Randomized Quicksort Analysis

Theoretical Analysis

Average-Case Time Complexity

The Randomized Quicksort algorithm is known for its efficient average-case performance. The time complexity of Randomized Quicksort is O(nlogn), where n is the number of elements in the array.

- **Recurrence Relation**: The average-case recurrence relation for Quicksort, assuming the pivot divides the array into two nearly equal parts, is:

  $T(n)=T(n / 2)+O(n)$

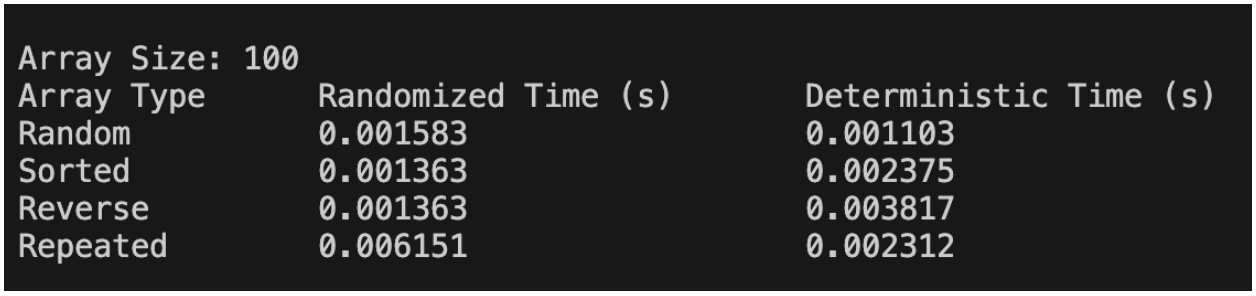
  This simplifies to O(nlogn) using the Master Theorem.
- **Random Pivot Selection**: We expect the array to be divided into reasonably balanced partitions on average by choosing a random pivot. This reduces the likelihood of encountering worst-case scenarios.
- **Indicator Random Variables**: The probability of a pivot dividing the array into two roughly equal halves is high, as any element is equally likely to be selected in the indicator random variable. This randomness helps ensure that the expected recursion depth remains O(logn).

Empirical Comparison

Randomized Quicksort was compared with the Deterministic Quicksort algorithm, in which the first element was selected as the pivot. The timeit library was used to compare the time these two algorithms took to execute the various test scenarios.

The following test cases were used for this comparison with input array sizes of 100, 1000, and 5000.

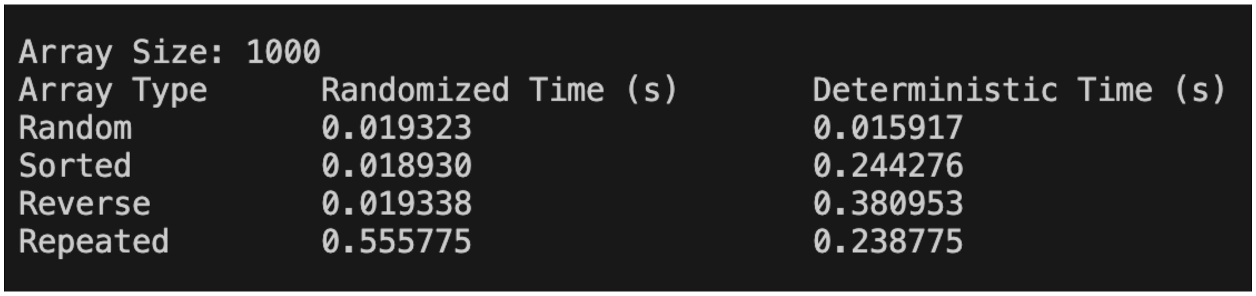
Array size: 100



```
Array Size: 100
Array Type      Randomized Time (s)       Deterministic Time (s)
Random          0.001583                  0.001103
Sorted          0.001363                  0.002375
Reverse         0.001363                  0.003817
Repeated        0.006151                  0.002312
```

For smaller arrays, both techniques work well. Randomized Quicksort performs somewhat slower on repeated elements due to inefficient partitioning. At the same time, Deterministic Quicksort becomes inefficient with sorted and reverse-sorted arrays because the first element is a poor pivot in these scenarios.

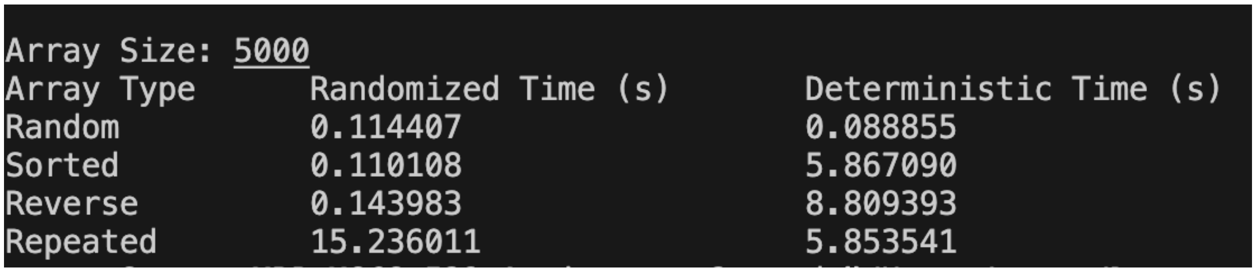
Array size: 1000



```
Array Size: 1000
Array Type      Randomized Time (s)       Deterministic Time (s)
Random          0.019323                  0.015917
Sorted          0.018930                  0.244276
Reverse         0.019338                  0.380953
Repeated        0.555775                  0.238775
```

Randomized Quicksort performs reliably on random, sorted, and reverse-sorted arrays but struggles with repeated elements due to inefficient partitioning. Deterministic Quicksort becomes much slower on sorted and reverse-sorted arrays, as expected because the first element is a poor pivot, resulting in unbalanced partitions and additional recursive operations.

Array size: 5000



```
Array Size: 5000
Array Type      Randomized Time (s)       Deterministic Time (s)
Random          0.114407                  0.088855
Sorted          0.110108                  5.867090
Reverse         0.143983                  8.809393
Repeated        15.236011                 5.853541
```

For large arrays, Randomized Quicksort performs consistently for random, sorted, and reverse-sorted arrays but significantly slows down with repeated elements. The random pivot selection algorithm does not handle repeated items well, resulting in several recursive calls and deep recursion. Deterministic Quicksort is exceedingly inefficient for sorted and reverse-sorted arrays. It performs poorly due to continually poor pivot selection (the first element), which results in very unbalanced partitions.

Discussion of Result

Randomized Quicksort is more robust across different input distributions, delivering consistent results except for repeated elements. Deterministic Quicksort performs poorly on sorted and reverse-sorted arrays due to fixed pivot selection, especially as input size grows. However, it outperforms Randomized Quicksort on random and repetitive elements.