

Quick Sort

- Undoubtedly the most popular sorting algorithm.
- $O(n \log(n))$.
 - Arrays
 - faster than mergesort
 - Does not require as much space as merge sort.
 - Recall that merge sort requires twice as much space as original array to store temporary sub arrays.
 - Linked Lists
 - Merge sort better – cannot play the games that make quicksort fast on linked lists

Partitioning

- Divide data into two groups, such that all items with a key value higher than a specified amount will be in one group, and the ones with lower key value in the other.
- Important: the items in each group are NOT sorted.
- The threshold value used to determine which group an item belongs is called the “pivot value”.

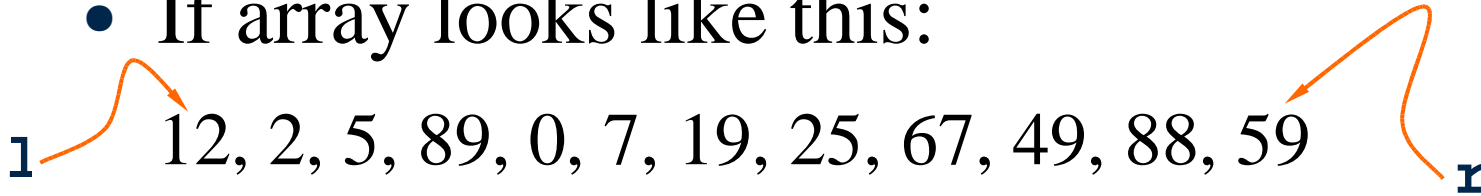
Partitioning Algorithm

- Starts with two indices, \mathbf{l} and \mathbf{r} , each will be moving towards each other.
- Advance the left index through the array until the first element larger than the pivot is encountered.
- Similarly advance the right index until the first element smaller than pivot is found.
- Swap the elements pointed to by left and right indices – thus they end up on the right side.
- Stop when two indices meet or cross each other.

Example

- If array looks like this:

1 12, 2, 5, 89, 0, 7, 19, 25, 67, 49, 88, 59 **R**

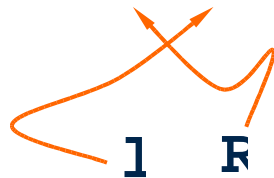


- Partition with pivot value 50:

12, 2, 5, 89, 0, 7, 19, 25, 67, 49, 88, 59

- First swap (89, 49)

12, 2, 5, 49, 0, 7, 19, 25, 67, 89, 88, 59



Partitioning

```
mid = (l+r)/2;
pivot = theArray[mid];
while( l <= hi ) {
    while((l<hi0) && (theArray[l]<pivot))
        l++;
    while((r>lo0)&& (theArray[r]>pivot))
        r--;
    if( l <= r ) {
        swap(l, r);
        l++;
        r--;
    }
}
```

Quicksort

- Partition the array into two subarrays according to some pivot value.
- Call itself recursively on each subarray.
- Somewhat like merge sort, except that there is no merge, as partitioning guarantees that left side is smaller than right side.

recQsort

```
public void sort(int left, int right) {  
    if (right-left<=0) //size 1, already sorted  
        return;  
    //partition  
    sort(l, mid-1);  
    sort(mid+1, r);  
}
```

How to Choose a Pivot?

- The pivot should be the key value of an actual data item.
- The choice can be more or less random.
- Let us always pick the last element.
- After partition, if the pivot is inserted between left and right partitions, it will be at its correct location.
- Bad pivot choice can result in $O(n^2)$ time

Putting it together

- Make partition a method
 - problem partition returns two values
 - new lo and new hi
- Create a class tht holds these values
 - only needed inside the QuickSort class
 - Java allows such things
 - `public class QuickSort {
 private class Parter {`
 - the private class exists ONLY within the public class

Stable and Unstable Sorting

- Stable
 - elements with equal keys retain order
- Unstable
 - order of elements with equal keys is arbitrary
- When do you care?
- Insertion, Selection, Bubble, Merge, Quick
 - which are they?

Quicksort example applets

- <http://java.sun.com/applets/jdk/1.0/demo/SortDemo/example1.html>
- <http://mainline.brynmawr.edu/Courses/cs206/fall2004/WorkshopApplets/Chap07/QuickSort1/QuickSort1.html>