Sorting Algorithms

- Sorting: A process that organizes a collection of data into either ascending or descending order
- The sort key
  - The part of a data item that we consider when sorting a data collection
  - Typically, the operator < must be defined in the type of key

Categories of sorting algorithms

- An internal sort
  - Requires that the collection of data fit entirely in the computer’s main memory
- An external sort
  - The collection of data will not fit in the computer’s main memory all at once, but must reside in secondary storage

Sorting

Selection Sort

- Strategy
  - Place the largest item in its correct place
  - Place the next largest item in its correct place, and so on

Selection Sort

- Strategy
  - Place the largest item in its correct place
  - Place the next largest item in its correct place, and so on

Initial array:

29	10	14	37	13

Selection Sort

- Strategy
  - Place the largest item in its correct place
  - Place the next largest item in its correct place, and so on

Initial array:	29	10	14	37	13
After 1st swap:	29	10	14	13	**37**

Selection Sort

- Strategy
  - Place the largest item in its correct place
  - Place the next largest item in its correct place, and so on

Initial array:	29	10	14	37	13
After 1st swap:	29	10	14	13	**37**
After 2nd swap:	13	10	14	**29**	**37**

Selection Sort

- Strategy
  - Place the largest item in its correct place
  - Place the next largest item in its correct place, and so on

Initial array:	29	10	14	37	13
After 1st swap:	29	10	14	13	**37**
After 2nd swap:	13	10	14	**29**	**37**
After 3rd swap:	13	10	**14**	**29**	**37**

Selection Sort

- Strategy
  - Place the largest item in its correct place
  - Place the next largest item in its correct place, and so on

Initial array:	29	10	14	37	13
After 1st swap:	29	10	14	13	**37**
After 2nd swap:	13	10	14	**29**	**37**
After 3rd swap:	13	10	**14**	**29**	**37**
After 4th swap:	**10**	**13**	**14**	**29**	**37**

Implementation & Analysis

- Analysis:
  - Worst case: $O(n^2)$
  - Average case: $O(n^2)$
  - Does not depend on the initial arrangement of the data
  - Only appropriate for small n

Bubble Sort

Strategy:

- Compare adjacent elements and exchange them if they are out of order
  - Moves the largest elements to the end of the array
  - Repeating this process eventually sorts the array into ascending order

Bubble Sort

Strategy:

- Compare adjacent elements and exchange them if they are out of order
  - Moves the largest elements to the end of the array
  - Repeating this process eventually sorts the array into ascending order

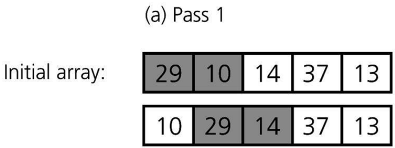
(a) Pass 1

Initial array:	29	10	14	37	13

Bubble Sort

Strategy:

- Compare adjacent elements and exchange them if they are out of order
  - Moves the largest elements to the end of the array
  - Repeating this process eventually sorts the array into ascending order



Bubble Sort

Strategy:

- Compare adjacent elements and exchange them if they are out of order
  - Moves the largest elements to the end of the array
  - Repeating this process eventually sorts the array into ascending order

(a) Pass 1

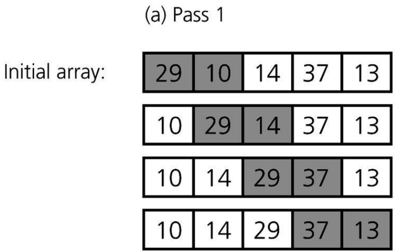
Initial array:

29	10	14	37	13
10	29	14	37	13
10	14	29	37	13

Bubble Sort

Strategy:

- Compare adjacent elements and exchange them if they are out of order
  - Moves the largest elements to the end of the array
  - Repeating this process eventually sorts the array into ascending order

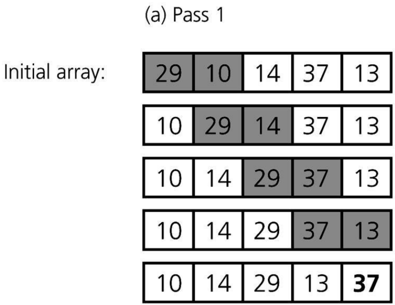


Bubble Sort

Algorithm Efficiency
Measuring the Efficiency of Algorithms
Algorithm Growth Rates
Keeping Your Perspective
Recursion?
Examples
Iteration vs. Recursion
Towers of Hanoi
Summary
Sorting Algorithms
Selection Sort
Bubble Sort
Insertion Sort
Mergesort
Quick Sort
Radix Sort
Summary
STL Sorting Algorithms

Strategy:

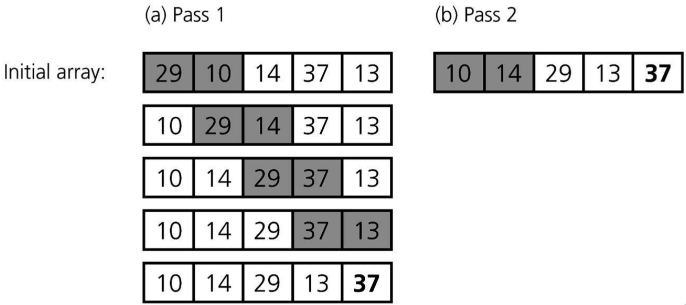
- Compare adjacent elements and exchange them if they are out of order
  - Moves the largest elements to the end of the array
  - Repeating this process eventually sorts the array into ascending order



Bubble Sort

Strategy:

- Compare adjacent elements and exchange them if they are out of order
  - Moves the largest elements to the end of the array
  - Repeating this process eventually sorts the array into ascending order



Bubble Sort

Strategy:

- Compare adjacent elements and exchange them if they are out of order
  - Moves the largest elements to the end of the array
  - Repeating this process eventually sorts the array into ascending order

(a) Pass 1

Initial array:

29	10	14	37	13
10	29	14	37	13
10	14	29	37	13
10	14	29	37	13
10	14	29	13	**37**

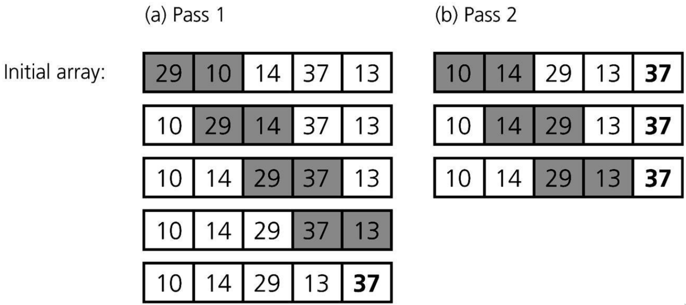
(b) Pass 2

10	14	29	13	**37**
10	14	29	13	**37**

Bubble Sort

Strategy:

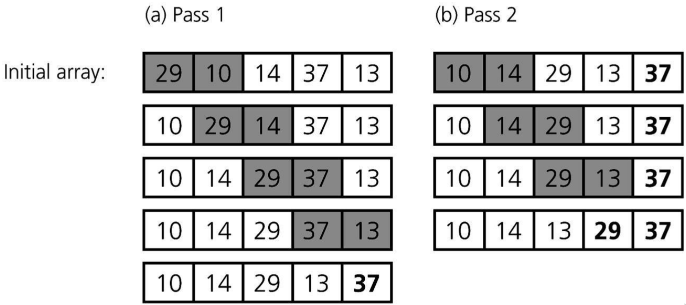
- Compare adjacent elements and exchange them if they are out of order
  - Moves the largest elements to the end of the array
  - Repeating this process eventually sorts the array into ascending order



Bubble Sort

Strategy:

- Compare adjacent elements and exchange them if they are out of order
  - Moves the largest elements to the end of the array
  - Repeating this process eventually sorts the array into ascending order



Implementation & Analysis

- Analysis:
  - Worst case: $O(n^2)$
  - Average case: $O(n^2)$
  - Best case: $O(n)$

Algorithm Analysis, Recursion, and Sorting

CSCI 3110

Algorithm Efficiency
Measuring the Efficiency of Algorithms
Algorithm Growth Rates
Keeping Your Perspective
Recursion?
Examples
Iteration vs. Recursion
Towers of Hanoi
Summary
Sorting Algorithms
Selection Sort
Bubble Sort
Insertion Sort
Mergesort
Quick Sort
Radix Sort
Summary
STL Sorting Algorithms

Insertion Sort

Strategy:

- Partition the array into two regions: sorted and unsorted
  - Take each item from the unsorted region and insert it into its correct order in the sorted region

Insertion Sort

Strategy:

- Partition the array into two regions: sorted and unsorted
  - Take each item from the unsorted region and insert it into its correct order in the sorted region

Initial array:

29	10	14	37	13

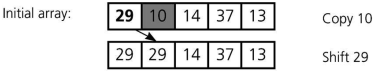
Copy 10

elf

Insertion Sort

Strategy:

- Partition the array into two regions: sorted and unsorted
    - Take each item from the unsorted region and insert it into its correct order in the sorted region

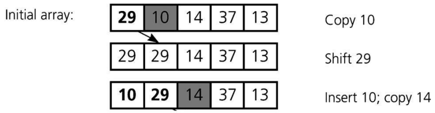


tself

Insertion Sort

Strategy:

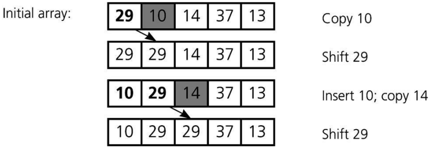
- Partition the array into two regions: sorted and unsorted
  - Take each item from the unsorted region and insert it into its correct order in the sorted region



Insertion Sort

Strategy:

- Partition the array into two regions: sorted and unsorted
  - Take each item from the unsorted region and insert it into its correct order in the sorted region



If

Insertion Sort

Strategy:

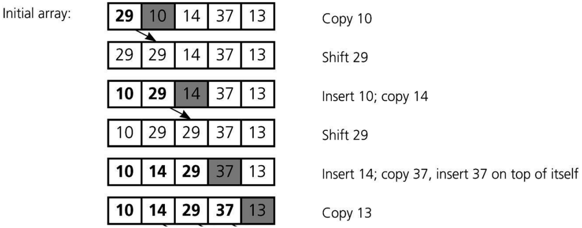
- Partition the array into two regions: sorted and unsorted
  - Take each item from the unsorted region and insert it into its correct order in the sorted region

Initial array:	**29**	10	14	37	13	Copy 10
	29	29	14	37	13	Shift 29
	10	**29**	14	37	13	Insert 10; copy 14
	10	29	29	37	13	Shift 29
	10	**14**	**29**	37	13	Insert 14; copy 37, insert 37 on top of itself

Insertion Sort

Strategy:

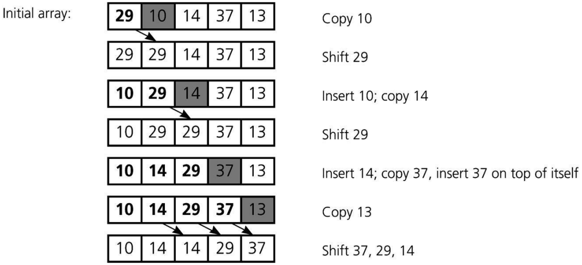
- Partition the array into two regions: sorted and unsorted
  - Take each item from the unsorted region and insert it into its correct order in the sorted region



Insertion Sort

Strategy:

- Partition the array into two regions: sorted and unsorted
  - Take each item from the unsorted region and insert it into its correct order in the sorted region



Insertion Sort

Strategy:

- Partition the array into two regions: sorted and unsorted
  - Take each item from the unsorted region and insert it into its correct order in the sorted region

Initial array:	**29**	10	14	37	13	Copy 10
	29	29	14	37	13	Shift 29
	10	**29**	14	37	13	Insert 10; copy 14
	10	29	29	37	13	Shift 29
	10	**14**	**29**	37	13	Insert 14; copy 37, insert 37 on top of itself
	10	**14**	**29**	**37**	13	Copy 13
	10	14	14	29	37	Shift 37, 29, 14
Sorted array:	**10**	**13**	**14**	**29**	**37**	Insert 13

Algorithm Efficiency
Measuring the Efficiency of Algorithms
Algorithm Growth Rates
Keeping Your Perspective
Recursion?
Examples
Iteration vs. Recursion
Towers of Hanoi
Summary
Sorting Algorithms
Selection Sort
Bubble Sort
Insertion Sort
Mergesort
Quick Sort
Radix Sort
Summary
STL Sorting Algorithms

Implementation & Analysis

- Analysis:
  - Worst case: $O(n^2)$
  - Average case: $O(n^2)$
  - Best case: $O(n)$

Algorithm Analysis, Recursion, and Sorting

CSCI 3110

Algorithm Efficiency
Measuring the Efficiency of Algorithms
Algorithm Growth Rates
Keeping Your Perspective
Recursion?
Examples
Iteration vs. Recursion
Towers of Hanoi
Summary
Sorting Algorithms
Selection Sort
Bubble Sort
Insertion Sort
Mergesort
Quick Sort
Radix Sort
Summary
STL Sorting Algorithms

Merge Sort

Strategy (Divide-and-conquer):

- Divide an array into halves
- Sort each half
- Merge the sorted halves into one sorted array

Merge Sort

Strategy (Divide-and-conquer):

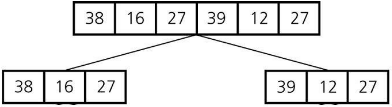
- Divide an array into halves
- Sort each half
- Merge the sorted halves into one sorted array

38	16	27	39	12	27

Merge Sort

Strategy (Divide-and-conquer):

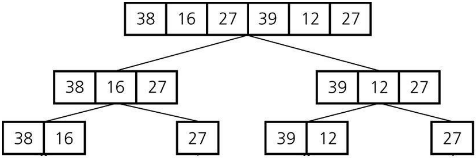
- Divide an array into halves
- Sort each half
- Merge the sorted halves into one sorted array



Merge Sort

Strategy (Divide-and-conquer):

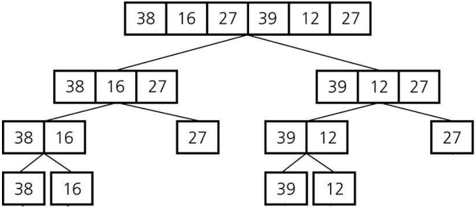
- Divide an array into halves
- Sort each half
- Merge the sorted halves into one sorted array



Merge Sort

Strategy (Divide-and-conquer):

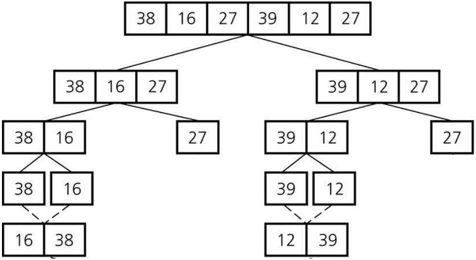
- Divide an array into halves
- Sort each half
- Merge the sorted halves into one sorted array



Merge Sort

Strategy (Divide-and-conquer):

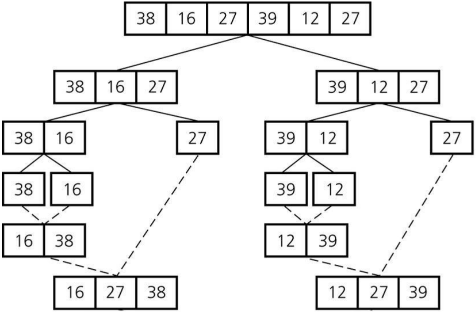
- Divide an array into halves
- Sort each half
- Merge the sorted halves into one sorted array



Merge Sort

Strategy (Divide-and-conquer):

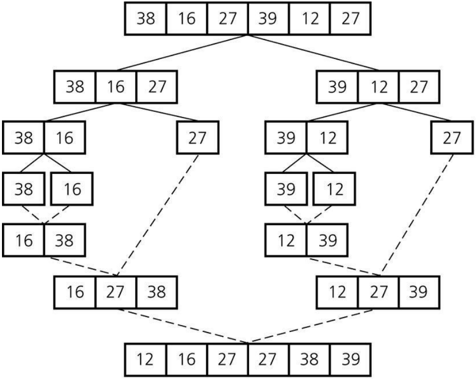
- Divide an array into halves
- Sort each half
- Merge the sorted halves into one sorted array



Merge Sort

Strategy (Divide-and-conquer):

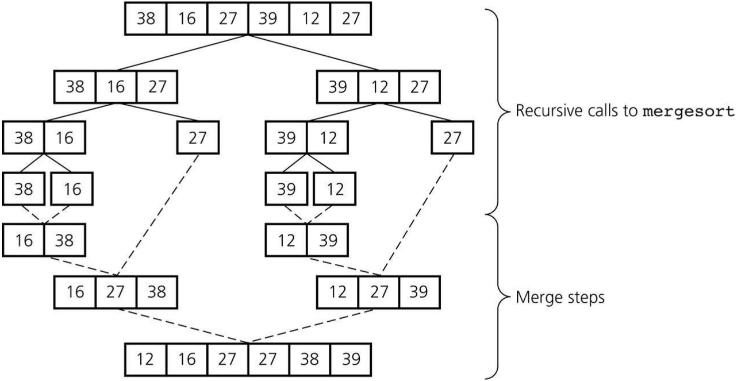
- Divide an array into halves
- Sort each half
- Merge the sorted halves into one sorted array



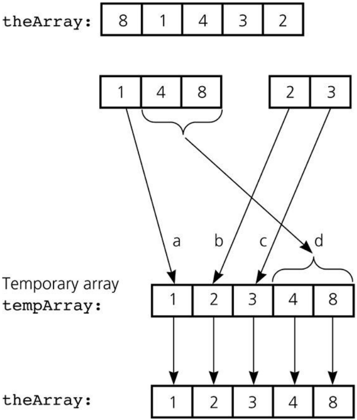
Merge Sort

Strategy (Divide-and-conquer):

- Divide an array into halves
- Sort each half
- Merge the sorted halves into one sorted array



How to Merge Two Arrays



Divide the array in half

Sort the halves

Merge the halves:

a. 1 < 2, so move 1 from left half to `tempArray`
b. 4 > 2, so move 2 from right half to `tempArray`
c. 4 > 3, so move 3 from right half to `tempArray`
d. Right half is finished, so move rest of left half to `tempArray`

Copy temporary array back into original array

Implementation & Analysis

- Analysis:
  - Worst case: $O(n * \log_2 n)$
  - Average case: $O(n * \log_2 n)$
  - Performance is independent of the initial order of the array items
- Advantage:
  - Mergesort is an extremely fast algorithm
- Disadvantage:
  - Mergesort requires a second array as large as the original array

Quick Sort

Strategy (Divide-and-conquer):

- Choose a pivot
- Partition the array about the pivot
  - left section: items < pivot
  - right section: items > pivot
  - Pivot is now in correct sorted position
- Sort the left and the right section recursively

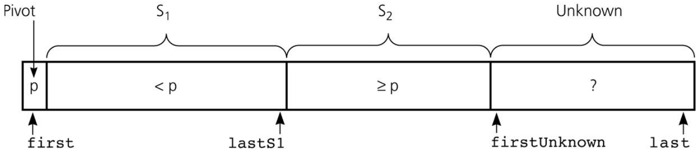
Quick Sort

Strategy (Divide-and-conquer):

- Choose a pivot
- Partition the array about the pivot
  - left section: items < pivot
  - right section: items > pivot
  - Pivot is now in correct sorted position
- Sort the left and the right section recursively

How to partition an array:

- Invariant: The items in the region S_1 are all less than the pivot, and those in S_2 are all greater than or equal to the pivot.



Worst Case Partition

Worst Case Partition

Original array:	5	6	7	8	9

Worst Case Partition

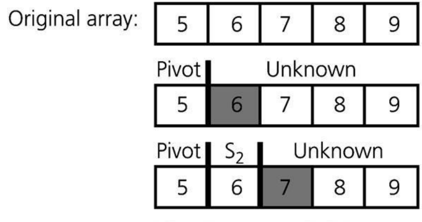
Original array:

5	6	7	8	9

Pivot | Unknown

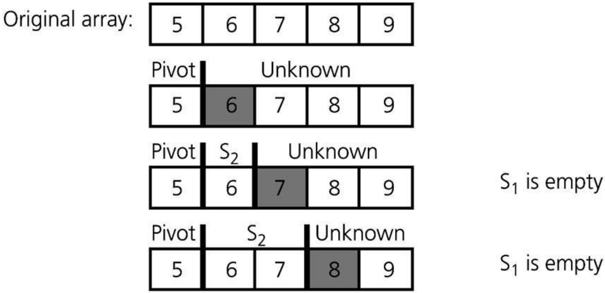
5	6	7	8	9

Worst Case Partition

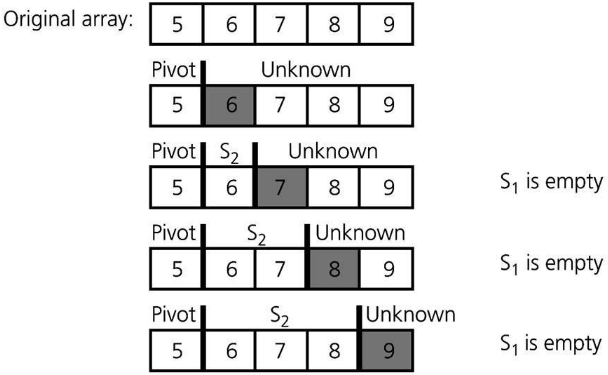


S_1 is empty

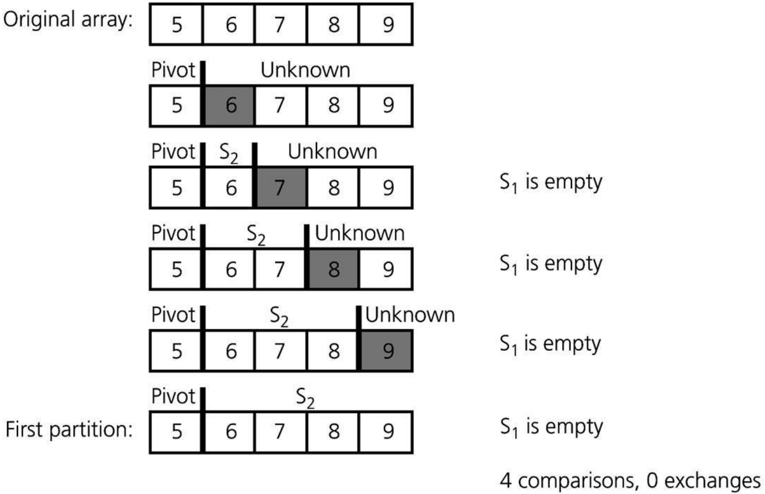
Worst Case Partition



Worst Case Partition



Worst Case Partition



Implementation & Analysis

- Analysis:
  - Average case: $O(n * \log_2 n)$
  - Worst case: $O(n^2)$
    - When the array is already sorted and the smallest item is chosen as the pivot
  - Quicksort is usually extremely fast in practice
  - Even if the worst case occurs, quicksort's performance is acceptable for moderately large arrays

MIDDLE TENNESSEE STATE UNIVERSITY

Algorithm Analysis, Recursion, and Sorting

CSCI 3110

Radix Sort

Strategy (Divide-and-conquer):

- Treats each data element as a character string
- Repeatedly organizes the data into groups according to the i^{th} character in each element

Analysis:

- Average case: $O(n)$

Radix Sort

Radix Sort

0123, 2154, 0222, 0004, 0283, 1560, 1061, 2150 Original integers

Radix Sort

0123, 2154, 0222, 0004, 0283, 1560, 1061, 2150	Original integers
(156**0**, 215**0**) (106**1**) (022**2**) (012**3**, 028**3**) (215**4**, 000**4**)	Grouped by fourth digit

Radix Sort

0123, 2154, 0222, 0004, 0283, 1560, 1061, 2150	Original integers
(156**0**, 215**0**) (106**1**) (022**2**) (012**3**, 028**3**) (215**4**, 000**4**)	Grouped by fourth digit
1560, 2150, 1061, 0222, 0123, 0283, 2154, 0004	Combined

Radix Sort

0123, 2154, 0222, 0004, 0283, 1560, 1061, 2150	Original integers
(156**0**, 215**0**) (106**1**) (022**2**) (012**3**, 028**3**) (215**4**, 000**4**)	Grouped by fourth digit
1560, 2150, 1061, 0222, 0123, 0283, 2154, 0004	Combined
(00**0**4) (02**2**2, 01**2**3) (21**5**0, 21**5**4) (15**6**0, 10**6**1) (02**8**3)	Grouped by third digit

Radix Sort

0123, 2154, 0222, 0004, 0283, 1560, 1061, 2150	Original integers
(156**0**, 215**0**) (106**1**) (022**2**) (012**3**, 028**3**) (215**4**, 000**4**)	Grouped by fourth digit
1560, 2150, 1061, 0222, 0123, 0283, 2154, 0004	Combined
(00**0**4) (02**2**2, 01**2**3) (21**5**0, 21**5**4) (15**6**0, 10**6**1) (02**8**3)	Grouped by third digit
0004, 0222, 0123, 2150, 2154, 1560, 1061, 0283	Combined

Radix Sort

0123, 2154, 0222, 0004, 0283, 1560, 1061, 2150	Original integers
(156**0**, 215**0**) (106**1**) (022**2**) (012**3**, 028**3**) (215**4**, 000**4**)	Grouped by fourth digit
1560, 2150, 1061, 0222, 0123, 0283, 2154, 0004	Combined
(00**0**4) (02**2**2, 01**2**3) (21**5**0, 21**5**4) (15**6**0, 10**6**1) (02**8**3)	Grouped by third digit
0004, 0222, 0123, 2150, 2154, 1560, 1061, 0283	Combined
(0**0**04, 1**0**61) (0**1**23, 2**1**50, 2**1**54) (0**2**22, 0**2**83) (1**5**60)	Grouped by second digit

Radix Sort

0123, 2154, 0222, 0004, 0283, 1560, 1061, 2150	Original integers
(156**0**, 215**0**) (106**1**) (022**2**) (012**3**, 028**3**) (215**4**, 000**4**)	Grouped by fourth digit
1560, 2150, 1061, 0222, 0123, 0283, 2154, 0004	Combined
(00**0**4) (02**2**2, 01**2**3) (21**5**0, 21**5**4) (15**6**0, 10**6**1) (02**8**3)	Grouped by third digit
0004, 0222, 0123, 2150, 2154, 1560, 1061, 0283	Combined
(0**0**04, 1**0**61) (0**1**23, 2**1**50, 2**1**54) (0**2**22, 0**2**83) (1**5**60)	Grouped by second digit
0004, 1061, 0123, 2150, 2154, 0222, 0283, 1560	Combined

Radix Sort

0123, 2154, 0222, 0004, 0283, 1560, 1061, 2150	Original integers
(156**0**, 215**0**) (106**1**) (022**2**) (012**3**, 028**3**) (215**4**, 000**4**)	Grouped by fourth digit
1560, 2150, 1061, 0222, 0123, 0283, 2154, 0004	Combined
(00**0**4) (02**2**2, 01**2**3) (21**5**0, 21**5**4) (15**6**0, 10**6**1) (02**8**3)	Grouped by third digit
0004, 0222, 0123, 2150, 2154, 1560, 1061, 0283	Combined
(0**0**04, 1**0**61) (0**1**23, 2**1**50, 2**1**54) (0**2**22, 0**2**83) (1**5**60)	Grouped by second digit
0004, 1061, 0123, 2150, 2154, 0222, 0283, 1560	Combined
(**0**004, **0**123, **0**222, **0**283) (**1**061, **1**560) (**2**150, **2**154)	Grouped by first digit

Algorithm Analysis, Recursion, and Sorting

CSCI 3110

Sorting Algorithm Comparison

	Worst case	Average case
Selection sort	n^2	n^2
Bubble sort	n^2	n^2
Insertion sort	n^2	n^2
Mergesort	$n * \log n$	$n * \log n$
Quicksort	n^2	$n * \log n$
Radix sort	n	n
Treesort rt	n^2	$n * \log n$
	$n * \log n$	$n * \log n$

Click here to get a straightforward feeling of the difference between $O(n^2)$ and $O(n * \log_2 n)$.

Algorithm Efficiency
Measuring the Efficiency of Algorithms
Algorithm Growth Rates
Keeping Your Perspective
Recursion?
Examples
Iteration vs. Recursion
Towers of Hanoi
Summary
Sorting Algorithms
Selection Sort
Bubble Sort
Insertion Sort
Mergesort
Quick Sort
Radix Sort
Summary
STL Sorting Algorithms

STL Sorting Algorithms

Some sort functions in the STL library header *(algorithm)*:

- *sort*
  - Sorts a range of elements in ascending order by default
- *stable_sort*
  - Sorts as above, but preserves original ordering of equivalent elements
- *partial_sort*
  - Sorts a range of elements and places them at the beginning of the range
- *nth_element*
  - Partitions the elements of a range about the n^{th} element
  - The two subranges are not sorted
- *partition*
  - Partitions the elements of a range according to a given predicate