result, quick sort should often offer significantly improved performance when an occasional "blowout" to O(n^2) is acceptable, especially if one of the updated pivot choosing techniques is applied. IN this case, I suspect that quick sort is somehow degrading to O (n^2).

c. How does the performance of your heap sort and quick sort implementations compare with the performance of the std:sort? Explain any similarities or differences you observe. What could be the potential reasons for these similarities and differences?
    a. For smaller problem sizes, the custom heap sort and quick sort implementations are competitive with std::sort, with execution times in the same order of magnitude. This suggests that my implementations are reasonably efficient for small input sizes. As the problem size increases, std::sort consistently outperforms both custom implementations. This is likely because the standard library's sorting algorithm is highly optimized and may employ various techniques and optimizations that my custom implementations lack. The custom implementations of heap sort and quick sort show similar performance patterns, with quick sort being generally faster than heap sort for most of the larger problem sizes. One potential reason for the differences in performance between std::sort and my custom implementations is the use of advanced optimizations and algorithms in the standard library. Additionally, my custom implementations may lack certain optimizations, such as parallelism and adaptive sorting algorithms, that std::sort can leverage.



Execution time for different problem sizes

Problem Size	std::sort			Heap Sort			Quick Sort		
	Case #1	Case #2	Case #3	Case #1	Case #2	Case #3	Case #1	Case #2	Case #3
10	1e-06	1.1e-07	1e-06	2.6e-06	1e-06	3.3e-07	2.3e-06	6.6e-07	3.3e-07
100	5.6e-06	2.3e-06	2e-06	1.2e-05	1.1e-05	3.3e-07	8e-06	5e-06	5e-06
1,000	7.4e-05	2.6e-05	2.2e-05	0.000173	0.00016	0.0015	8.73e-05	3.16e-05	4.56e-05
10,000	0.0009	0.0004	0.00034	0.00297	0.00203	0.00201	0.00169	0.00072	0.00786
100,000	0.01231	0.005152	0.00545	0.031885	0.028084	0.02492	0.016416	0.00730	0.00905
1,000,000	0.12949	0.049772	0.052346	0.40041	0.30965	0.32511	0.175389	0.10177	0.103344
10,000,000	1.37529	0.585826	0.611438	5.9506	3.36084	3.57077	4.13808	3.52951	3.50706
100,000,000	15.2949	6.97731	7.14934	77.0844	37.2247	40.5666	296.551	287.478	291.386