Sorting

Week 5

Required Activities

- Check Announcements regularly (every 2-3 days)
- Read DSA book: Chapter 4
- Complete and submit Assignment 2 (due Friday 11/30)

Sorting

- Arranging data in order such as ascending or descending or alphabetical based on a key

- Two categories:
  - **Internal sorting** – sorted in memory
  - **External sorting** - data is so large, some of it is in auxiliary memory (e.g. hard disk). Merge sort is often used.

Sort Complexity

	Time complexity	Time complexity	Time complexity	
Internal sorting	**Best case**	**Average case**	**Worst case**	**Space complexity**
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)
Quick Sort	O(n log n)	O(n log n)	$O(n^2)$	O(log n)

O(n) is faster than $O(n^2)$
$O(n^2)$ is faster in real life for smaller value of n than O(n log n) (e.g. n=100) but as n grows, O(n log n) it approaches O(n) and so is faster

Bubble Sort

- Passes though list multiple time and exchanging two elements if not in proper order.
- Least efficient but O(n) when list close to already sorted.

8 5 7 1 9 3

Start front	**8 5** 7 1 9 3 compare and swap => 5 8 7 1 9 3
	5 **8 7** 1 9 3 compare and swap => 5 7 8 1 9 3
	5 7 **8 1** 9 3 compare and swap => 5 7 1 8 9 3
	5 7 1 **8 9** 3 compare and leave => 5 7 1 8 9 3
	5 7 1 8 **9 3** compare and swap => 5 7 1 8 3 9
Start again	**5 7** 1 8 3 9 compare and leave => 5 7 1 8 3 9
	5 **7 1** 8 3 9 compare and swap => 5 1 7 8 3 9
	…

Stops when went through all data and nothing was swapped

Selection Sort

- Looks for smallest element and puts in first place by doing a swap, then searches for next element and puts in second place doing a swap, etc.
- No difference if already sorted.
- Faster than Bubble sort.

8 5 7 1 9 3

8 5 7 **1** 9 3	find smallest and swap to 1st place =>	**1** 5 7 **8** 9 3
1 5 7 8 9 **3**	find next smallest and swap to 2nd place =>	1 **3** 7 8 9 **5**
1 3 7 8 9 **5**	find next smallest and swap to 3nd place =>	1 3 **5** 8 9 **7**

…

Keeps going until processed whole list

Insertion Sort

- Takes first element compares to next and swaps, then take the next element and insert in the right place for the sorted portion (currently two elements), so now three are sorted so looks at next elements and inserts in right place in those 3 sorted elements, etc.
- More efficient than bubble sort and may be more efficient than selection as more data is already sorted.
- Used for small n because becomes slow $O(n^2)$ the greater the size of n.

8 5 7 1 9 3

8 5 7 1 9 3	compare and swap =>	**5** 8 7 1 9 3
5 **8** 7 1 9 3	checks next element and already in place =>	5 8 7 1 9 3
5 8 **7** 1 9 3	checks next element and inserts in right place =>	5 **7** 8 1 9 3
5 8 7 **1** 9 3	checks next element and inserts in right place =>	**1** 5 7 8 9 3

…

Keeps going until processed whole list

Merge Sort

- Divides data into smaller lists, sorts, and merges into one sorted list
- Uses temporary lists to merge when use array (better implementation with linked list)
- Splitting is simple but merging hard (compared to quick sort where opposite)

8 5 7 1 9 3

Keep dividing list until 1 unit	8 5 7 1 9 3	
	…	
	8 5 7 1 9 3	
	8 **5** 7 1 9 3	compare with adjacent and merge sorting 5 8
	5 8 **7** **1** 9 3	compare with adjacent and merge sorting 1 7
	5 8 1 7 9 3	compare with adjacent and merge sorting 3 9
	5 8 **1 7** 9 3	compare with adjacent and merge sorting 1 5 7 8
	…	

Keeps merging until back to whole sorted list

Quick Sort

- Selects pivot element where list is split in two. The pivot can be random or median of the first, middle, and last element of the partition (or other method); then sorts elements to be less value on left of pivot and greater on right of pivot. Recursively splits each sub-list and sorts
- Worse case when elements already sorted and best when totally unsorted (opposite of bubble sort)
- Splitting is hard but merging easy (compared to merge sort where opposite)
- Better for arrays than linked list since sorts in place
- For real applications often fastest because of low overhead

8 5 7 1 9 3

Determine pivot (e.g. middle)	8 5 **7** 1 9 3
Look for element in left list greater than pivot value	8
Look for element in right list smaller than pivot value	3
swap if right is smaller than left	3 5 7 1 9 8
Keep doing for all elements	3 5 1 7 9 8
Select pivot in left substring and repeat	3 **5** 1 7 9 8
Look left and right and swap	3 1 5 7 9 8
	…
Keep going recursively in left sub-lists	
Select pivot in left substring and repeat	3 5 1 7 **9** 8
Look left and right and swap	3 5 1 7 8 9
Keep going recursively in right sub-lists	

Keeps dividing and sorting recursively until whole sorted list

Questions ?

- Post in the discussions
- Send email to RMcFadden@HarrisburgU.edu
- Respond usually within 48hours