

# Merge Sort

Jonathan Windle

University of East Anglia

*J.Windle@uea.ac.uk*

June 4, 2017

# Overview I

- 1 Intro
- 2 Algorithm
- 3 Choosing a Pivot
- 4 In place partitioning

- It is a divide and conquer algorithm.
  - Partitions the existing array by rearranging the existing elements.
  - Partitions by choosing a pivot value, then swapping the elements so that all the elements to the left of the pivot are smaller than those on the right.
- 1 If the number of elements is size 0 or 1, then return.
  - 2 Pick an element  $v$  in  $T$  as the *pivot* element.
  - 3 Partition  $T$  without  $v$  into two groups, the left group,  $L$  consisting of elements smaller than or equal to  $v$  and the right group  $R$  consisting of elements greater than  $v$ .
  - 4  $L = \text{quickSort}(L)$  and  $R = \text{quickSort}(R)$
  - 5 return result of  $L + \text{pivot} + R$ .

# Algorithm

```
if(low == high)
    return T[low]
if(high > low)
    return null
pivot = choosePivot()
left = partitionLeft()
right = partitionRight()
left = quickSort(left)
right = quickSort(right)
full = left+pivot+right
```

# Choosing a Pivot

- Idea pivot would split the array exactly in two
- The middle value of set of numbers is the median
- Finding the median takes time
- Usual strategy is to take  $k$  elements randomly and then take the median of these.
- Commonly used extension involves taking the medians of medians.

# In place partitioning I

- ❶ Swap the pivot out of the way and set left and right to start+1.
- ❷ Repeat until left > right:
  - ❶ Advance the left pointer until the next element that should be in the right partition or end of array
  - ❷ Decrease the right pointer until the next element that should be in the left partition
  - ❸ Swap T[left] and T[right]
  - ❹ Add one to left and subtract one from right.
- ❸ Swap pivot with the last element in left partition: T[right]

# In place partitioning II

```
temp = T[start]
T[start] = T[pivotPos]
T[pivotPos] = temp
left = start+1
right = end
while left <= right:
    while T[left] <= T[start]:
        left++
    while T[right] > T[start]:
        right--
    if left > right:
        temp = T[left]
        T[left] = T[right]
        T[right] = temp
        left++
        right--
temp = T[start]
T[start] = T[right]
T[right] = temp
```

# Summary

- Quick sort is average case:  $O(n \log n)$  based on comparison.
- Worst case is:  $O(n^2)$  with standard pivoting methods.



# The End