CMPT 225

Sorting

Sorting Summary

- Comparison Based Sort Algorithms
- Radix Sort

Sorting Algorithm Summary

Algorithm	Best	Average	Worst	Stable	In-place	Space
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$	yes	yes	$O(1)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	no	yes	$O(1)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$	yes	yes	$O(1)$
Merge sort	$O(n*\log_2 n)$	$O(n*\log_2 n)$	$O(n*\log_2 n)$	yes	no	$O(\mathrm{n})$
Quicksort	$O(n*\log_2 n)$	$O(n*\log_2 n)$	$O(n^2)$	no	yes	$O(\log_2 \mathrm{n})$
Heapsort	$O(n*\log_2 n)$	$O(n*\log_2 n)$	$O(n*\log_2 n)$	no	yes	$O(1)$

Stable Sorts

- What is a stable sorting algorithm?
  - One that sorts duplicate values in the same order in which they appear in the input
- Why is this useful
  - Because the original order is to be preserved as much as possible for some reason
  - Or so that data can be sorted on multiple attributes
    - So that the duplicate values of one attribute are ordered by the second attribute

Stable Sort Example

Sort by last, then first name

Last Name	First Name
Smith	Mike
Smith	Sue
Lee	Chris
Lee	Kate
Bard	Sue
Lee	Vera
Lee	Andromeda
Taylor	Chris
Lee	Stan
Smith	Kate
Bard	Kate

Sort by first name

Last Name	First Name
Lee	Andromeda
Taylor	Chris
Lee	Chris
Lee	Kate
Smith	Kate
Bard	Kate
Smith	Mike
Lee	Stan
Smith	Sue
Bard	Sue
Lee	Vera

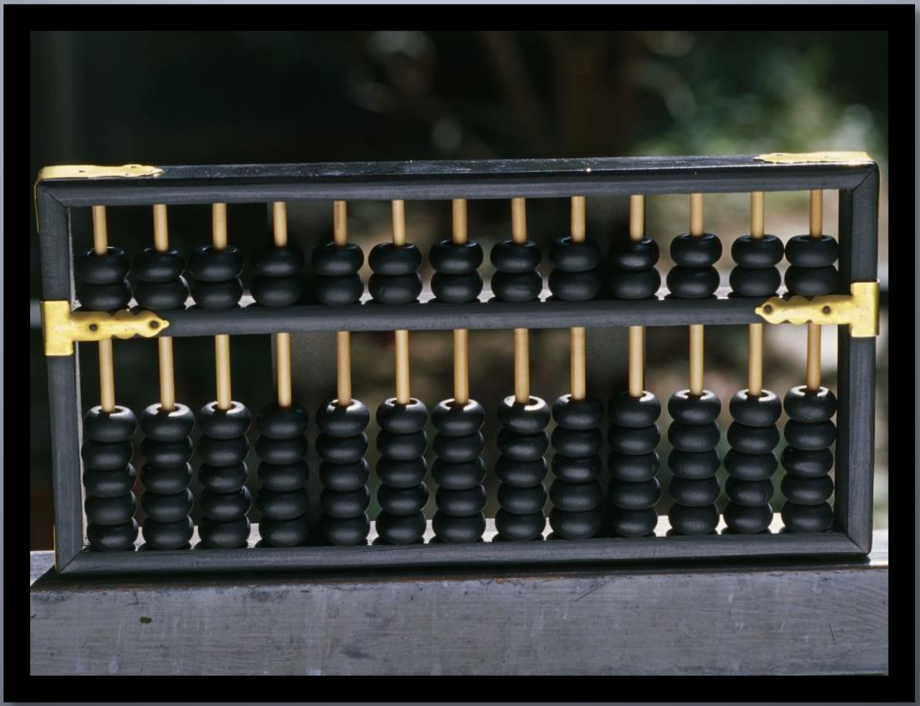
Then by last name using a stable sort

Last Name	First Name
Bard	Kate
Bard	Sue
Lee	Andromeda
Lee	Chris
Lee	Kate
Lee	Stan
Lee	Vera
Smith	Kate
Smith	Mike
Smith	Sue
Taylor	Chris

Comparison Based Sorting

- Comparison based sorting algorithms compares elements of its input with each other
  - All the algorithms we have looked at do this
- The best O Notation running time of comparison based sorts is O(n log n)
  - The proof is beyond the scope of CMPT 225
    - But not CMPT 307
- There are non comparison based sorting algorithms

Counting Sort and Radix Sort



Non Comparison Based Sorts

- The fastest O Notation running time for a comparison-based sort is O(n log n)
  - A comparison-based sorts compare input elements
    - Selection sort compares input element to find the smallest
    - Merge sort compares input elements to merge them
    - ...
  - There are sorting algorithms that do not do this
- We will look at two such sorting algorithms
  - Counting Sort
  - Radix Sort

Counting Sort Description

- A fast sort for sorting an array of elements with non-negative integer keys
  - Assume that *k* is the highest valued key
  - The sort requires an (additional) array of size *k*
- Process
  - Store the count of the number of incidences of each element using its key as the index
  - Modify the array to store the cumulative number of values
  - Write

Counting Sort – First Pass

index	0	1	2	3	4	5	6	7	8
value	4	0	0	1	0	2	4	6	1

Input array, *A*

Create an array, *B*, of size *k*+1, where *k* is the maximum key value

index	0	1	2	3	4	5	6
count	0	0	0	0	0	0	0

Initialization: set each element to 0

Process – traverse input array *A*, incrementing each *B*[*A*[*i*]]

index	0	1	2	3	4	5	6
count	3	2	1	0	2	0	1

Counting Sort – Second Pass

index	0	1	2	3	4	5	6	7	8
value	4	0	0	1	0	2	4	6	1

Input array, *A*

index	0	1	2	3	4	5	6
count	0	0	0	0	0	0	0

Store cumulative sums of the number of elements in B

index	0	1	2	3	4	5	6
count	3	5	6	6	8	8	9

Add to each element (except the first) the value of the previous

Counting Sort – Third Pass

index	0	1	2	3	4	5	6	7	8
value	4	0	0	1	0	2	4	6	1

Input array, *A*

index	0	1	2	3	4	5	6
count	3	5	6	6	8	8	9

Create an output array, *C*, the same size as the input array

index	0	1	2	3	4	5	6	7	8
value	-	-	-	-	-	-	-	-	-

Elements do not need to be initialized

Traverse *A* in reverse, inserting each value in *C* at *C*[*B*[*A*[i]]-1] and

deduct 1 from each value in *B* as it is accessed

Counting Sort – Third Pass Illustrated

index	0	1	2	3	4	5	6	7	8
value	4	0	0	1	0	2	4	6	1

Input array, *A*

index	0	1	2	3	4	5	6
count	3	4	6	6	8	8	9

deduct 1 from *B* [1]

index	0	1	2	3	4	5	6	7	8
value	-	-	-	-	1	-	-	-	-

C[*B*[1]-1]

Repeat for each element of the input array

Counting Sort Notes

- Note the similarity to hash tables
  - More correctly, a *direct-address table*
  - A direct-address table has keys that are used directly as indexes into the table

    Like the *Convenientville* example
    - Without requiring a hash function
- O Notation running time
  - Requires two passes through the original array
  - And one pass through the array storing the frequencies to accumulate the counts

    O(n + k)
  - Original array is size n, frequency array, size k

Radix Sort by Example

- Sort the following integers
  - 329,557,457,226,720,657,449,355,839,510,719,845
  - Note that they all have three digits
- Make three passes through the values
  - For each pass put each value in one of ten *buckets* based on the value of its *i*th digit
  - Start with the right-most digit and end with the left-most digit

Radix Sort – First Pass

Put into buckets (storage) based on the value of the 0^{th} digit counting from the right-most digit

0	1	2	3	4	5	6	7	8	9
720					355	226	557		329
510					845		457		449
							657		839
									719

Sort
329
557
457
226
720
657
449
355
839
510
719
845

Radix Sort – Second Pass

Put into buckets (storage) based on the value of the 1st digit in order from the first set of buckets

0	1	2	3	4	5	6	7	8	9
720					355	226	557		329
510					845		457		449
							657		839
									719

0	1	2	3	4	5	6	7	8	9
	510	720	839	845	355				
	719	226		449	557				
		329			457				
					657				

Sort
329
557
457
226
720
657
449
355
839
510
719
845

Radix Sort – Third Pass

Put into buckets (storage) based on the value of the 2^{nd} digit in order from the second set of buckets

0	1	2	3	4	5	6	7	8	9
	510	720	839	845	355				
	719	226		449	557				
		329			457				
					657				

0	1	2	3	4	5	6	7	8	9
		226	329	449	510	657	719	839	
			355	457	557		720	845	

Sort
329
557
457
226
720
657
449
355
839
510
719
845

Radix Sort Discussion

- Time complexity
  - One pass through the array for each digit of the number
    - Or character in a string
  - If there are *n* values of at most *k* digits
    - O ($n*k$)
  - If *n* is large and *k* relatively small faster than O($n\log n$)
- Space complexity
  - The naïve version presented is very space inefficient
  - Requires 10 * *n* storage for each set of buckets
  - More space efficient (and complex) versions exist

Appendix

No Stable Sort Algorithms

Stable Sort? Selection Sort

Last Name	First Name
Lee	Andromeda
Taylor	Chris
Lee	Chris
Lee	Kate
Smith	Kate
Bard	Kate
Smith	Mike
Lee	Stan
Smith	Sue
Bard	Sue
Lee	Vera

Last Name	First Name
Bard	Kate
Bard	Sue
Lee	Chris
Lee	Kate
Smith	Kate
Lee	Andromeda
Smith	Mike
Lee	Stan
Smith	Sue
Taylor	Chris
Lee	Vera

Last Name	First Name
Bard	Kate
Bard	Sue
Lee	Chris
Lee	Kate
Lee	Andromeda
Smith	Kate
Smith	Mike
Lee	Stan
Smith	Sue
Taylor	Chris
Lee	Vera

Stable Sort? Quick Sort

Last Name	First Name
Lee	Andromeda
Taylor	Chris
Lee	Chris
Lee	Kate
Smith	Kate
Bard	Kate
Smith	Mike
Lee	Stan
Smith	Sue
Bard	Sue
Lee	Vera

Last Name	First Name
Lee	Andromeda
Bard	Sue
Lee	Chris
Lee	Kate
Lee	Stan
Bard	Kate
Smith	Mike
Smith	Kate
Smith	Sue
Taylor	Chris
Lee	Vera

Stable Sort? Heap Sort – Heapify 1

Last Name	First Name
Lee	Andromeda
Taylor	Chris
Lee	Chris
Lee	Kate
Smith	Kate
Bard	Kate
Smith	Mike
Lee	Stan
Smith	Sue
Bard	Sue
Lee	Vera

heapify

index	Last Name	First Name
- : **0** : 1,2	Lee	Andromeda
0 : **1** : 3,4	Taylor	Chris
0 : **2** : 5,6	Smith	Mike
1 : **3** : 7,8	Smith	Sue
1 : **4** : 9,10	Smith	Kate
2 : **5** : -	Bard	Kate
2 : **6** : -	Lee	Chris
3 : **7** : -	Lee	Stan
3 : **8** : -	Lee	Kate
4 : **9** : -	Bard	Sue
4 : **10** : -	Lee	Vera

Stable Sort? Heap Sort – Heapify 2

index	Last Name	First Name
- : **0** : 1,2	Lee	Andromeda
0 : **1** : 3,4	Taylor	Chris
0 : **2** : 5,6	Smith	Mike
1 : **3** : 7,8	Smith	Sue
1 : **4** : 9,10	Smith	Kate
2 : **5** : -	Bard	Kate
2 : **6** : -	Lee	Chris
3 : **7** : -	Lee	Stan
3 : **8** : -	Lee	Kate
4 : **9** : -	Bard	Sue
4 : **10** : -	Lee	Vera

index	Last Name	First Name
- : **0** : 1,2	Taylor	Chris
0 : **1** : 3,4	Smith	Sue
0 : **2** : 5,6	Smith	Mike
1 : **3** : 7,8	Lee	Andromeda
1 : **4** : 9,10	Smith	Kate
2 : **5** : -	Bard	Kate
2 : **6** : -	Lee	Chris
3 : **7** : -	Lee	Stan
3 : **8** : -	Lee	Kate
4 : **9** : -	Bard	Sue
4 : **10** : -	Lee	Vera

Stable Sort? Heap Sort – Remove 1

index	Last Name	First Name
- : **0** : 1,2	Taylor	Chris
0 : **1** : 3,4	Smith	Sue
0 : **2** : 5,6	Smith	Mike
1 : **3** : 7,8	Lee	Andromeda
1 : **4** : 9,10	Smith	Kate
2 : **5** : -	Bard	Kate
2 : **6** : -	Lee	Chris
3 : **7** : -	Lee	Stan
3 : **8** : -	Lee	Kate
4 : **9** : -	Bard	Sue
4 : **10** : -	Lee	Vera

index	Last Name	First Name
- : **0** : 1,2	Smith	Sue
0 : **1** : 3,4	Smith	Kate
0 : **2** : 5,6	Smith	Mike
1 : **3** : 7,8	Lee	Andromeda
1 : **4** : 9	Lee	Vera
2 : **5** : -	Bard	Kate
2 : **6** : -	Lee	Chris
3 : **7** : -	Lee	Stan
3 : **8** : -	Lee	Kate
4 : **9** : -	Bard	Sue
4 : **10** : -	Taylor	Chris

Stable Sort? Heap Sort – Remove 2

index	Last Name	First Name
- : **0** : 1,2	Smith	Sue
0 : **1** : 3,4	Smith	Kate
0 : **2** : 5,6	Smith	Mike
1 : **3** : 7,8	Lee	Andromeda
1 : **4** : 9	Lee	Vera
2 : **5** : -	Bard	Kate
2 : **6** : -	Lee	Chris
3 : **7** : -	Lee	Stan
3 : **8** : -	Lee	Kate
4 : **9** : -	Bard	Sue
4 : **10** : -	Taylor	Chris

index	Last Name	First Name
- : **0** : 1,2	Smith	Kate
0 : **1** : 3,4	Lee	Andromeda
0 : **2** : 5,6	Smith	Mike
1 : **3** : 7,8	Lee	Stan
1 : **4** : -	Lee	Vera
2 : **5** : -	Bard	Kate
2 : **6** : -	Lee	Chris
3 : **7** : -	Bard	Sue
3 : **8** : -	Lee	Kate
4 : **9** : -	Smith	Sue
4 : **10** : -	Taylor	Chris

Stable Sort? Heap Sort – Remove 3

index	Last Name	First Name
- : **0** : 1,2	Smith	Kate
0 : **1** : 3,4	Lee	Andromeda
0 : **2** : 5,6	Smith	Mike
1 : **3** : 7,8	Lee	Stan
1 : **4** : -	Lee	Vera
2 : **5** : -	Bard	Kate
2 : **6** : -	Lee	Chris
3 : **7** : -	Bard	Sue
3 : **8** : -	Lee	Kate
4 : **9** : -	Smith	Sue
4 : **10** : -	Taylor	Chris

index	Last Name	First Name
- : **0** : 1,2	Smith	Mike
0 : **1** : 3,4	Lee	Andromeda
0 : **2** : 5,6	Lee	Kate
1 : **3** : 7	Lee	Stan
1 : **4** : -	Lee	Vera
2 : **5** : -	Bard	Kate
2 : **6** : -	Lee	Chris
3 : **7** : -	Bard	Sue
3 : **8** : -	Smith	Kate
4 : **9** : -	Smith	Sue
4 : **10** : -	Taylor	Chris

Stable Sort? Heap Sort – Remove 4

index	Last Name	First Name
- : **0** : 1,2	Smith	Mike
0 : **1** : 3,4	Lee	Andromeda
0 : **2** : 5,6	Lee	Kate
1 : **3** : 7	Lee	Stan
1 : **4** : -	Lee	Vera
2 : **5** : -	Bard	Kate
2 : **6** : -	Lee	Chris
3 : **7** : -	Bard	Sue
3 : **8** : -	Smith	Kate
4 : **9** : -	Smith	Sue
4 : **10** : -	Taylor	Chris

index	Last Name	First Name
- : **0** : 1,2	Lee	Andromeda
0 : **1** : 3,4	Lee	Stan
0 : **2** : 5,6	Lee	Kate
1 : **3** : -	Bard	Sue
1 : **4** : -	Lee	Vera
2 : **5** : -	Bard	Kate
2 : **6** : -	Lee	Chris
3 : **7** : -	Smith	Mike
3 : **8** : -	Smith	Kate
4 : **9** : -	Smith	Sue
4 : **10** : -	Taylor	Chris