1. Random Word Generation Scores

Word	Score	Key Observations
AKMLFGSSGQA	31	The highest valid score recorded was 31.
(Other words with invalid combinations)	-1	Invalid combinations or non-recognized words received a score of -1.
(Shorter words)	Lower Scores	Shorter words generally scored lower.
(Longer words)	Mixed Results	Longer words had mixed results, some scoring high while others were invalid.
		The scoring system is based on letter values but does not validate words against a dictionary.

2. Sorting Performance: Bubble Sort vs. Merge Sort

Number of Words	Bubble Sort Time (ns)	Merge Sort Time (ns)	Key Observations
1000	442,846,100	6,741,800	Bubble Sort took approximately 66 times longer than Merge Sort.
5000	9,874,333,600	48,580,600	The time difference between Bubble Sort and Merge Sort increased significantly as the number of words grew.
10,000	38,965,837,800	94,793,900	For 10,000 words, Bubble Sort took about 411 times longer than Merge Sort.
			Sorting results were the same for both algorithms, showing correct implementation but highlighting Merge Sort's superior efficiency.

3. Searching Performance: Sequential Search vs. Binary Search (Searching for an Existing Word)

Number of Words	Sequential Search Time (ns)	Binary Search Time (ns)	Key Observations
1000	52,100	51,100	Binary Search was slightly faster than Sequential Search.
5000	384,500	15,700	Binary Search was approximately 24 times faster than Sequential Search.
10,000	507,800	18,000	Binary Search was around 28 times faster for 10,000 words.
			The time difference was significant for larger datasets, showing Binary Search's superior efficiency for sorted lists.

4. Searching Performance: Sequential Search vs. Binary Search (Searching for a Non-Existent Word)

Number of Words	Sequential Search Time (ns)	Binary Search Time (ns)	Key Observations
10,000	960,400	24,100	Binary Search was around 40 times faster than Sequential Search when searching for a non-existent word.

5. Java's `Arrays.sort` and `Arrays.binarySearch` Performance

Sorting Times (ns)

Number of Words	Arrays.sort Time (ns)	Key Observations
1000	6,361,200	`Arrays.sort` is highly efficient for smaller datasets.
5000	45,518,400	The sorting time increased as the number of words grew, but performance remained strong.
10,000	97,625,600	Even with 10,000 words, `Arrays.sort` performed efficiently.

Searching Times (ns)

Number of Words	Word Search Time (ns)	Non-Existent Word Search Time (ns)	Key Observations
1000	62,800	51,800	Searching for an existing word was quick, but non-existent word search took slightly longer.
5000	24,600	18,200	Java's `Arrays.binarySearch` was faster than custom methods in both cases.
10,000	29,400	20,100	Searching performance remained efficient for both existing and non-existent words.