CSCI 136
Data Structures & Advanced Programming

Lecture 15

Spring 2020

Profs Bill & Dan

Administrative Details

- We will navigate the chaos together.
  - Be proactive; we understand and we want to help
  - The situation is unreasonable, we are not
- Remember, nothing about this is fair, but nothing about this is anyone's fault. We have to be good to each other and to ourselves.
  - There is more than CS136 in our lives.

Last Time

- `Comparable`: objects impose an ordering
- `Comparator`: a separate class that imposes an ordering on objects
- More “simple” sorting
  - Bubble, Insertion, and Selection Sorts
  - General behaviors, Big-O, pros/cons
- Maud’s email

Today's Plan

- Merge sort
- Quick sort
- Looking ahead

“Simple” Sorts Review

- Bubble, insertion, and selection sorts are $O(n^2)$ in the worst case, but:
  - They are fast to implement
  - They are more than good enough for small sets
  - Insertion sort is $O(n)$ for sorted lists

- But we can do better! (asymptotically…)

Merge Sort

- A divide and conquer algorithm
- Merge sort works as follows:
  - Base case:
    - If the list is of length 0 or 1, then *it is already sorted*. Return the sorted list.
  - Divide the unsorted list into two sublists of about half the size of original list.
  - Recursive call:
    - Sort each sublist by re-applying merge sort.
  - Merge the two sublists back into one sorted list.

Merge Sort

- [8 14 29 1 17 39 16 9]
- [8 14 29 1] [17 39 16 9] split
- [8 14] [29 1] [17 39] [16 9] split
- [8] [14] [29] [1] [17] [39] [16] [9] split
- [8 14] [1 29] [17 39] [9 16] merge
- [1 8 14 29] [9 16 17 39] merge
- [1 8 9 14 16 17 29 39] merge

Transylvanian Merge Sort Folk Dance

Merge Sort

- How would we implement it?
- Pseudocode:

```
//recursively mergesorts A[from..To] "in place"
void recMergeSortHelper(A[], int from, int to)
  if ( from < to )
     // find midpoint
     mid = (from + to) / 2
     //sort each half
     recMergeSortHelper(A, from, mid)
     recMergeSortHelper(A, mid+1, to)
     // merge sorted lists
     merge(A, from, to)
```

But `merge` hides a number of important details....

Merge Sort

- How would we implement it?
  - Review `MergeSort.java`
  - Note carefully how temp array is used to reduce copying
  - Make sure the data is in the correct array!
- Time Complexity?
  - Takes at most `2k` comparisons to merge two lists of size `k`
  - Number of splits/merges for list a of size `n`? It's `log n`
  - Claim: At most time `O(n log n)`...We'll see soon...
- Space Complexity?
  - `O(n)`?
  - Need an extra array, so really `O(2n)`!
    - But `O(2n) = O(n)`

Merge Sort = $O(n \log n)$

- [8 14 29 1 17 39 16 9]
- [8 14 29 1] [17 39 16 9] split
- [8 14] [29 1] [17 39] [16 9] split
- [8] [14] [29] [1] [17] [39] [16] [9] split
- [8 14] [1 29] [17 39] [9 16] merge
- [1 8 14 29] [9 16 17 39] merge
- [1 8 9 14 16 17 29 39] merge

split steps: log n
merge steps: log n

merge takes at most n comparisons per line

Merge Sort

- Unlike Bubble, Insertion, and Selection sort, Merge sort is a divide and conquer algorithm
  - Bubble, Insertion, Selection sort: `O(n`^{2}`)`
  - Merge sort: `O(n log n)`
- Are there any problems or limitations with Merge sort?
- Why would we ever use any other algorithm for sorting?

Problems with Merge Sort

- Need extra temporary array
  - If data set is large, this could be a problem
- Waste time copying values back and forth between original array and temporary array
- Can we avoid this?

Quick Sort

- Quick sort is designed to behave much like Merge sort, without requiring extra storage space

Merge Sort	Quick Sort
Divide list in half	Partition* list into 2 parts
Sort halves	Sort parts
Merge halves	Join* sorted parts

Recall Merge Sort

```
private static void mergeSortRecursive(Comparable data[],
                    Comparable temp[], int low, int high) {
   int n = high-low+1;
   int middle = low + n/2;
   int i;

   if (n < 2) return;
   // move lower half of data into temporary storage
   for (i = low; i < middle; i++) {
       temp[i] = data[i];
   }
   // sort lower half of array
   mergeSortRecursive(temp,data,low,middle-1);
   // sort upper half of array
   mergeSortRecursive(data,temp,middle,high);
   // merge halves together
   merge(data,temp,low,middle,high);
}
```

Quick Sort

```
// pre: low <= high
// post: data[low..high] in ascending order
public void quickSortRecursive(Comparable data[],
                    int low, int high) {
        int pivot;
        /* base case: low and high coincide */
        if (low >= high) return;

        /* step 1: split using pivot */
        pivot = partition(data, low, high);
        /* step 2: sort small */
        quickSortRecursive(data, low, pivot-1);
        /* step 3: sort large */
        quickSortRecursive(data, pivot+1, high);
}
```

Partition

1. Put first element (pivot) into sorted position
2. All to the left of “pivot” are smaller and all to the right are larger
3. Return index of “pivot”

Partition by Hungarian Folk Dance

Partition

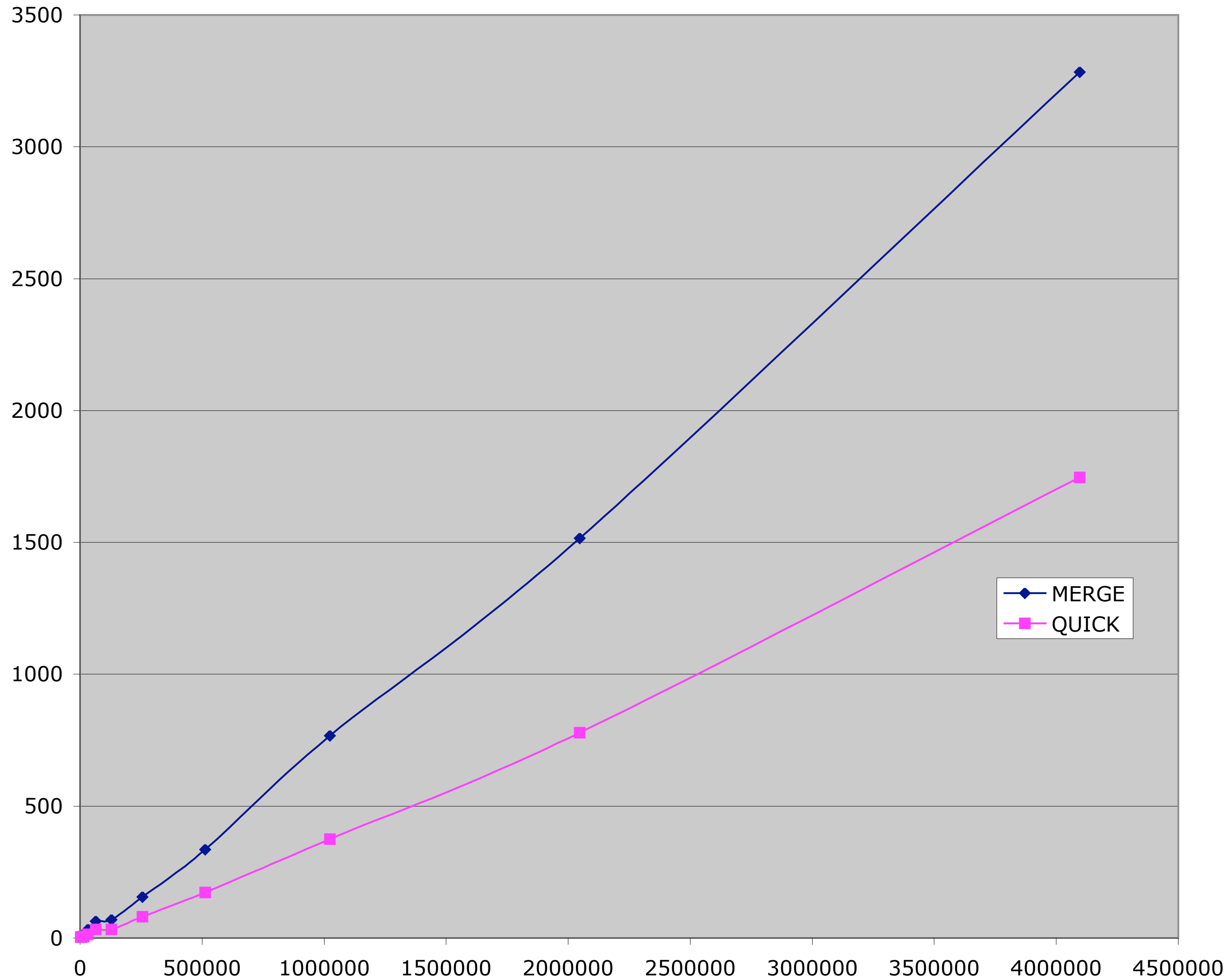
```
int partition(int data[], int left, int right) {
  while (true) {
    while (left < right && data[left] < data[right])
      right--;
    if (left < right) {
      swap(data, left++, right);
    } else {
      return left;
    }

    while (left < right && data[left] < data[right])
      left++;
    if (left < right) {
      swap(data, left, right--);
    } else {
      return right;
    }
  }
}
```

Complexity

- Time:
  - Partition is O(n)
  - If partition breaks list exactly in half, same as merge sort, so O(n log n)
  - If data is already sorted, partition splits list into groups of 1 and n-1, so $O(n^2)$
- Space:
  - O(n) (so is MergSort)
    - In fact, it's n + c compared to 2n + c for MergeSort

Merge vs. Quick



Food for Thought…

- How to avoid picking a bad pivot value?
  - Pick median of 3 elements for pivot? (heuristic!)

- Idea: combine selection sort with quick sort
  - For small n, selection sort is faster
  - Switch to selection sort when elements is <= 7
  - Switch to selection/insertion sort when the list is almost sorted (partitions are very unbalanced)
    - Another heuristic!

Sorting Wrapup

	Time	Space
Bubble	Worst: $O(n^2)$ Best: $O(n)$ - if "optimiazed"	$O(n)$: $n + c$
Insertion	Worst: $O(n^2)$ Best: $O(n)$	$O(n)$: $n + c$
Selection	Worst = Best: $O(n^2)$	$O(n)$: $n + c$
Merge	Worst = Best:: $O(n \log n)$	$O(n)$: $2n + c$
Quick	Average = Best: $O(n \log n)$ Worst: $O(n^2)$	$O(n)$: $n + c$

More Skill-Testing (Try these at home)

Given the following list of integers:

9 5 6 1 10 15 2 4

1) Sort the list using Bubble sort. Show your work!
2) Sort the list using Insertion sort. Show your work!
3) Sort the list using Merge sort. Show your work!
4) Verify the best and worst case time and space complexity for each of these sorting algorithms as well as for selection sort.

Looking Ahead

- So far we've focused on the `List` interface and linear structures
  - Vector and Linked Lists
- We will build more powerful structures *using these ideas as building blocks* so that we can:
  - search faster
  - encode *relationships* between objects
  - implement concepts present in our daily lives

Linear Structures with Restrictions

- Idea: take a “list”, and add some restrictions
  - Stack: you can only add/remove elements from the top
  - Queue: enqueue (add) elements at the back, dequeue (remove) from elements from the front

Structures With Multiple Links

- Idea: take a “list”, allow more than one link per node
  - Binary tree:
    - each node is a leaf or has two “children”
  - Graph:
    - arbitrary relationships between nodes

Random Access Hash Structures

- Idea: take an array, assign elements a “home” based on their values
  - Hash function:
    - One-way function that takes a value and yields an index
    - Ideally, evenly distribute values throughout the space
    - Good hash functions have nice mathematical properties that make lookup approximately O(1)!

Stay Safe and Healthy

- It's not going to be easy, but we will work together to make the course a success
  - We want to support you! BUT
  - It is up to you to let us know when things aren't going as planned
- We know what it is like to be stuck and not understand something…
  - Do not accept defeat alone. We are a team.

Stay Safe and Healthy

- If things come up in your life outside of class, let us know
  - We will find ways to accommodate your situation
- If things come up in class, let us know
  - We will find ways to resolve issues on our end

Stay Safe and Healthy

- Find routines and practices that work for you
  - Want a study partner from CS136?
    - Reach out
  - Hard time concentrating?
    - "Work Uniform", mynoise.net, daily planner
  - Get the big picture, but not the details?
    - Teach a friend!
  - Easily distracted?
    - draw pictures on paper, take physical notes, get away from a computer