Trie Benchmark Evaluation

February 2, 2025

Evaluation

All benchmarks are run on:

```
Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
RAM  (DDR4):              7.6Gi
Swap (NVMe PCIe SSD): 63.0Gi
```

Run command:

```
taskset -c $CORE sudo chrt -f 99 ./benchmark
```

Variation: Vector Trie

```
struct Node
{
  bool is_end = false;
  std::vector<std::pair<unsigned char,
                        std::unique_ptr<Node>>> children
};
```

Variation: Array Trie

```
struct Node
{
  std::size_t is_end = false;
  std::unique_ptr<Node> children[63];
  Node() { static_assert(sizeof(Node) == 64 * 8); }
};
```

Variation: Hash Trie

```
struct Node
{
  bool is_end = false;
  std::unordered_map<unsigned char,
                     std::unique_ptr<Node>> children;
};
```

Benchmark Parameters

- num_words
- min_word_length
- max_word_length
- num_insert_queries
- num_contains_queries
- num_remove_queries
- chance_random_query

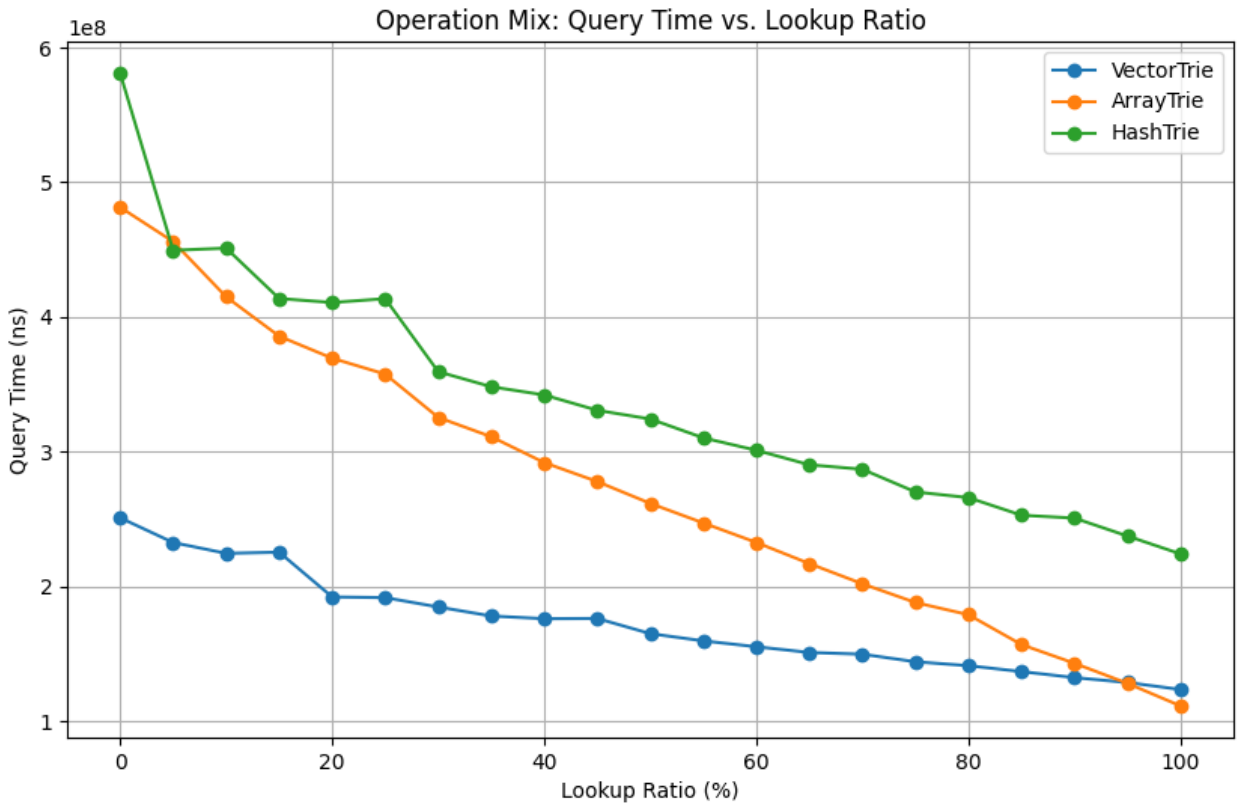
Operations

Non-Lookup:

* Insert
* Remove

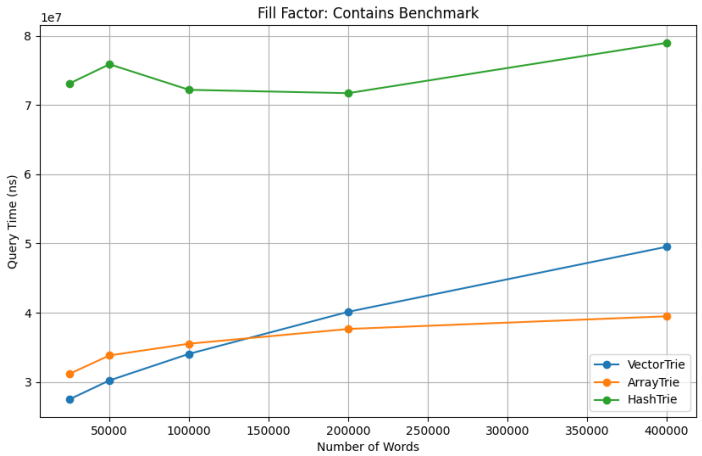
Lookup:

* Contains

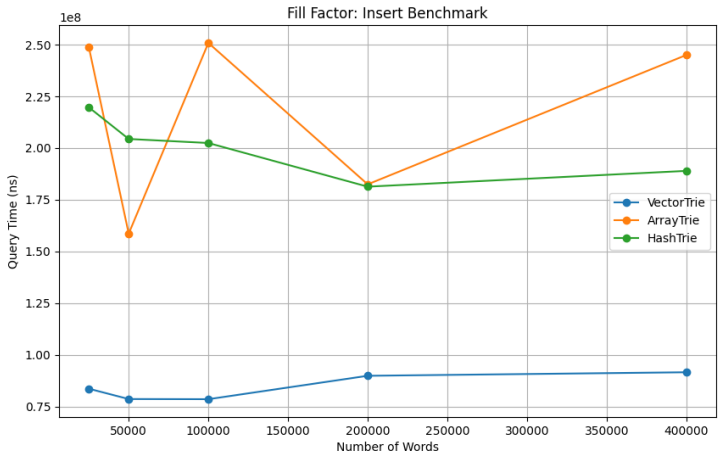


Fill Factor

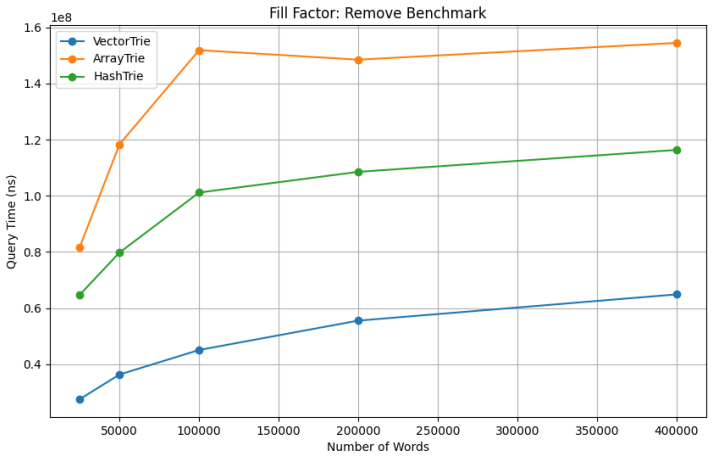
Contains



Insert

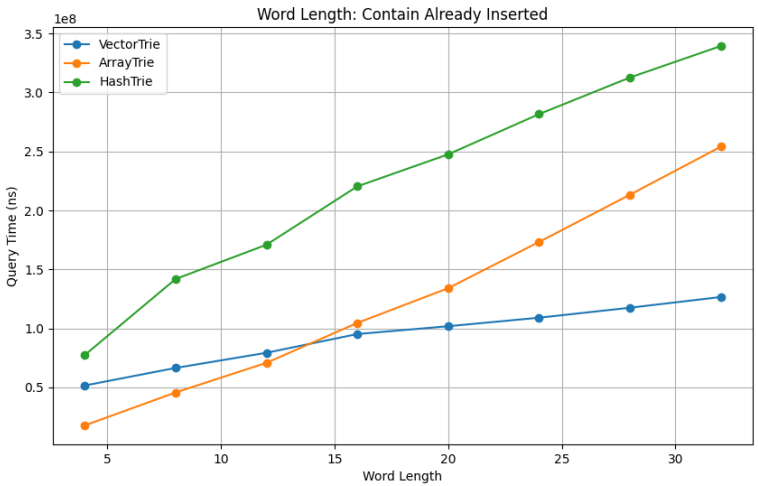


Remove

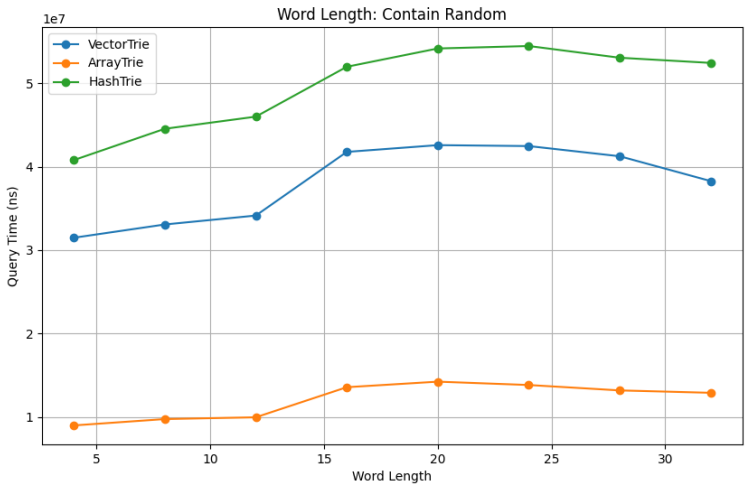


Word Length (Contains)

Already Inserted

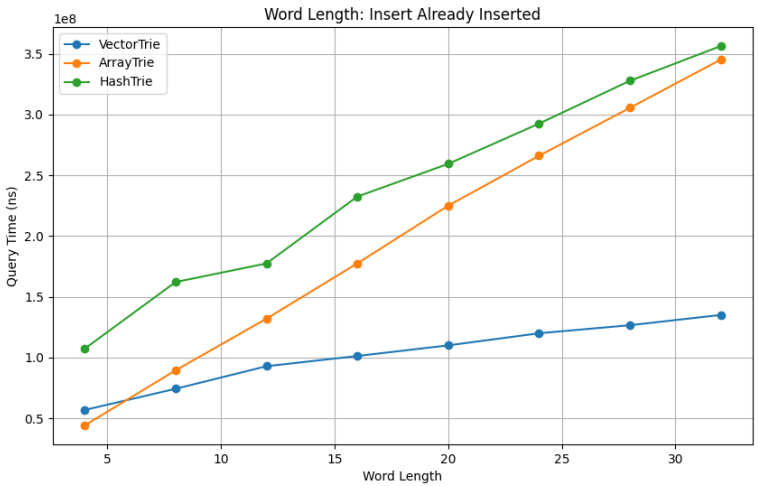


Random String

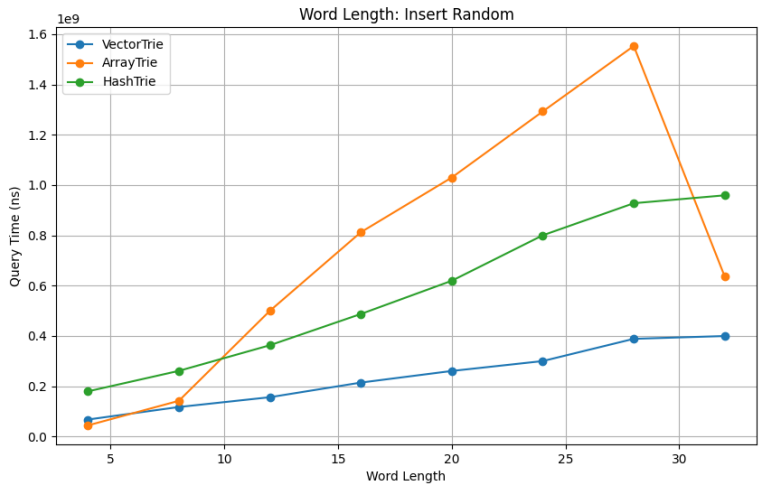


Word Length (Insert)

Already Inserted

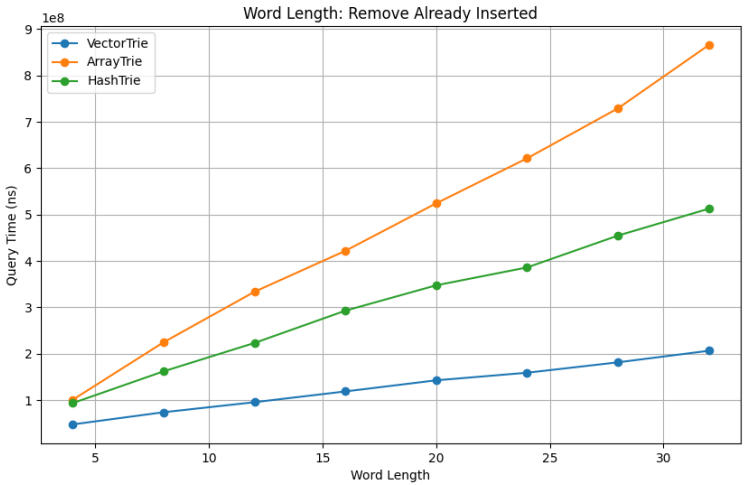


Random String

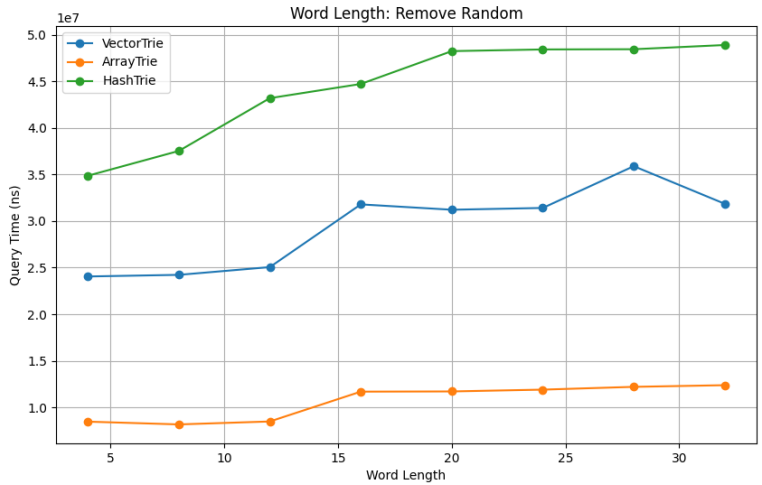


Word Length (Remove)

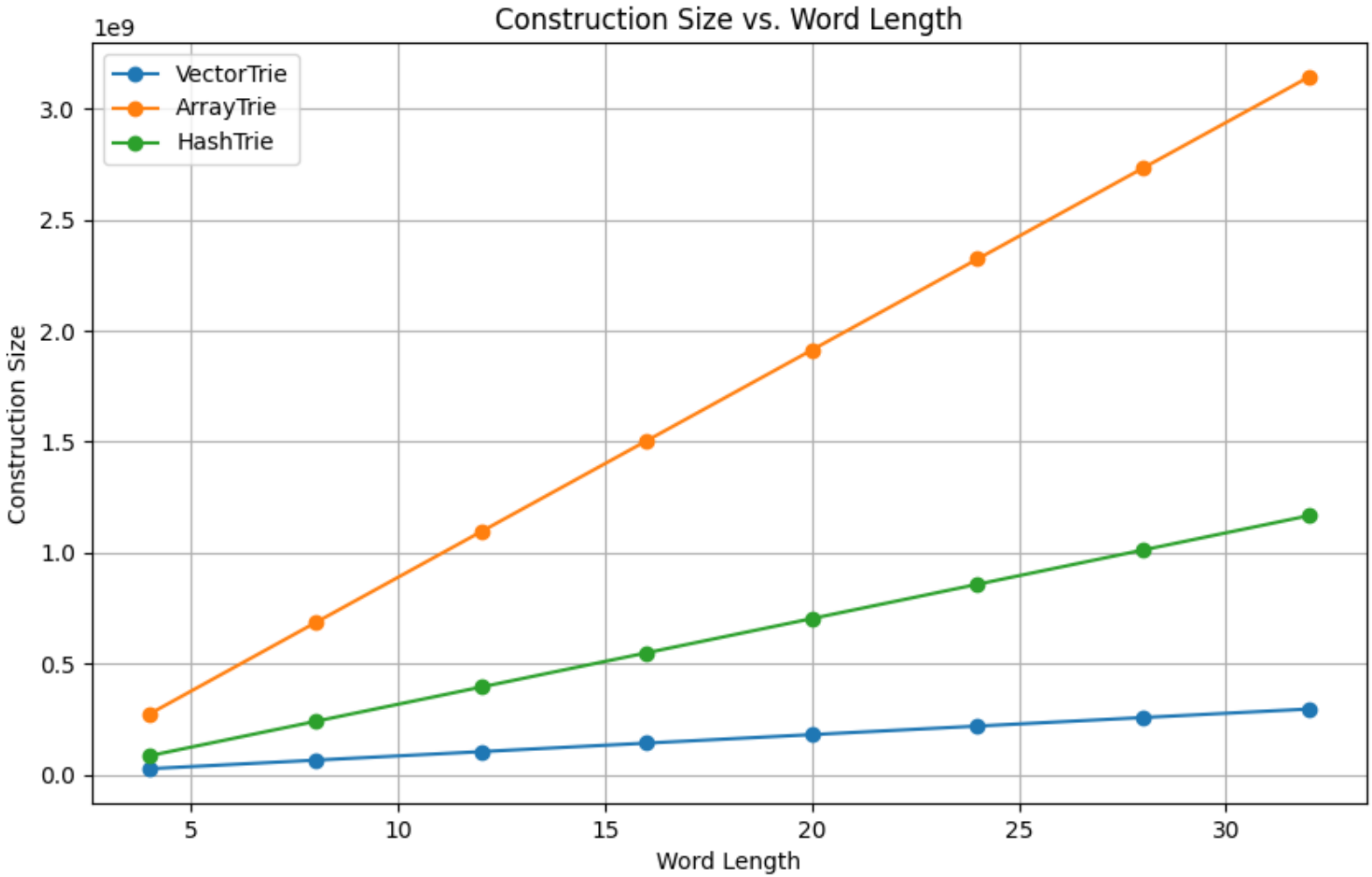
Already Inserted



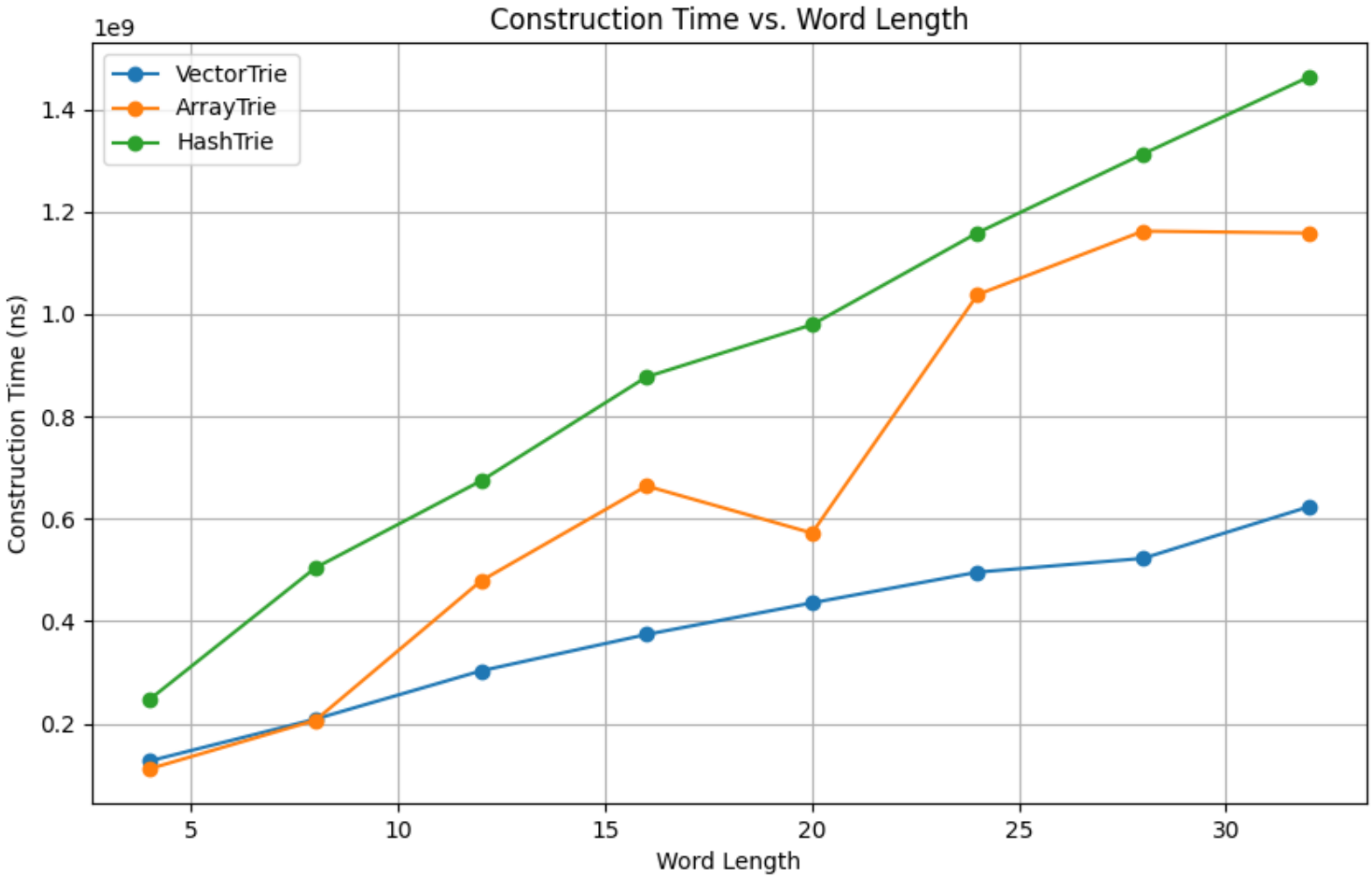
Random String



Construction: Size



Construction: Time



Conclusion

- **Hash Trie:** Not performing well. While using a hash map in a trie isn't always bad, this implementation is simply not smart enough.
- **Array Trie:** Very fast on lookup operations but suffers from significant size overhead and long construction time. A smarter allocation strategy (e.g., bulk allocation) might help.
- **Vector Trie:** The overall winner in this comparison, yielding strong performance in every scenario. It is the go-to solution when there is no special use-case.