LECTURE 10

SORTING ALGORITHMS

OUTLINE

- Bubble Sort
- Insertion Sort
- Merge Sort
- Quick Sort

SORTING RUNTIME

Algorithm	Worst Case	Best Case
Selection Sort (Last Lecture)	$O(n^2)$	$O(n^2)$
Bubble Sort		
Insertion Sort		
Merge Sort		
Quick Sort		

BUBBLE SORT

- Iterate through array one element at a time
  - If the next element is larger (or smaller, depending on sorting) than the current element, swap them
- Repeat until sorted

First Pass Example: Sort From Largest to smallest

Starting Array	87	68	99	71	66	59	60

Don’t swap

87	68	99	71	66	59	60

Swap

87	99	68	71	66	59	60

Swap

87	99	71	68	66	59	60

On the second pass, this will be swapped

…

Swap

BUBBLE SORT

- Iterate through array one element at a time
  - If the next element is larger (or smaller, depending on sorting) then the current element, swap them
- Repeat until sorted

Second Pass Example: Sort From Largest to smallest

2nd Pass Array

87	99	71	68	66	60	59

Swap

99	87	71	68	66	60	59

Continuing thru array, it is in sorted order…

BUBBLE SORT IMPLEMENTATION

- What is the best and worst case runtime?

```
int swaps;
do
{
   swaps = 0;
   for(int i=0; i<N-1; i++) //N is my_array size
   {
      if (my_array[i+1]>my_array[i])
      {
         swap(my_array[i+1], my_array[i]);
         swaps++;
      }
   }
}while(swaps>0);
```

How many passes on the original array?

87	68	99	71	66	59	60

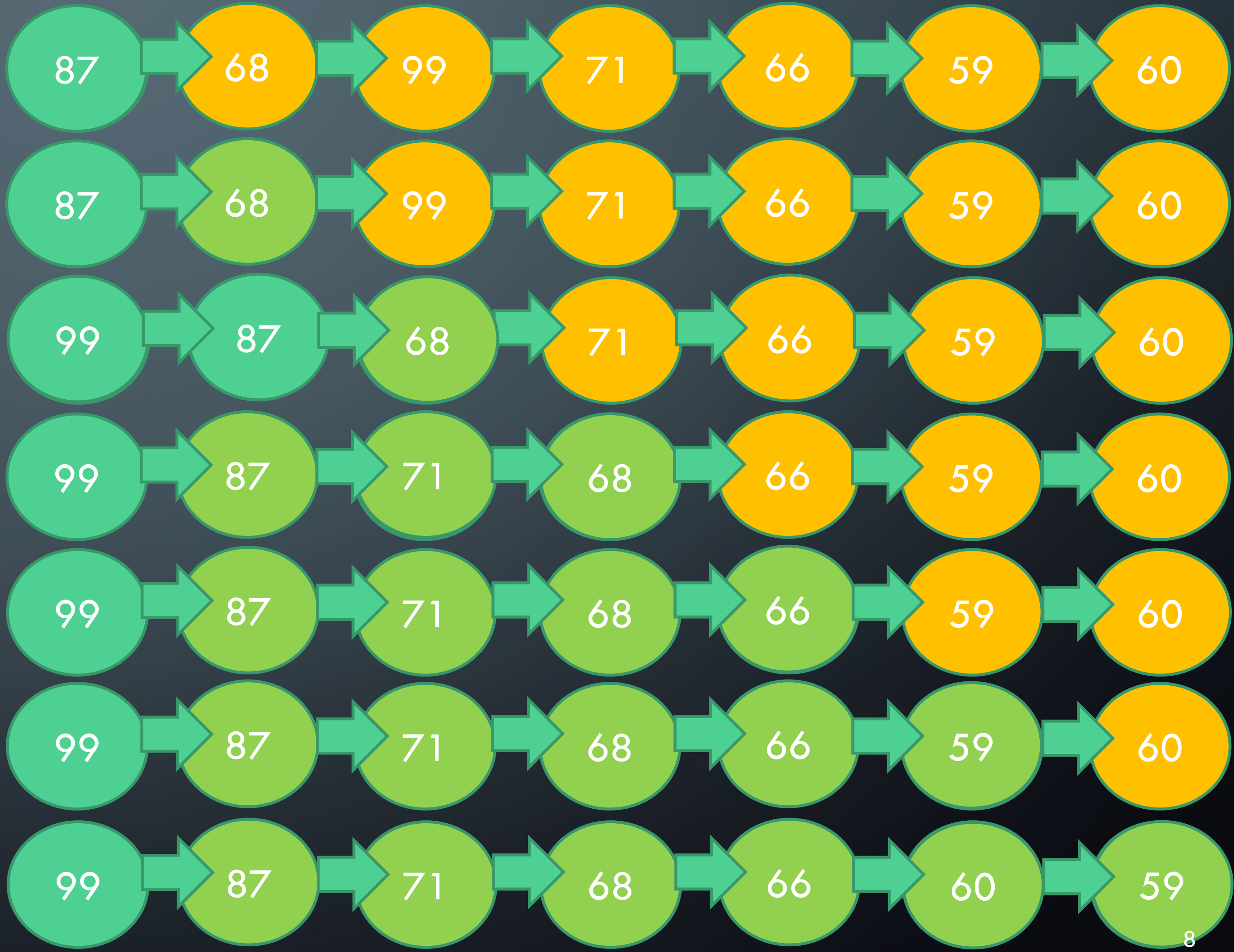
SORTING RUNTIME

Algorithm	Worst Case	Best Case
Selection Sort (Last Lecture)	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n^2)$	$O(n)$
Insertion Sort		
Merge Sort		
Quick Sort		

INSERTION SORT

- Treat the list as broken into a sorted and unsorted section, starting with first element as the sorted section
- One-by-one, grab the next element in the unsorted section and insert it into the position it belongs in the sorted section

Example: Sort From Largest to smallest



INSERTION SORT IMPLEMENTATION

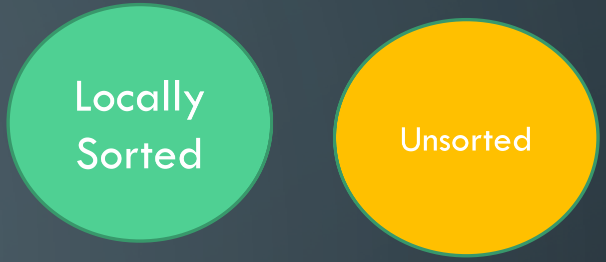
- What is the best and worst case runtime?
  - What happens if the list starts in reverse sorted order?

```
for (int i=1; i<N; i++)
{
    T val = list[i];
    for (int j = i-1; j>=0; j--)
    {
        if (val < list[j])
            break;
        swap(val, list[j]);
    }
}
```

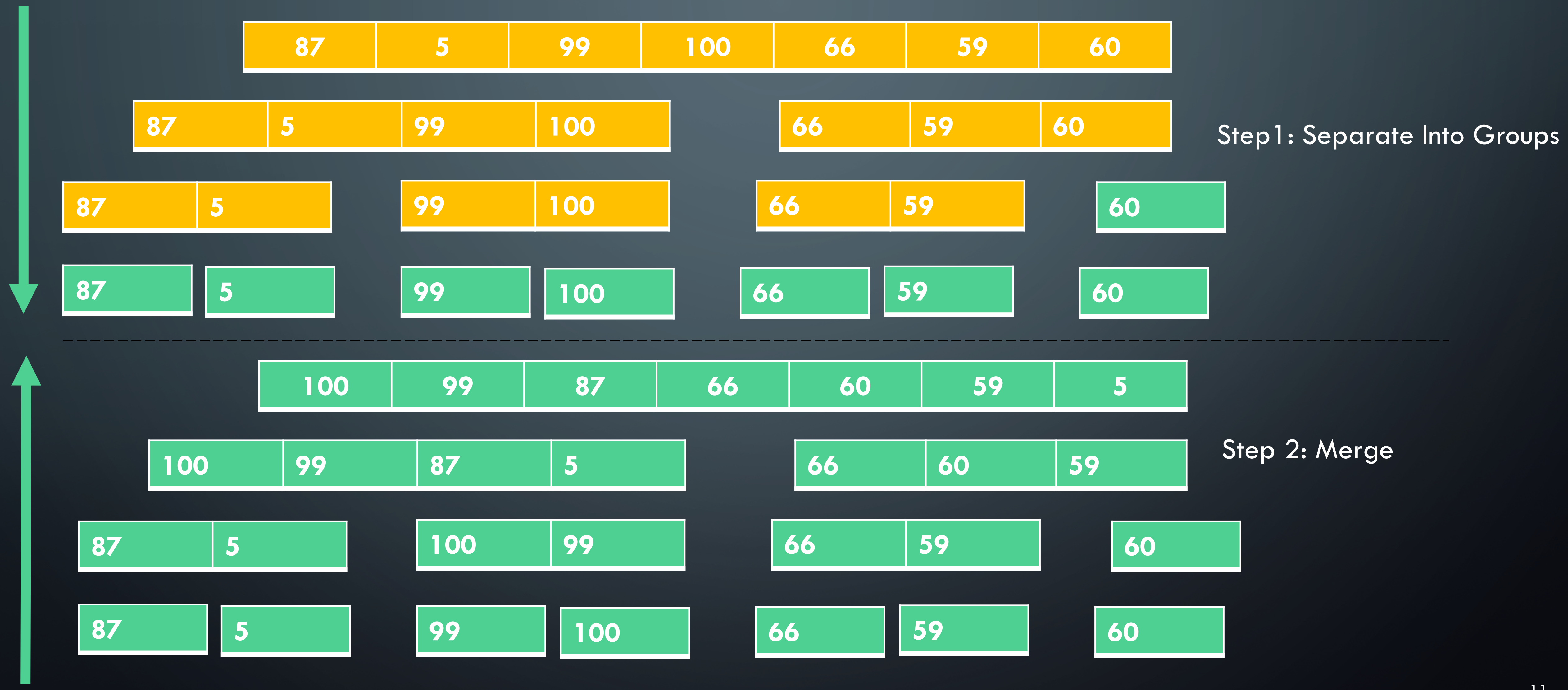
SORTING RUNTIME

Algorithm	Worst Case	Best Case
Selection Sort (Last Lecture)	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n^2)$	$O(n)$
Insertion Sort	$O(n^2)$	$O(n)$
Merge Sort		
Quick Sort		

MERGE SORT

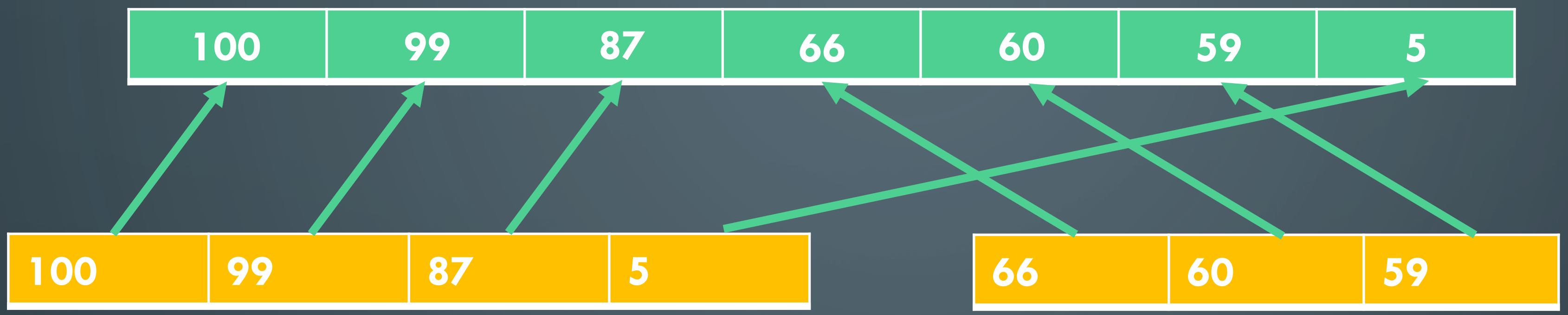


- Recursively sort sub-lists in the list



MERGE OPERATION

- Recursively sort sub-lists in the list



a. 100 > 66, move 100 from the left list to the merged list
b. 99 > 66, move 99 from left list to merged list
c. 87 > 66, move 87 from left list to merged list
d. 5 < 66, move 66 from right list to merged list
e. 5 < 60, move 60 from right list to merged list
f. 5 < 59, move 59 from right list to merged list
g. Right list is empty, move 5 from left list to merged list

How many operations?

MERGE SORT IMPLEMENTATION

- What is the runtime?
  - With an optimal algorithm, what is the runtime of Merge?
  - What is the depth of MergeSort?

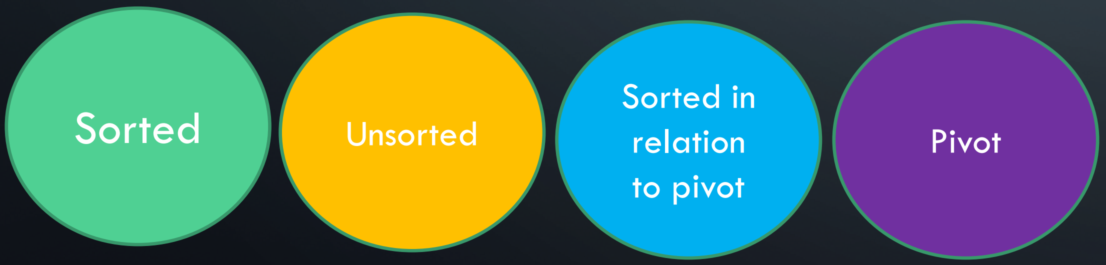
```
function MergeSort(list, first, last)
{
// first and last are indices
// defining the sublist
  if(first < last)
  {
    mid = floor ( (first + last)/2);
    MergeSort(list, first, mid);
    MergeSort(list, mid+1, last);
    Merge(list, first, mid, last);
  }
}
```

SORTING RUNTIME

Algorithm	Worst Case	Best Case
Selection Sort (Last Lecture)	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n^2)$	$O(n)$
Insertion Sort	$O(n^2)$	$O(n)$
Merge Sort	$O(nlog(n))$	$O(nlog(n))$
Quick Sort		

QUICK SORT

- Pick an element from the list
- Partition so that everything to the left of the pivot is greater (lesser) than the pivot, and everything to the right is lesser (greater) than the pivot
- Recursively apply to sub-lists on the left and right of pivot



Sort largest to smallest, Partitioning 1

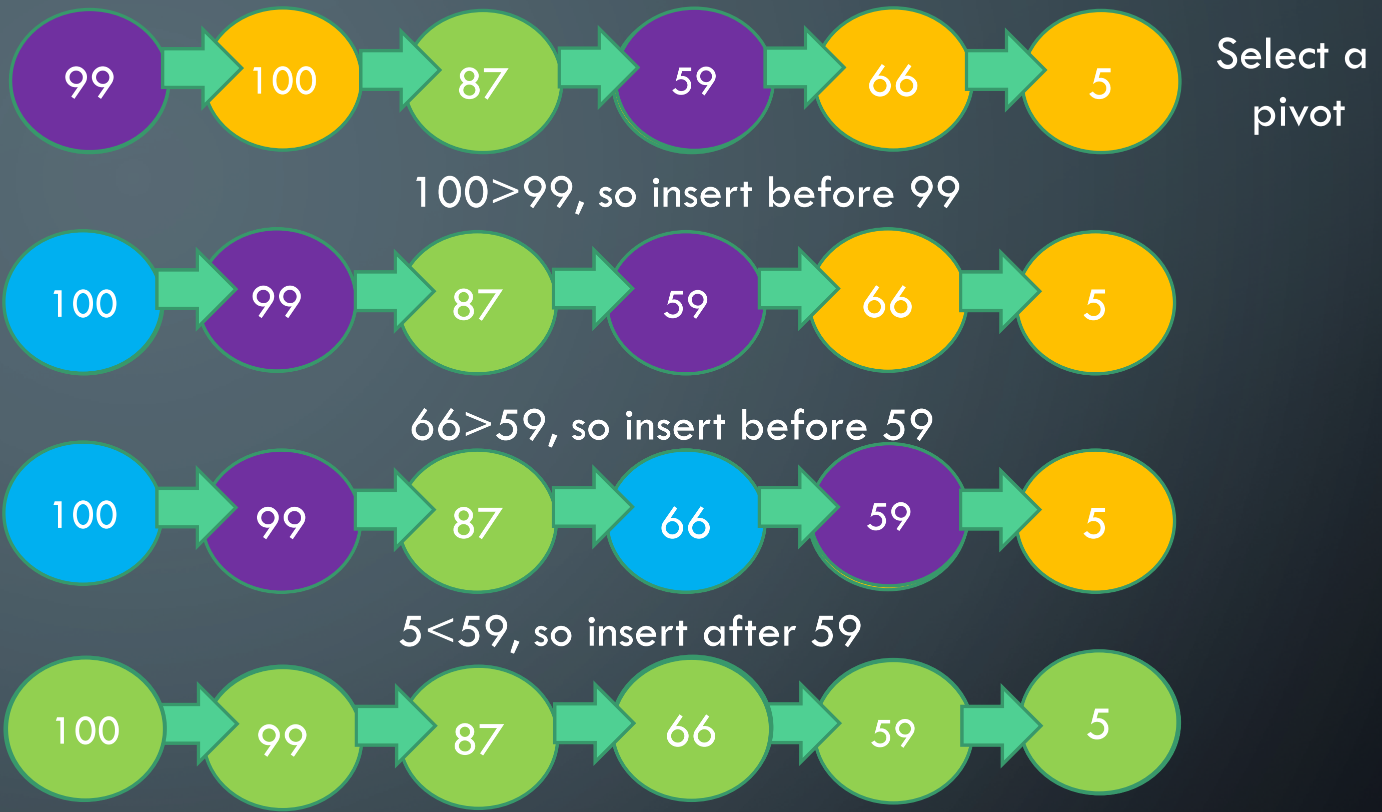
Select a pivot

87 is now in sorted order!

QUICK SORT

- Pick an element from the list
- Partition so that everything to the left of the pivot is greater (lesser) than the pivot, and everything to the right is lesser (greater) than the pivot
- Recursively apply to sub-lists on the left and right of pivot

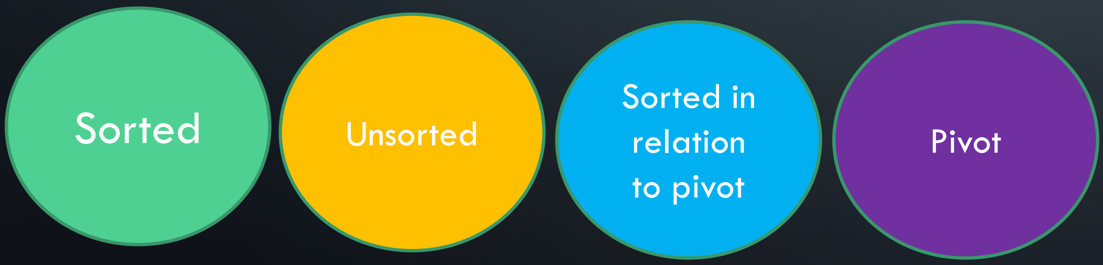
Sort largest to smallest, Partitioning2



99 and 59 are now in sorted order!

100, 66, and 5 are partitions of 1.

They are now in sorted order!



QUICK SORT IMPLEMENTATION

- What is the runtime of partition (shown on previous slide)?
- What is the best and worst case depth of calls to QuickSort?
- What is the worst case for quick sort, and its runtime?

```
function QuickSort(list, first, last)
{
  if first < last
  {
    pivot = partition(list, first, last);
    QuickSort(list, first, pivot);
    QuickSort(list, pivot + 1, last);
  }

}
```

SORTING RUNTIME

Algorithm	Worst Case	Best Case
Selection Sort (Last Lecture)	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n^2)$	$O(n)$
Insertion Sort	$O(n^2)$	$O(n)$
Merge Sort	$O(n\log(n))$	$O(n\log(n))$
Quick Sort	$O(n^2)$	$O(n\log(n))$

RADIX SORT

- Binned sort, sorting by one factor one after another.
- Consider a deck of cards
  - Create 4 bins
  - In each bin put one card in order from
    2, 3, …, 10, J, Q, K, A.
  - From each "bin" place in final array in suit order
  - Clubs, Diamonds, Hearts, Spades
- Now in sorted order
- What's the complexity?

123	2154	222	4	283	1560	1061	2150
0123	2154	0222	0004	0283	1560	1061	2150
1560	2150	1061	0222	0123	0283	2154	0004
0004	0222	0123	2150	2154	1560	1061	0283
0004	1061	0123	2150	2154	0222	0283	1560
0004	0123	0222	0283	1061	1560	2150	2154

SORTING RUNTIME

Algorithm	Worst Case	Best Case
Selection Sort (Last Lecture)	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n^2)$	$O(n)$
Insertion Sort	$O(n^2)$	$O(n)$
Merge Sort	$O(n\log(n))$	$O(n\log(n))$
Quick Sort	$O(n^2)$	$O(n\log(n))$
Radix Sort	$O(n)$	$O(n)$

IMPORTANT CONSIDERATIONS

- Through information theory, can prove best possible sort on average is O(nlog(n)) runtime (excl. radix)
- Concept: stability. If you perform the sort multiple times, you will get the same results. True for all sorting algorithms, except the variation of QuickSort where the pivot is chosen randomly
- Space overhead: all sorting algorithms except MergeSort and RadixSort sort in-place

Algorithm	Added Space Overhead
Selection Sort (Last Lecture)	O(1)
Bubble Sort	O(1)
Insertion Sort	O(1)
Merge Sort	O(n)
Quick Sort	O(1)
Radix Sort	O(d*n)

ASSIGNMENT/HOMEWORK

- Read Carrano pp 353 – 366, 435-443
- HW4 due on Tuesday
- HW5 released:
  - Carrano Exercises Chapter 11: 3, 4, 15
  - Carrano Programming Problem Chapter 11: 1