

Randomized QuickSelect for Large Dataset Analytics

Pushkar Desai 2024eb02433

Introduction

Efficient analytics require quickly finding statistics like the median or percentiles in unsorted datasets. Randomized QuickSelect achieves expected $O(n)$ time, avoiding the unnecessary $O(n \log n)$ work of full sorting.

Problem Statement

Given an unsorted array of transaction amounts, compute the k -th smallest value.

Example:

transactions = [7, 10, 4, 3, 20, 15], $k = 3 \rightarrow$ Output: 7

Algorithm Design

Steps:

- Randomly select a pivot.
- Partition around the pivot.
- Recurse only into the relevant partition.

Pseudocode

```
FUNCTION QuickSelect(arr, low, high, target_index):
IF low == high:
RETURN arr[low]

pivot_index = RandomizedPartition(arr, low, high)

IF pivot_index == target_index:
RETURN arr[pivot_index]
ELSE IF pivot_index > target_index:
RETURN QuickSelect(arr, low, pivot_index - 1, target_index)
ELSE:
RETURN QuickSelect(arr, pivot_index + 1, high, target_index)

FUNCTION RandomizedPartition(arr, low, high):
random_pivot_idx = RandomInteger(low, high)
Swap(arr[random_pivot_idx], arr[high])
RETURN Partition(arr, low, high)
```

```

FUNCTION Partition(arr, low, high):
pivot_value = arr[high]
i = low
FOR j FROM low TO high-1:
IF arr[j] <= pivot_value:
Swap(arr[i], arr[j])
i++
Swap(arr[i], arr[high])
RETURN i

```

Justification for Randomization

Random pivots prevent consistently poor splits. Deterministic pivot rules fail on sorted or adversarial data.

Expected partitions remain balanced, giving expected $O(n)$ time.

Complexity Analysis

Expected case: $T(n) = T(n/2) + O(n) \Rightarrow O(n)$

Worst case: $T(n) = T(n-1) + O(n) \Rightarrow O(n^2)$

Sorting approaches always require $O(n \log n)$.

Validation Results

Tested on random, sorted, and reverse-sorted datasets:

- Random: $O(n)$
- Sorted: deterministic fails, randomized remains $O(n)$
- Reverse-sorted: randomized avoids $O(n^2)$ worst case.