**Assignment 1**

**Question 1**:Coin Toss Simulation

Through a coin toss simulation, show that probability of getting HEAD, by tossing a fair coin, is about 0.5. Write your observation from the simulation run.

**Solution**:

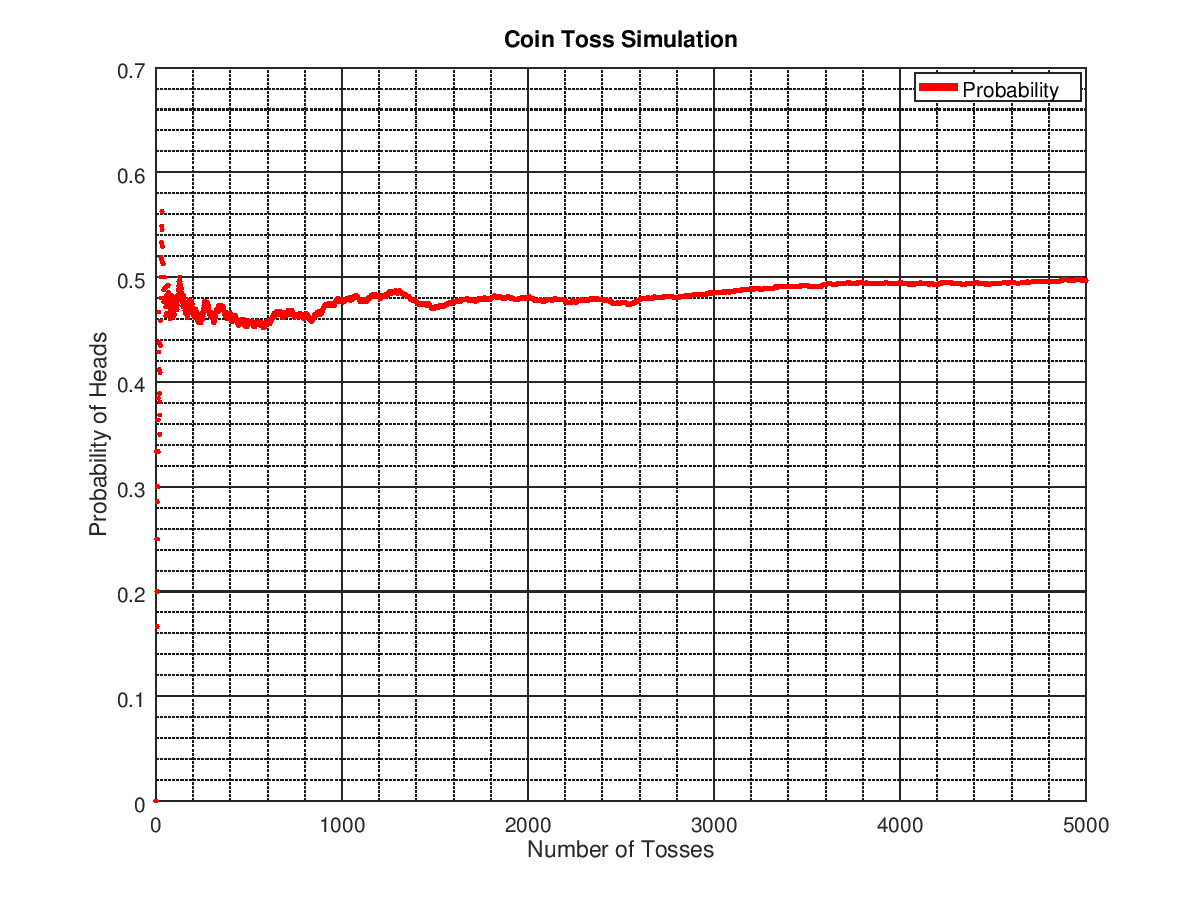


Figure 1: Probablity of heads in a coin toss simulation

Here a function was used to generate random numbers between 0 and 1 based on the normal distribution, and then was rounded to 0 for tails and 1 for heads. The number of heads vs the total number of tosses was plotted which gradually converged towards y=0.5 as the number of tosses increases. Eventually, at ∞, the probability converges to 0.5. At the end of the above simulation run, the probability was 0.49710.

**Question 2**: Bubble Sort Analysis

Implement two different versions of bubble sort for a randomized data sequence.

**Solution:** Alongside the classical bubble sort algorithm, a modified version was implemented where the loop terminates before the nth/n-1th iteration. As the array is often sorted before the final pass, this proves to reduce the number of iterations required significantly. However, it still remains an O(n2) algorithm as shown below in Figure 2. For sake of comparison, insertion sort has also been implemented.

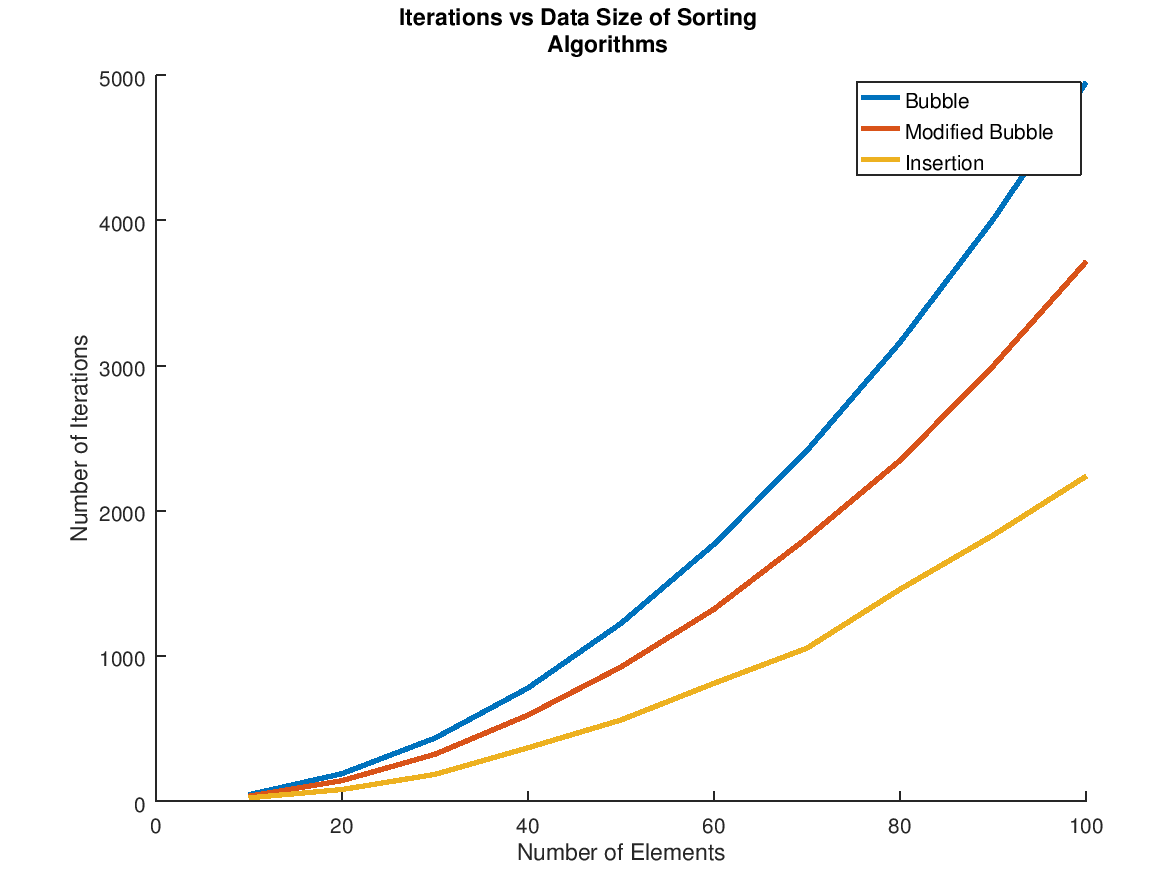


Figure 2: Asymptotic analysis of bubble and modified bubble sort

**Question 3**: Internal Sorting Analysis

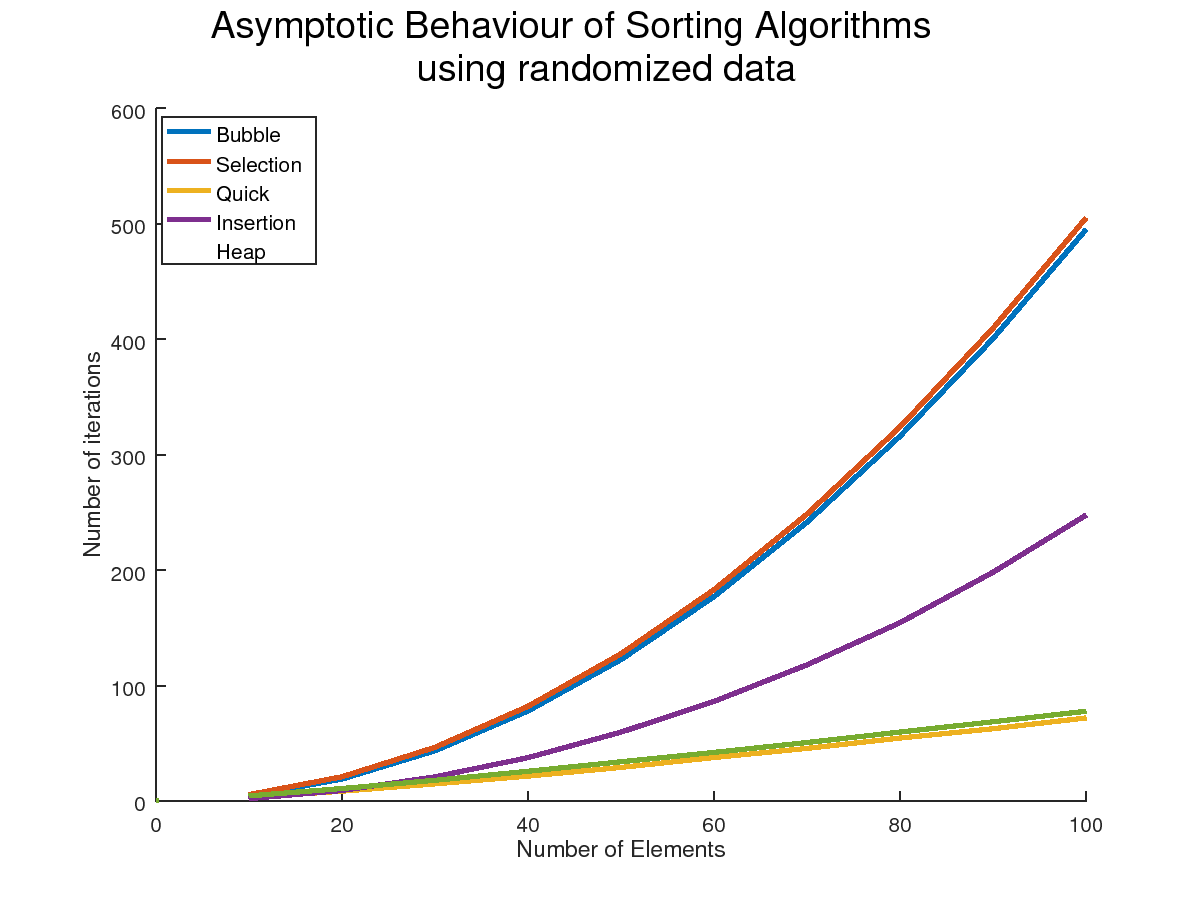
Determine the asymptotic behaviour of at least 5 internal sorting algorithms for each of the following input variations:

1. Random Data
2. Reverse Ordered Data
3. Almost Sorted Data
4. Highly Repetitive Data

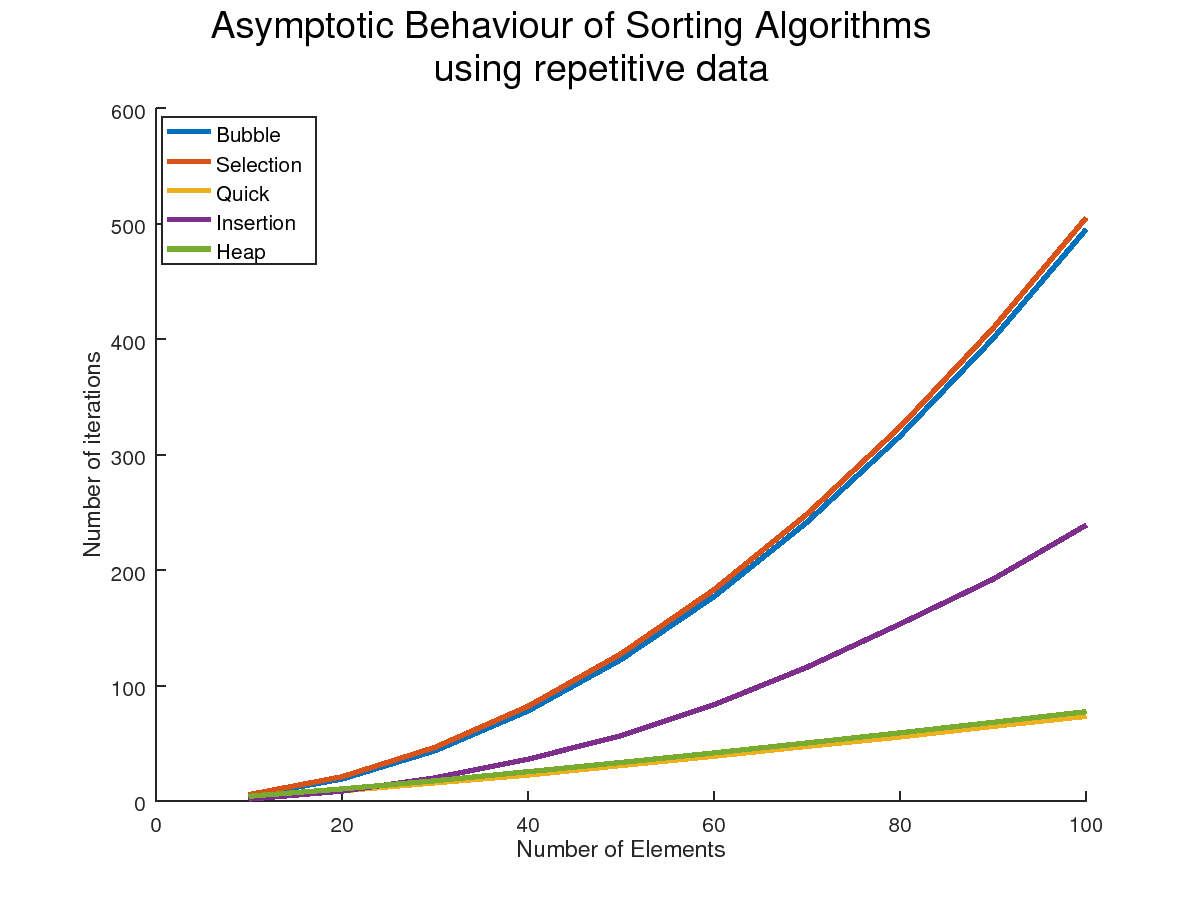
Select a suitable number of elements for the analysis that supports your program.

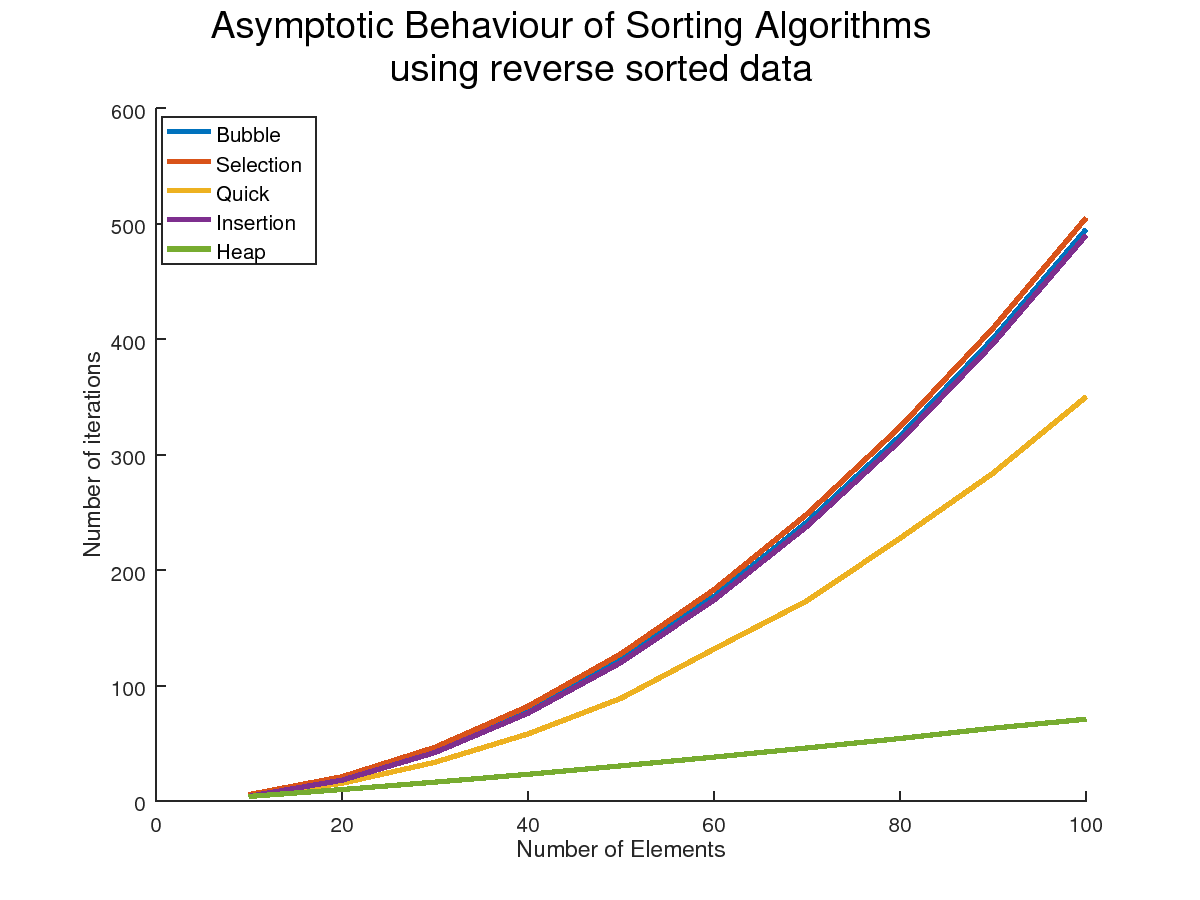
**Solution**: For analysis, bubble, selection, insertion, quick and heap sorting methods have been implemented.

1. Clearly, we observe that selection sort takes the most time, followed by bubble, insertion, quick and heap sorting methods.

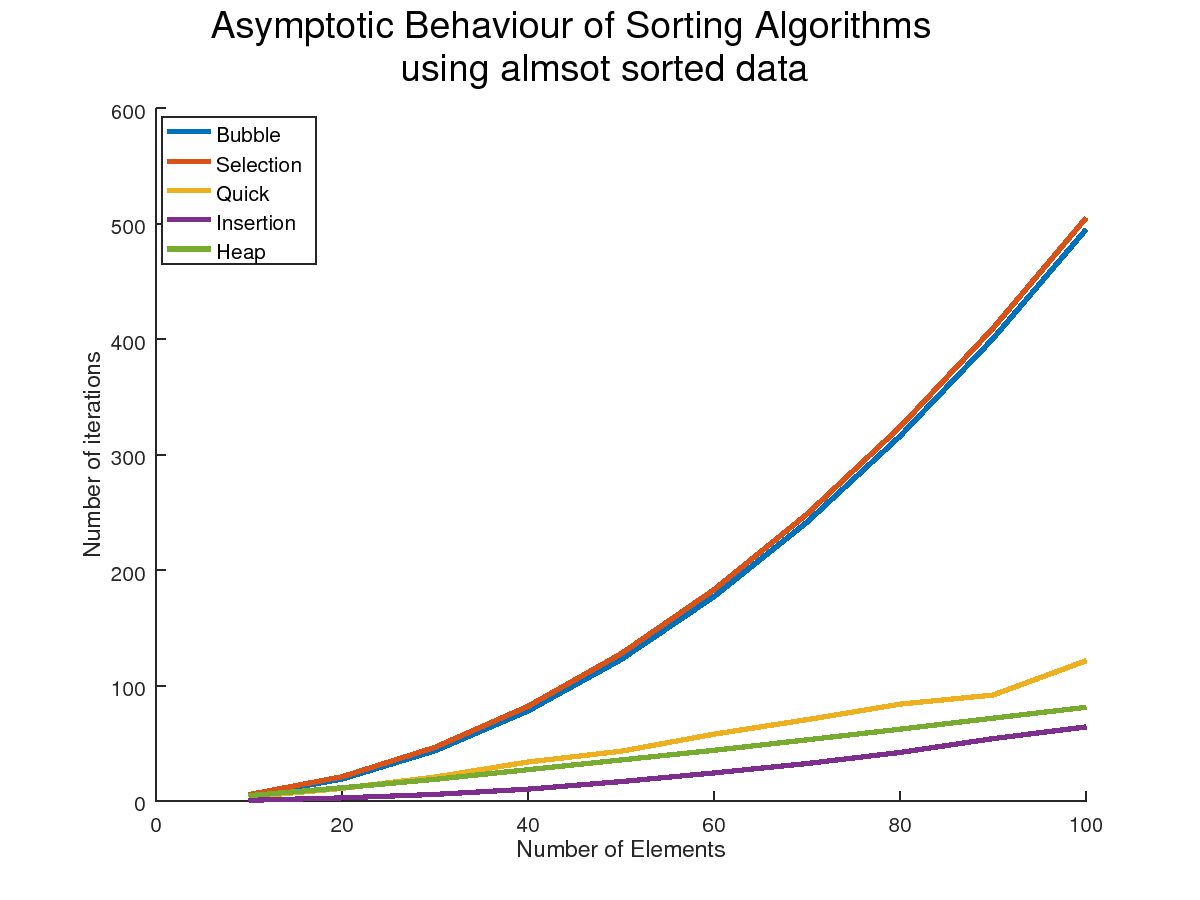


2. Repetitive data was generated by limiting the random number generation range to a reasonably small limit. The data was then fed through the sorting algorithms and they all produced nearly identical results to the previous question.



3. When reverse sorted data was used as input to the sorting algorithms, they all approached their worst-case complexity, which is evidenced by insertion sort being O(n2). Quick sort also increases significantly in its number of iterations, whereas heap sort retains its linear complexity.

4. For the fourth case, nearly sorted data was generated by sorting random sub-arrays of the original randomized data array, whose length is at least 70% of the original. Selection and bubble sorting methods retain their O(n2) complexity whereas insertion sort drops to a linear complexity. Quick sort also notably improves whereas heap sort remains the same



Conclusion: Clearly, the nature of the data provided as input significantly impacts the chosen algorithm’s time complexity and performance. Therefore, we can conclude that there is no “best” algorithm as such, there is only a “best” for each use case.

**Question 4**: Quick Sort Analysis

Compare the performance of variants of the quick sort algorithm for n=10 ... 1000. Use the results of Question 3 and accordingly modify your implemented quick sorting algorithm. Repeat the experiment for 50 iterations and record the same set of statistics and compare the results for the two different sorting techniques.