Topic 17
Faster Sorting

"The bubble sort seems to have nothing to recommend it, except a catchy name and the fact that it leads to some interesting theoretical problems."

\- Don Knuth

Previous Sorts

- Insertion Sort and Selection Sort are both average case $O(N^2)$
- Today we will look at two faster sorting algorithms.
  - quicksort
  - mergesort

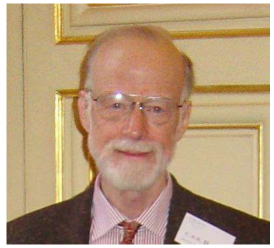
Properties of Sorting Algorithms

- In place?
  - Do we use another data structure or not?
  - Program stack *typically* not considered another data structure if only using O(log N) space
- Comparison?
  - Works by comparing the items to be sorted to each other?
  - *How could we not?*
- Stable?
  - Next slide!

Stable Sorting

- A property of sorts
- If a sort guarantees the relative order of equal items stays the same then it is a *stable sort*
- $[7_1, 6, 7_2, 5, 1, 2, 7_3, -5]$ original data
  - subscripts added for clarity
- $[-5, 1, 2, 5, 6, 7_1, 7_2, 7_3]$ sorted data
  - result of stable sort
- Real world example:
  - sort a table in Wikipedia by one criteria, then another
  - sort by country, then by major wins

Quicksort



- Invented by C.A.R. (Tony) Hoare
- A divide and conquer approach that uses recursion

1. If the list has 0 or 1 elements it is sorted
2. otherwise, pick any element p in the list. This is called the ***pivot*** value
3. ***Partition*** the list minus the pivot into two sub lists according to values less than or greater than the pivot. (equal values go to either)
4. return the quicksort of the first list followed by the quicksort of the second list

Quicksort in Action

39 23 17 90 33 72 46 79 11 52 64 5 71

Pick middle element as pivot: 46

Partition list

23 17 5 33 39 11 46 79 72 52 64 90 71

quick sort the less than list

Pick middle element as pivot: 33

23 17 5 11 33 39

quicksort the less than list, pivot now 5

{} 5 23 17 11

quicksort the less than list, base case

quicksort the greater than list

Pick middle element as pivot: 17

and so on….

Quicksort on Another Data Set

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
44	68	191	119	119	37	83	95	191	45	158	130	76	153	39	25

Big O of Quicksort?


```
private static void swapReferences(Object[] a, int index1, int index2) {
    Object tmp = a[index1];
    a[index1] = a[index2];
    a[index2] = tmp;
}

private void quicksort(Comparable[] data, int start, int stop) {
    if(start < stop) {
        int pivotIndex = (start + stop) / 2;

        // Place pivot at start position
        swapReferences(data, pivotIndex, start);
        Comparable pivot = data[start];

        // Begin partitioning
        int j = start;

        // from first to j are elements less than or equal to pivot
        // from j to i are elements greater than pivot
        // elements beyond i have not been checked yet
        for(int i = start + 1; i <= stop; i++ ) {
            //is current element less than or equal to pivot
            if (data[i].compareTo(pivot) <= 0) {
                // if so move it to the less than or equal portion
                j++;
                swapReferences(data, i, j);
            }
        }

        //restore pivot to correct spot
        swapReferences(data, start, j);
        quicksort( data, start, j - 1 );    // Sort small elements
        quicksort( data, j + 1, stop );   // Sort large elements
    } // else start >= stop, 0 or 1 element, base case, do nothing
}
```

Clicker 1

- What are the best case and worst case Orders (Big O) for quicksort?

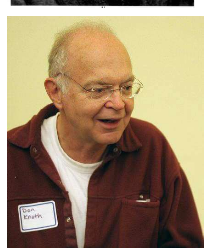
	Best	Worst
A.	O(NlogN)	$O(N^2)$
B.	$O(N^2)$	$O(N^2)$
C.	$O(N^2)$	O(N!)
D.	O(NlogN)	O(NlogN)
E.	O(N)	O(NlogN)

Clicker 2

- Is quicksort always stable?
  A. No
  B. Yes

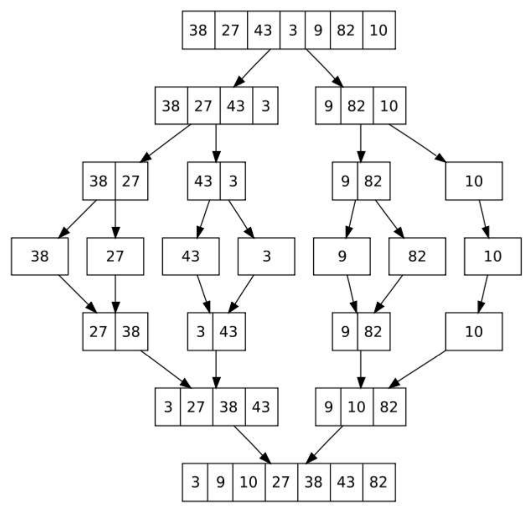
Merge Sort Algorithm

Don Knuth cites John von Neumann as the creator of this algorithm



1. If a list has 1 element or 0 elements it is sorted
2. If a list has more than 1 split into 2 separate lists
3. Perform this algorithm on each of those smaller lists
4. Take the 2 sorted lists and merge them together

Merge Sort



When implementing one temporary array is used instead of multiple temporary arrays.

Why?

Merge Sort code

```
/**
 * perform a merge sort on the elements of data
 * @param data data != null, all elements of data
 * are the same data type
 */
public static void mergeSort(Comparable[] data) {
	Comparable[] temp = new Comparable[data.length];
	sort(data, temp, 0, data.length - 1);
}

private static void sort(Comparable[] data, Comparable[] temp,
				int low, int high) {
	if(low < high) {
		int center = (low + high) / 2;
		sort(data, temp, low, center);
		sort(data, temp, center + 1, high);
		merge(data, temp, low, center + 1, high);
	}
}
```


Merge Sort Code

```
private static void merge( Comparable[] data, Comparable[] temp,
				int leftPos, int rightPos, int rightEnd) {
		int leftEnd = rightPos - 1;
		int tempPos = leftPos;
		int numElements = rightEnd - leftPos + 1;
		//main loop
		while(leftPos <= leftEnd && rightPos <= rightEnd){
			if( data[leftPos].compareTo(data[rightPos]) <= 0) {
				temp[tempPos] = data[leftPos];
				leftPos++;
			} else {
				temp[tempPos] = data[rightPos];
				rightPos++;
			}
			tempPos++;
		}
		//copy rest of left half
		while (leftPos <= leftEnd) {
			temp[tempPos] = data[leftPos];
			tempPos++;
			leftPos++;
		}
		//copy rest of right half
		while (rightPos <= rightEnd) {
			temp[tempPos] = data[rightPos];
			tempPos++;
			rightPos++;
		}
		//Copy temp back into data
		for (int i = 0; i < numElements; i++, rightEnd--)
			data[rightEnd] = temp[rightEnd];
}
```


Clicker 3

- What are the best case and worst case Orders (Big O) for mergesort?

	Best	Worst
A.	O(NlogN)	$O(N^2)$
B.	$O(N^2)$	$O(N^2)$
C.	$O(N^2)$	O(N!)
D.	O(NlogN)	O(NlogN)
E.	O(N)	O(NlogN)

Clicker 4

- Is mergesort always stable?

A. No

B. Yes

Clicker 5

- You have 1,000,000 distinct items in random order that you will be searching. How many searches need to be performed before the data is changed to make it worthwhile to sort the data before searching?

A. ~40

B. ~100

C. ~500

D. ~2,000

E. ~500,000

Comparison of Various Sorts (2001)

Num Items	Selection	Insertion	Quicksort
1000	0.016	0.005	0 ??
2000	0.059	0.049	0.006
4000	0.271	0.175	0.005
8000	1.056	0.686	0??
16000	4.203	2.754	0.011
32000	16.852	11.039	0.045
64000	expected?	expected?	0.068
128000	expected?	expected?	0.158
256000	expected?	expected?	0.335
512000	expected?	expected?	0.722
1024000	expected?	expected?	1.550

times in seconds

Comparison of Various Sorts (2011)

Num Items	Selection	Insertion	Quicksort	Merge	Arrays.sort
1000	0.002	0.001	-	-	-
2000	0.002	0.001	-	-	-
4000	0.006	0.004	-	-	-
8000	0.022	0.018	-	-	-
16000	0.086	0.070	0.002	0.002	0.002
32000	0.341	0.280	0.004	0.005	0.003
64000	1.352	1.123	0.008	0.010	0.007
128000	5.394	4.499	0.017	0.022	0.015
256000	21.560	18.060	0.035	0.047	0.031
512000	86.083	72.303	0.072	0.099	0.066
1024000	???	???	0.152	0.206	0.138
2048000			0.317	0.434	0.287
4096000			0.663	0.911	0.601
8192000			1.375	1.885	1.246

Comparison of Various Sorts (2020)

Num Items	Selection	Insertion	Quicksort	Mergesort	Arrays. sort(int)	Arrays.sort(Integer)	Arrays. parallelSort
1,000	<0.001	<0.001	-	-	-	-	-
2,000	0.001	<0.001	-	-	-	-	-
4,000	0.004	0.003	-	-	-	-	Speeds
8,000	0.017	0.010	-	-	-	-	up????
16,000	0.065	0.040	0.002	0.002	0.003	0.011	0.007
32,000	0.258	0.160	0.002	0.003	0.002	0.008	0.003
64,000	1.110	0.696	0.005	0.008	0.004	0.011	0.001
128,000	4.172	2.645	0.011	0.015	0.009	0.024	0.002
256,000	16.48	10.76	0.024	0.034	0.018	0.051	0.004
512,000	70.38	47.18	0.049	0.068	0.040	0.114	0.008
1,024,000	-	-	0.098	0.143	0.082	0.259	0.017
2,048,000	-	-	0.205	0.296	0.184	0.637	0.035
4,096,000	-	-	0.450	0.659	0.383	1.452	0.079
8,192,000	-	-	0.941	1.372	0.786	3.354	0.148

Concluding Thoughts

- Language libraries often have sorting algorithms in them
  - Java Arrays and Collections classes
  - C++ Standard Template Library
  - Python sort and sorted functions
- Hybrid sorts
  - when size of unsorted list or portion of array is small use insertion sort, otherwise use O(N log N) sort like Quicksort or Mergesort

Concluding Thoughts

- Sorts still being created!
- Timsort (2002)
  - created for python version 2.3
  - now used in Java version 7.0+
  - takes advantage of real world data
  - real world data is usually partially sorted, not totally random
- Library Sort (2006)
  - Like insertion sort, but leaves gaps for later elements