**Question-1:**

**Github:** <https://github.com/Kishan-Kumar-Zalavadia/ACA_Homework-1>

Discuss the behavior of the data structures along with which type of problems are most suitable for which data structures. Use the runtimes to support your analysis.

Detailed analysis using all the runtimes:

**1. Build Times (in nanoseconds):**

* **Binary Min Heap:** This data structure shows the quickest build times, with an average of around 1,352,175 nanoseconds. It is highly efficient for constructing a heap structure.
* **AVL Tree:** Building an AVL Tree takes significantly more time, averaging around 13,804,841,600 nanoseconds. This is due to the need for balancing during construction, which is a more complex operation.
* **Splay Tree:** Splay Tree construction also requires a substantial amount of time, averaging around 15,794,483,200 nanoseconds. This is because of the rotations needed to maintain the self-adjusting property of Splay Trees.

**2. Search Times for Lowest Numbers (in nanoseconds):**

* **Binary Min Heap:** Searching for the lowest numbers in a Binary Min Heap is very efficient, averaging around 2,933.2 nanoseconds. This is expected as heaps excel at finding the minimum element.
* **AVL Tree:** Searching for the lowest numbers in an AVL Tree is slightly slower, averaging around 2,391.6 nanoseconds. While this is higher than the heap, it's still reasonable for many applications.
* **Splay Tree:** Splay Trees exhibit competitive performance with an average search time of 1,450.0 nanoseconds. This indicates their adaptability and efficiency in finding frequently accessed elements.

**3. Search Times for Random Numbers (in nanoseconds):**

* **Binary Min Heap:** Searching for random numbers in a Binary Min Heap has significantly higher search times, averaging around 4,044,658.2 nanoseconds. This is expected as heaps are optimized for accessing the minimum element, not arbitrary elements.
* **AVL Tree:** Searching for random numbers in an AVL Tree is better than the heap but still shows higher search times, averaging around 22,183.4 nanoseconds. AVL Trees are more versatile but not as optimized for minimum access.
* **Splay Tree:** Splay Trees exhibit the lowest search times for random numbers, averaging around 5,191.4 nanoseconds. This suggests that Splay Trees quickly adapt to varying access patterns.

**4. Insertion Times (in nanoseconds):**

* **Binary Min Heap:** Insertions into a Binary Min Heap are highly efficient, averaging around 100,175.0 nanoseconds. Heaps are well-suited for inserting elements while maintaining the heap property.
* **AVL Tree:** Insertions into an AVL Tree take more time, averaging around 789,375.0 nanoseconds. This is due to the need for balancing the tree during insertion.
* **Splay Tree:** Splay Trees show competitive insertion times, averaging around 1,058,366.8 nanoseconds. Their self-adjusting behavior comes with a moderate cost in terms of insertion time.

**5. Standard Deviation (in nanoseconds):**

* Binary Min Heap tends to have smaller standard deviations, indicating more consistent and predictable behavior in its operations.
* AVL Tree and Splay Tree exhibit higher standard deviations, indicating more variability in their performance, especially noticeable in search times for random numbers.

In conclusion, the runtimes, expressed in nanoseconds, confirm the initial analysis:

* Binary Min Heap excels in build times, search times for the lowest numbers, and insertion times, making it suitable for applications where quick access to the minimum element is essential.
* AVL Trees offer a balanced performance across various operations, making them suitable when self-balancing and versatility are needed.
* Splay Trees are strong in adaptability, providing competitive search times for random numbers, making them ideal for dynamic and unpredictable access patterns.

The choice of data structure should align with the specific needs and access patterns of your application, as reflected in the runtimes.

In summary:

* For building data structures, Binary Min Heap has the minimum time.
* For searching for the lowest numbers, Splay Tree has the minimum time.
* For searching for random numbers, Splay Tree also has the minimum time.
* For inserting elements, Binary Min Heap has the minimum time.

**Reference:**

* Binary Min Heap: <https://www.geeksforgeeks.org/min-heap-in-java/> , chatGPT
* AVL Tree: <https://www.javatpoint.com/avl-tree-program-in-java>, <https://www.geeksforgeeks.org/find-the-minimum-element-in-a-binary-search-tree/>
* Splay Tree: <https://www.javatpoint.com/splay-tree> , <https://codeofcode.org/lessons/splay-trees/>, <https://www.geeksforgeeks.org/deletion-in-an-avl-tree/>
* Random Functions: chatGPT