CSIS 3103

Ch 8: Sorting – **Faster!**

Merge Sort

Merge is a common data processing operation performed on two sequences of data where

- The objects in both sequences are ordered according to a common `compareTo` method
- The result is a third sequence containing all the data from the first two sequences arranged in order

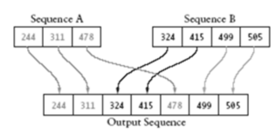
Merge Algorithm

1. Access the first item of both sequences
2. while both sequences have items

   Compare current items from each sequence, copy the smaller item to the output sequence, and access the next item from the sequence whose item was copied
3. Copy any remaining items from the first sequence to the output sequence
4. Copy any remaining items from the second sequence to the output sequence

Analysis of Merge

- Merge time is O(n)
  - For two input sequences with a total of n elements, each element is moved to the output sequence
- Dividing/merging done O(log n) times
- So merge sort is O(n log n)
- Both initial sequences and the output sequence need to be stored
  - The array cannot be merged in place
  - Additional space usage is O(n)

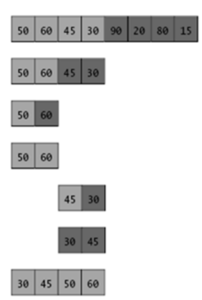


Merge Sort Algorithm

Algorithm for Merge Sort

1. if the `tableSize` is > 1
2. Set `halfSize` to `tableSize` divided by 2.
3. Allocate a table called `leftTable` of size `halfSize`.
4. Allocate a table called `rightTable` of size `tableSize - halfSize`.
5. Copy the elements from `table[0 ... halfSize - 1]` into `leftTable`.
6. Copy the elements from `table[halfSize ... tableSize]` into `rightTable`.
7. Recursively apply the merge sort algorithm to `leftTable`.
8. Recursively apply the merge sort algorithm to `rightTable`.
9. Apply the merge method using `leftTable` and `rightTable` as the input and the original table as the output.

Trace of Merge Sort (left table)



1. *Split array into two 4-element arrays*
2. *Split left array into two 2-element arrays*
3. *Split left array (50, 60) into two 1-element arrays*
4. *Merge two 1-element arrays into a 2-element array*
5. *Split right array from Step 2 into two 1-element arrays*
6. *Merge two 1-element arrays into a 2-element array*
7. *Merge two 2-element arrays into a 4-element array*

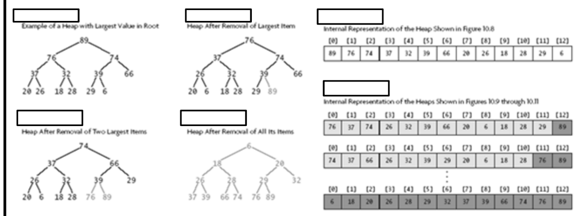
Heapsort

1. Build a heap, arranging the elements in an array
2. While the heap is not empty

   Remove the first item from the heap by swapping it with the last item and restoring the heap property for the remaining items

Heapsort

Heapsort is $O(n \log n)$ but does not require any additional storage



Quicksort

- Developed in 1962
- Rearranges an array into two parts so that
  - all the elements in the left subarray are less than or equal to a specified value, called the *pivot*
  - all the elements in the right subarray are larger than the *pivot*
- Average case for Quicksort is $O(n \log n)$

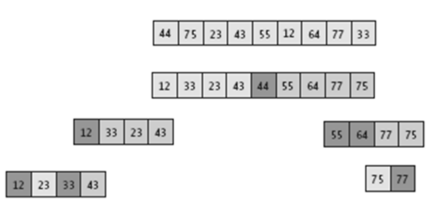
Algorithm for Quicksort

1. **`if first < last then`**
2. **Partition the elements in the subarray `first..last` so that the pivot value is in its correct place (subscript `pivIndex`)**
3. **Recursively apply quicksort to the subarray `first..pivIndex - 1`**
4. **Recursively apply quicksort to the subarray `pivIndex + 1..last`**

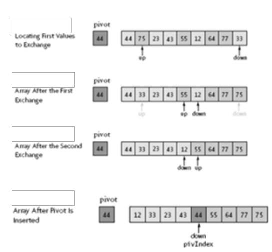
The indexes `first` and `last` are the end points of the array being sorted

The index of the pivot after partitioning is `pivIndex`

Quicksort



Algorithm for Partitioning



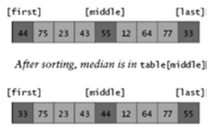
Algorithm for `partition` Method

1. Define the pivot value as the contents of `table[first]`.
2. Initialize `up` to `first` and `down` to `last`.
3. `do`
4. Increment `up` until `up` selects the first element greater than the pivot value or `up` has reached `last`.
5. Decrement `down` until `down` selects the first element less than or equal to the pivot value or `down` has reached `first`.
6. `if up < down then`
7. Exchange `table[up]` and `table[down]`.
8. `while up` is to the left of `down`
9. Exchange `table[first]` and `table[down]`.
10. Return the value of `down` to `pivIndex`.

Revised Partition Algorithm

- Quicksort is $O(n^2)$ when each partitioning yields one empty subarray,
  Occurs when the array is already sorted
- Need to pick a better pivot value
  - Requires three markers
    - First, middle, last
  - Use the median of the these items as the pivot

Sorting First, Middle, and Last Elements in Array



Testing the Sort Algorithms

- Need to use a variety of test cases
  - Small and large arrays
  - Arrays in random order
  - Arrays that are already sorted
  - Arrays with duplicate values
- Compare performance on each type of array

Sorting Algorithm Performance

	Number of Comparisons		
	Best	Average	Worst
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
Shell sort	$O(n^{7/6})$	$O(n^{5/4})$	$O(n^2)$
Merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Heapsort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quicksort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$

Arrays.sort() for objects Collections.sort() (Merge sort)

Arrays.sort() for for primitives (Quicksort)