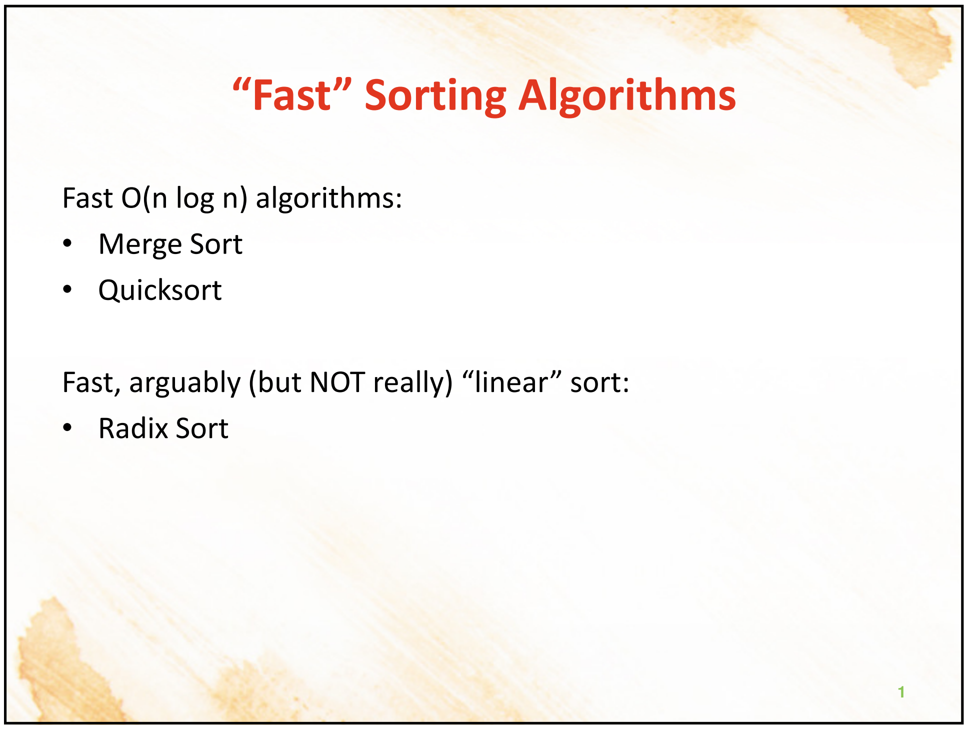


“Fast” Sorting Algorithms

Fast O(n log n) algorithms:

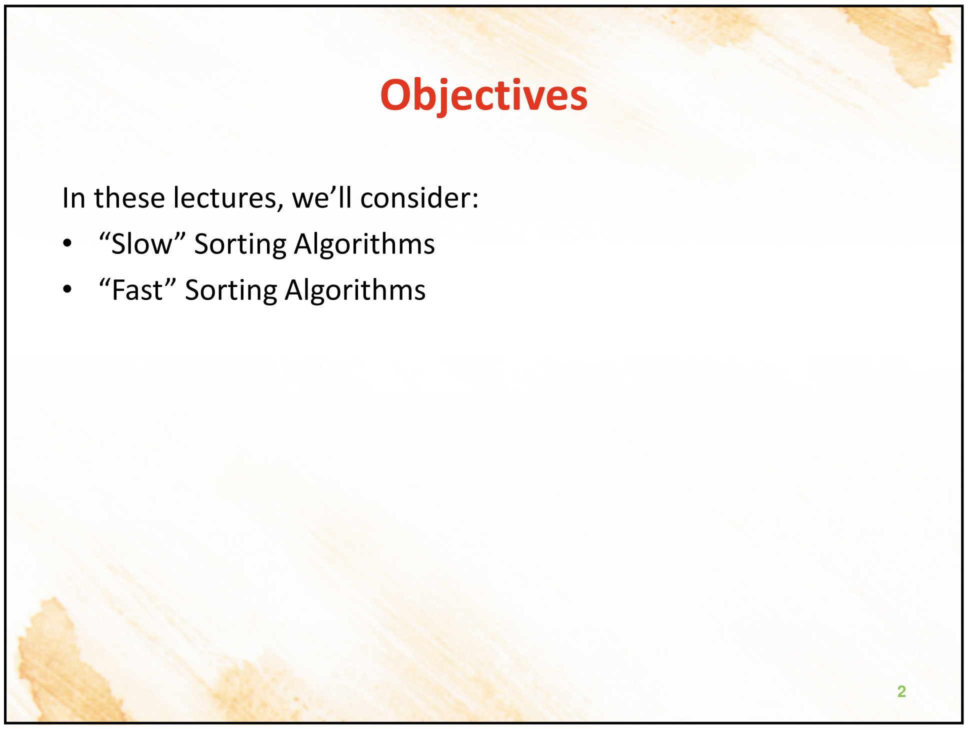
- Merge Sort
- Quicksort

Fast, arguably (but NOT really) “linear” sort:

- Radix Sort

1

1



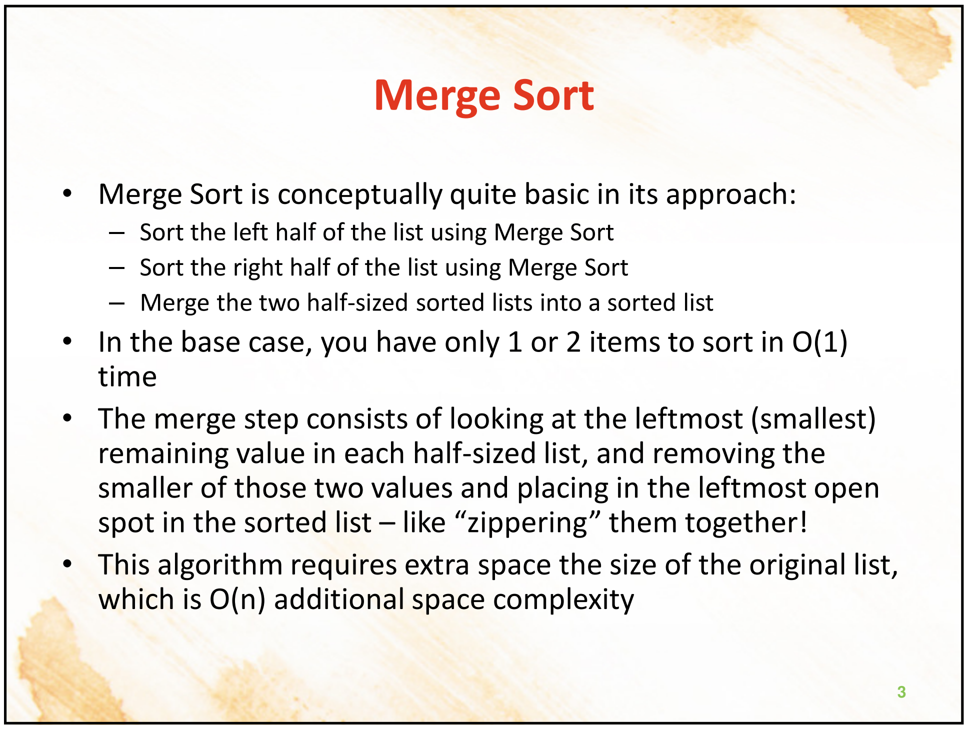
Objectives

In these lectures, we’ll consider:

- “Slow” Sorting Algorithms
- “Fast” Sorting Algorithms

2

2

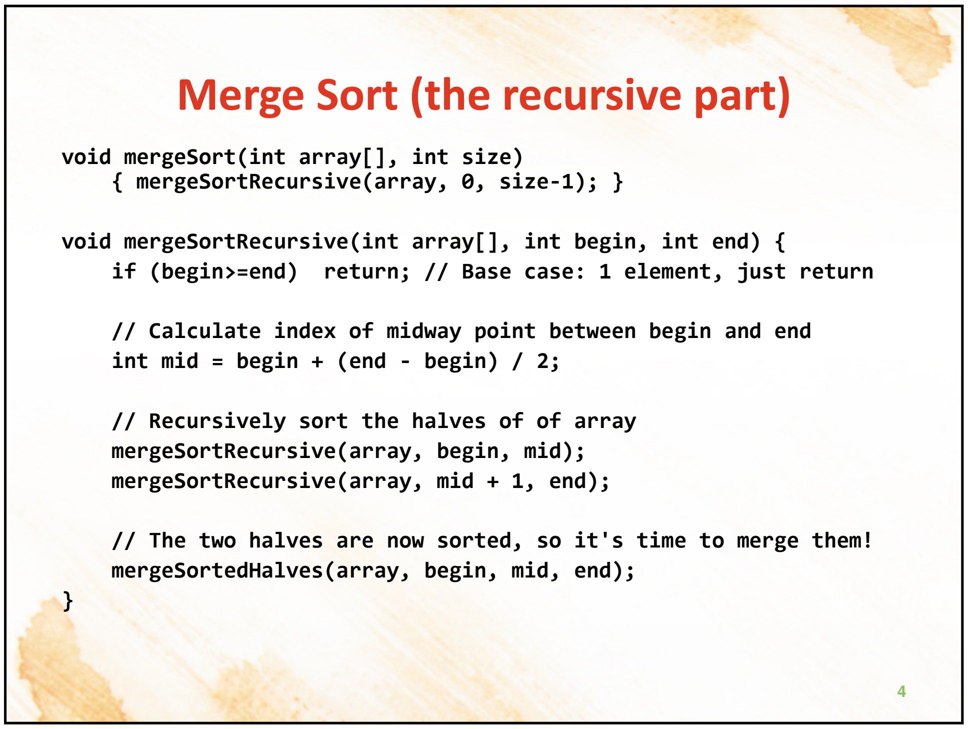


Merge Sort

- Merge Sort is conceptually quite basic in its approach:
  - Sort the left half of the list using Merge Sort
  - Sort the right half of the list using Merge Sort
  - Merge the two half-sized sorted lists into a sorted list
- In the base case, you have only 1 or 2 items to sort in O(1) time
- The merge step consists of looking at the leftmost (smallest) remaining value in each half-sized list, and removing the smaller of those two values and placing in the leftmost open spot in the sorted list – like "zippering" them together!
- This algorithm requires extra space the size of the original list, which is O(n) additional space complexity

3

3



Merge Sort (the recursive part)

```
void mergeSort(int array[], int size)
    { mergeSortRecursive(array, 0, size-1); }

void mergeSortRecursive(int array[], int begin, int end) {
    if (begin>=end)  return; // Base case: 1 element, just return

    // Calculate index of midway point between begin and end
    int mid = begin + (end - begin) / 2;

    // Recursively sort the halves of of array
    mergeSortRecursive(array, begin, mid);
    mergeSortRecursive(array, mid + 1, end);

    // The two halves are now sorted, so it's time to merge them!
    mergeSortedHalves(array, begin, mid, end);
}
```

4

4

Quicksort

- Often considered the "best" sort out there
- Similar to Merge Sort in that it divides array into two parts
- However, it does its work *before* the recursion
- It selects one value in the list to be the "pivot"
- It then places the pivot so that all values <= pivot are on the left of it, and all value >= pivot are on the right of it
- The pivot is now exactly where it's supposed to be in the list
- Quicksort is then recursively called on:
  - The list of values left of the pivot that are <= pivot
  - The list of values right of the pivot that are >= pivot

5

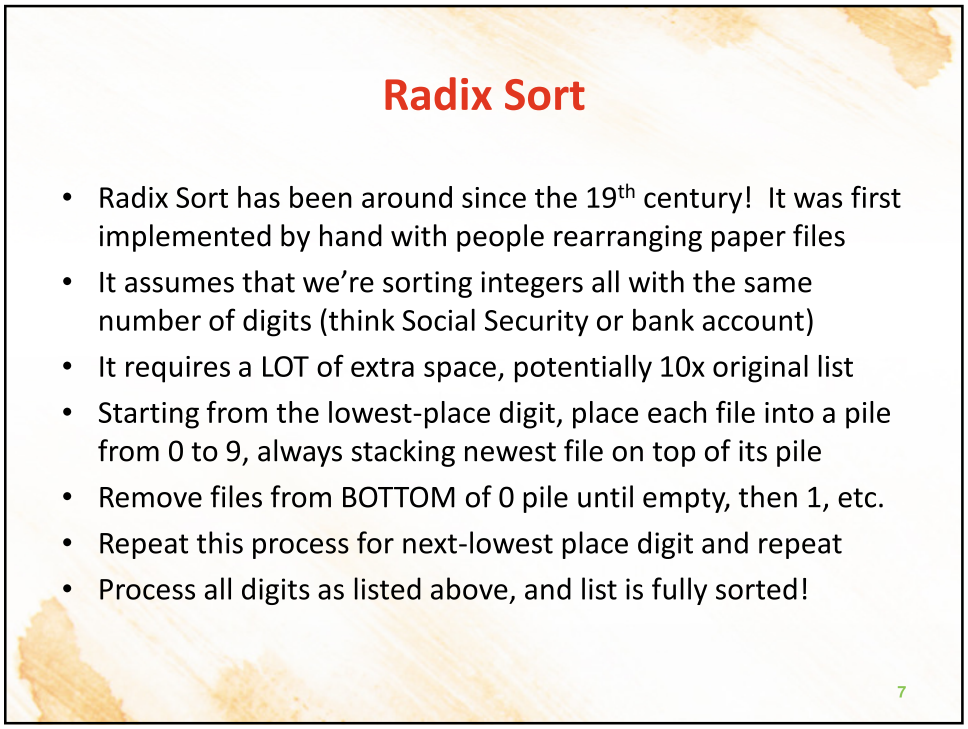
5

Quicksort (some code omitted :-)

```
void quickSort(int data[], int size)
    { quickSortRecursive(data, 0, size-1); }

void quickSortRecursive(int data[], int start, int end) {
    if (end <= start) return; // Base case-array size 0 or 1
    if ((end - start == 1) && (data[end] < data[start]))
         { swap(data[start], data[end]); return; } // Base case-array size 2

   // Recursive case - array of size 3 or more – select pivot
   int pivotIndex = start;  // We'll arbitrarily choose leftmost value in list

   // CODE TO MOVE VALUES <= PIVOT TO LEFT OF PIVOT, >= PIVOT TO RIGHT OF PIVOT
   // We will assume pivotIndex is changed to point where the pivot moves to

   quickSortRecursive(data, start, pivotIndex - 1);
   quickSortRecursive(data, pivotIndex + 1, end);

   // After these calls recurse all the way to the base cases, we're sorted!
}
```

6

6

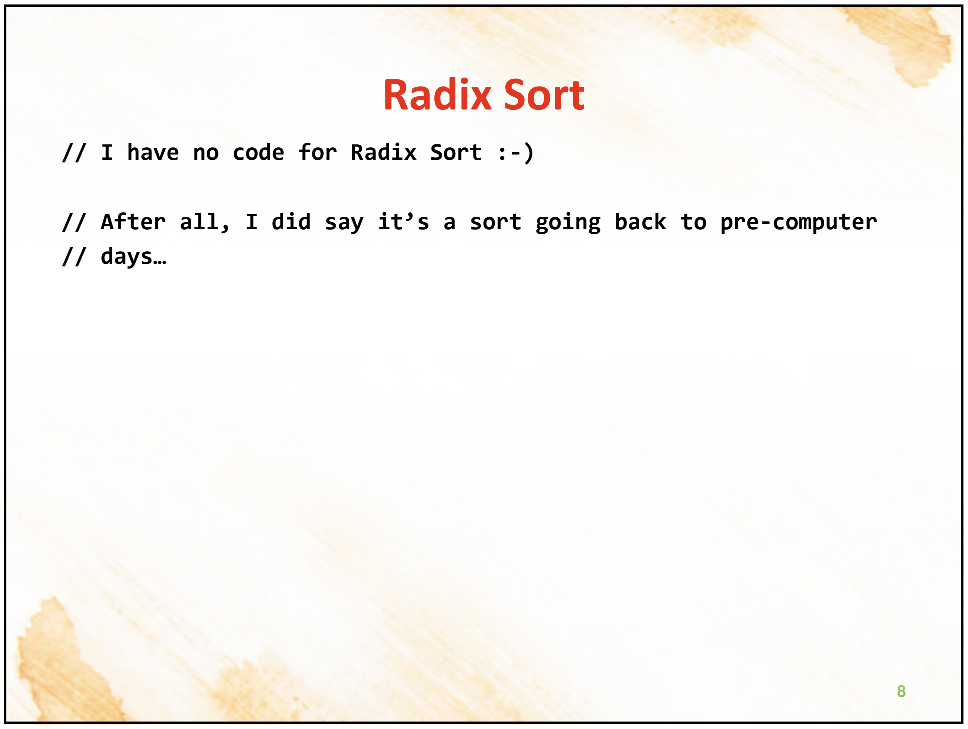


Radix Sort

- Radix Sort has been around since the 19th century! It was first implemented by hand with people rearranging paper files
- It assumes that we're sorting integers all with the same number of digits (think Social Security or bank account)
- It requires a LOT of extra space, potentially 10x original list
- Starting from the lowest-place digit, place each file into a pile from 0 to 9, always stacking newest file on top of its pile
- Remove files from BOTTOM of 0 pile until empty, then 1, etc.
- Repeat this process for next-lowest place digit and repeat
- Process all digits as listed above, and list is fully sorted!

7

7



Radix Sort

```
// I have no code for Radix Sort :-)

// After all, I did say it's a sort going back to pre-computer
// days…
```

8

8