Merge Sort

By:

Dustin Smith

Quick sort is another in place sorting algorithm. An efficient quick sort implementation is somewhat faster than merge sort and two to three times faster than heap sort. Quick sort follows the divide and conquer paradigm.

1. If the range has less than two elements, return.
2. If not, pick a pivot value–depends on the partitioning routine; could be random.
3. Partition the range; reorder while determining a point of division so that all elements with values less than the pivot come before the division and greater after.
4. Recursively apply.

Two common partitioning schemes are Lomuto and Hoare. However, Lomuto performs poorly on arrays with many repeated values giving $\theta(n^2)$ complexity on an array of all repeats. By the Master Theorem, we have complexity of $\theta(n \log_2 n)$ with worst case at $\theta(n^2)$. The Lomuto partitioning scheme is frequently used due to being easier to understand. However, the Hoare partitioning scheme is more efficient than Lomuto since it does three times fewer swaps.

```python
# Implementation from The Algorithms Python
def quick_sort(arr: List[int]) -> List[int]:
  if len(arr) < 2:
    return arr

  pivot = arr.pop()
  higher = []
  lower = []

  for i in arr:
    (higher if i > pivot else lower).append(i)

  return quick_sort(lower) + [pivot] + quick_sort(higher)


# Lomuto
def lomuto_partition(arr: List[int], low_idx: int, high_idx: int) -> int:
  pivot = arr[high_idx]
  i = low_idx - 1

  for j in range(low_idx, high_idx):
    if arr[j] <= pivot:
      i += 1
      arr[i], arr[j] = arr[j], arr[i]
  arr[i + 1], arr[high] = arr[high], arr[i + 1]
  return i + 1


def quick_sort_lomuto(arr: List[int], low: int, high: int) -> None:
  if low < high:
    part_idx = lomuto_partition(arr, low, high)
```

```
    quick_sort_lomuto(arr, low, part_idx - 1)
    quick_sort_lomuto(arr, part_idx + 1, high)


# Hoare
def hoare_partition(arr: List[int], low_idx: int, high_idx: int) -> int:
  pivot = arr[low_idx]
  i = low_idx - 1
  j = high_idx + 1

  while True:
    i += 1

    while arr[i] < pivot:
      i += 1

    j -= 1

    while arr[j] > pivot:
      j -= 1

    if i >= j:
      return j

    arr[i], arr[j] = arr[j], arr[i]


def quick_sort_hoare(arr: List[int], low: int, high: int) -> None:
  if low < high:
    part_idx = hoare_partition(arr, low, high)

    quick_sort_hoare(arr, low, part_idx)
    quick_sort_hoare(arr, part_idx + 1, high)
```