# Quicksort

Dr. Yingwu Zhu

# Quicksort

- A more efficient exchange sorting scheme than bubble sort
  - A typical exchange involves elements that are far apart
  - *Fewer interchanges are required to correctly position an element.*
- Quicksort uses a **divide-and-conquer** strategy
  - A recursive approach
  - The original problem partitioned into simpler sub-problems,
  - Each sub problem considered (conquered) independently.
- Subdivision continues until sub problems obtained are simple enough to be solved directly
  - How simple?

# Quicksort: Divide/Split

- Choose some element called a **pivot**
- *Perform a sequence of exchanges* so that
  - All elements that are less than this pivot are to its left.
  - All elements that are greater than the pivot are to its right.
- Divides the (sub)list into two smaller sub lists,
- Each of which may then be sorted independently in the *same* way.

# Quicksort

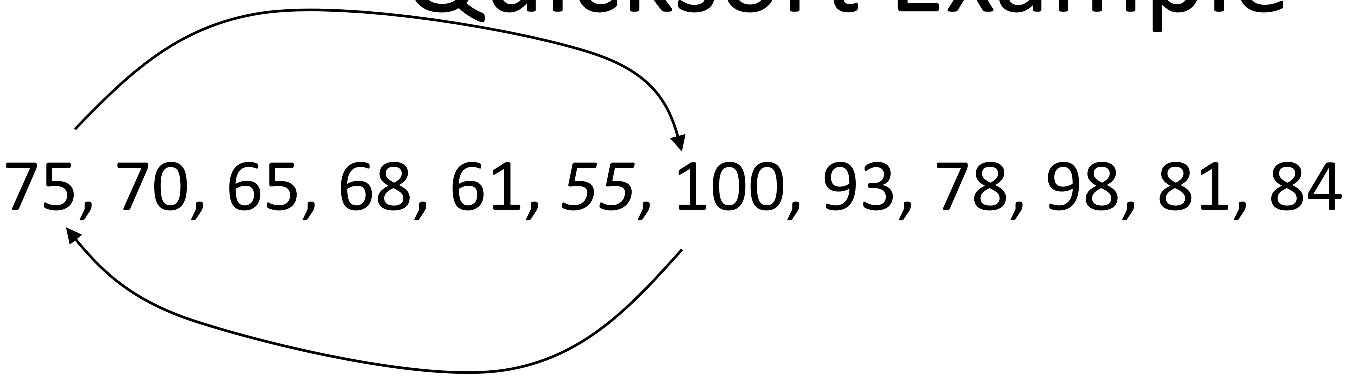
```
If the list has 0 or 1 elements,
                    return. // the list is sorted, simple enough!
Else do:
        Pick an element in the list to use as the pivot.
Split   Split the remaining elements into two disjoint groups:
            SmallerThanPivot = {all elements <= pivot}
            LargerThanPivot = {all elements > pivot}

 Return the list rearranged as:
    Quicksort(SmallerThanPivot),
    pivot,
    Quicksort(LargerThanPivot).
```

# Quicksort Example

- Given to sort:
  75, 70, 65, 84 , 98, 78, 100, 93, 55, 61, 81, 68
- Select, arbitrarily, the first element, 75, as pivot.
- Search from right for elements <= 75, stop at first element <=75
- Search from left for elements > 75, stop at first element >75
- Swap these two elements, and then repeat this process

# Quicksort Example



75, 70, 65, 68, 61, *55*, 100, 93, 78, 98, 81, 84

- When done, swap with pivot
- This SPLIT operation placed pivot 75 so that all elements to the left were <= 75 and all elements to the right were > 75.
- 75 is now placed appropriately
- Need to sort sublists on either side of 75

# Quicksort Example

- Need to sort (independently):

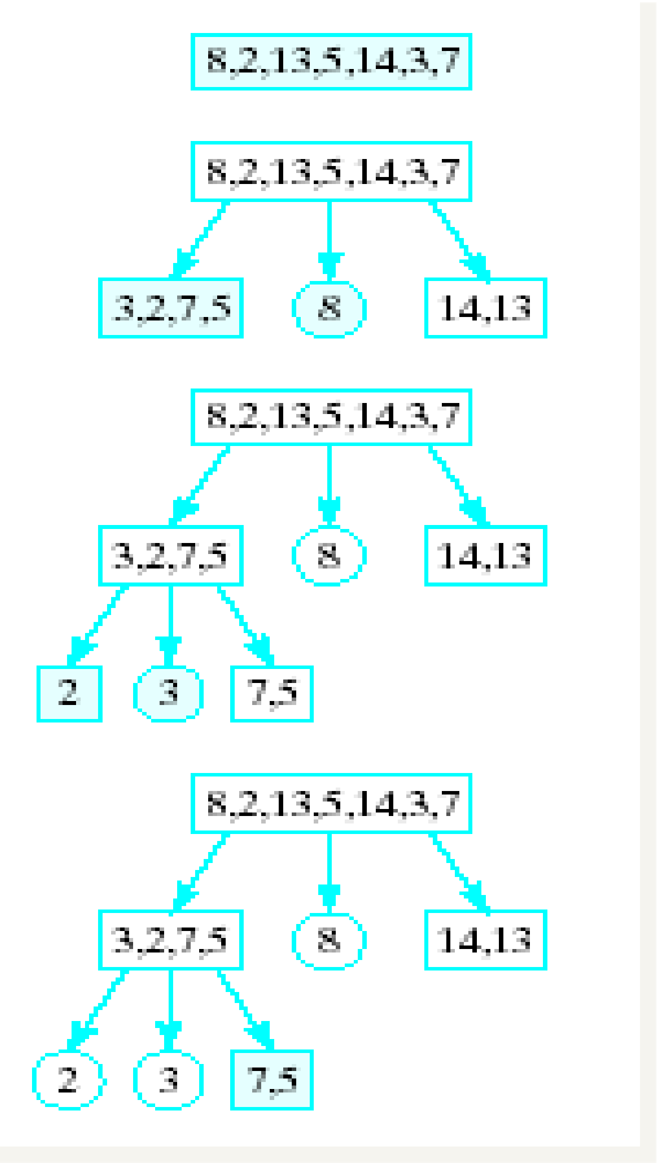
  55, 70, 65, 68, 61 and

  100, 93, 78, 98, 81, 84

- Let pivot be 55, look from each end for values larger/smaller than 55, swap
- Same for 2nd list, pivot is 100
- Sort the resulting sublists in the same manner until sublist is trivial (size 0 or 1)

# Quicksort

- Note visual example of a quicksort on an array



etc. …

# Reflection of Quicksort

- Perform spilt() operation on a (sub)list, such that: left-sublist, pivot, right-sublist
- Recursively and independently perform split() on left-sublist and right-sublist, until their sizes become 0 or 1 (simple enought).
- So, the basic operation is split!
  - int split(int x[], int low, int high)
  - [low, high] specifies the sublist.
  - Returns the final position of the pivot

# Implementing Quicksort

- Basic operation: split
  - Choose the pivot (e.g., the first element)
  - Scan the (sub)list from both ends, swap elements such that the resulting left sublist < pivot and right sublist >= pivot
  - int split (int x[], int first, int last)

# Recursive Quicksort

- void quicksort(int x[], int n)

# Quicksort: T(n)

- Best-case ?
- Worst-case ?

# Quicksort Performance

- $O(n\log_2 n)$ is the best case computing time
  - If the pivot results in sublists of approximately the same size.

- $O(n^2)$ worst-case
  - List already ordered, elements in reverse
  - When `Split()` repetitively results, for example, in one empty sublist

# Improvements to Quicksort

- An *arbitrary pivot* gives a poor partition for nearly sorted lists (or lists in reverse)
- Virtually all the elements go into either `SmallerThanPivot` or `LargerThanPivot`
  - all through the recursive calls.
- Quicksort takes quadratic time to do essentially nothing at all.

# Improvements to Quicksort

- Better method for selecting the pivot is the *median-of-three rule*,
  - Select the median of the first, middle, and last elements in each sublist as the pivot.

- Often the list to be sorted is already partially ordered
- Median-of-three rule will select a pivot closer to the middle of the sublist than will the “first-element” rule.

# Improvements to Quicksort

- For small files ($n$ <= 20), quicksort is worse than insertion sort;
  - *small files occur often because of recursion.*

- Use an efficient sort (e.g., insertion sort) for small files.
- Better yet, use `Quicksort()` until sublists are of a small size and then apply an efficient sort like insertion sort.

# Non-recursive Quicksort

- Think about non-recursive alg.?