**Questions and Answers**

* Which Algorithm wins (the best one)?
  + For smaller datasets (n=100), the differences is not very significant, and algorithms like Insertion Sort, Selection Sort, and Bubble Sort perform well.
  + For larger datasets, Merge Sort and Quick Sort outperform the quadratic time complexity algorithms (Insertion, Selection, Bubble Sort).
  + Insertion, Selection, Bubble Sort: These algorithms exhibit slower performance for larger datasets, emphasizing their ineffisciency for scaling.
* Reflect on the Results
  + Insertion, Selection, Bubble Sort have slower performance for larger datasets.
  + Merge Sort, Quick Sort: Both algorithms have better scalability and efficiency for larger datasets. Quick Sort shows the best result.
  + The bigger the array gets, the better the lambda performs compared to the other sorting algorithms.
* Reflect on the benefits of avoiding mutation and using the delegate (Functions as first-class values).
  + Avoiding mutation and using functions as first-class values align with principles of functional programming. This creates code that is easier to understand and maintain. Delegate allows for flexibility and less repetitiveness when wanting to test each sorting algorithm. All this contribute to a stable code and easy to understand code.
* Which algorithm wins? Reflect on the Results.
  + Binary search seems to be the most effective when looked at all the categories.
  + Using Lambda, it is the same as Linear Search as it also uses linear search to find the data.

**Array (T[]):** Arrays have constant time complexity for accessing elements by index (O(1)), but linear time complexity for addition, search, and deletion operations.

**Dynamic Array (List)**: Dynamic arrays have constant time complexity for access by index (amortized O(1)), but linear time complexity for addition, search, and deletion.

**Stack and Queue:** Stacks and queues both have constant time complexity for addition and deletion operations but linear time complexity for search operations. Stack and Queue do not have access by index. Stack gets data based on LIFO (last in first out) manner and queue based on FIFO (first in first out) manner. The complexity here is O(1)

**Dictionary**: Dictionaries provide constant time complexity for addition, search, and deletion operations. The access is not by index but by key. The complexity here is O(1)

**SortedDict.**: Like a dictionary, but with logarithmic time complexity for addition, search, and deletion operations due to maintaining sorted order. The access is not by index but by key. The complexity here is O(1)

**HashSet**: HashSet provides constant time complexity for addition, search, and deletion operations. The access is not by index but by key. The complexity here is O(1).

