CSC 225 - Summer 2019

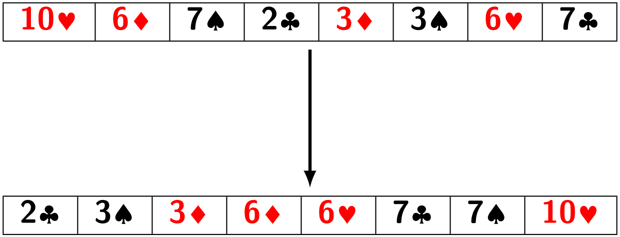
Sorting II

Bill Bird

Department of Computer Science
University of Victoria

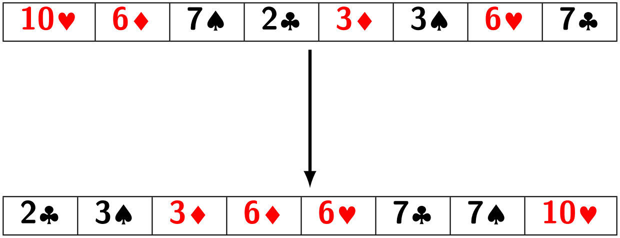
June 4, 2019

Stable Sorting (1)



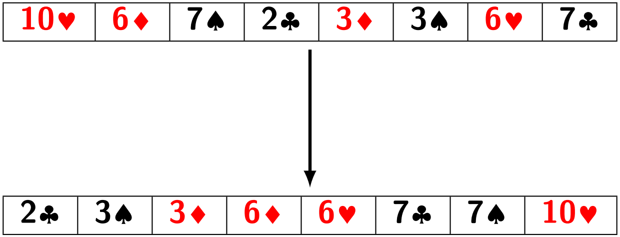
- (Example transcribed from Wikipedia; possibly originally from a book)

Stable Sorting (2)



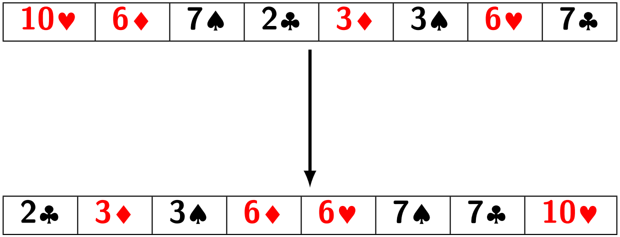
- Consider the problem of sorting a set of playing cards by number.
- The array on the bottom is a valid sorted representation.

Stable Sorting (3)



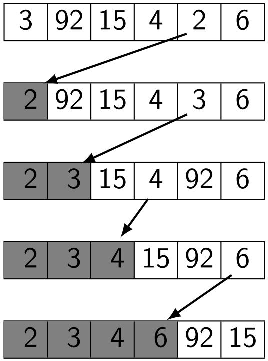
- Since the suits (♣, ♠, ♥ and ♦) are irrelevant to the sorting, it is acceptable for **7♠** to appear on either side of **7♣** in the sorted order.

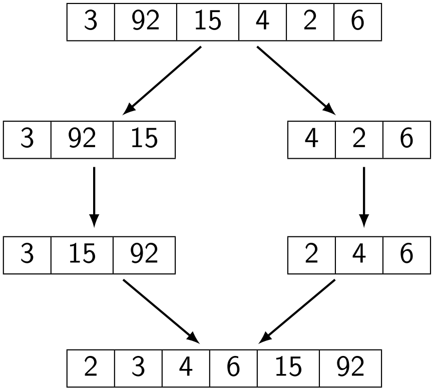
Stable Sorting (4)



- However, it is sometimes desirable for two elements which are equal to remain in the same order in the output array as they appeared in the input array.
- A sorting algorithm is **stable** if it guarantees that the order of equal elements is preserved.

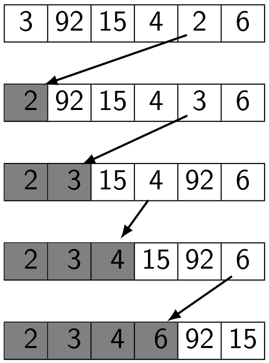
In-place algorithms (1)

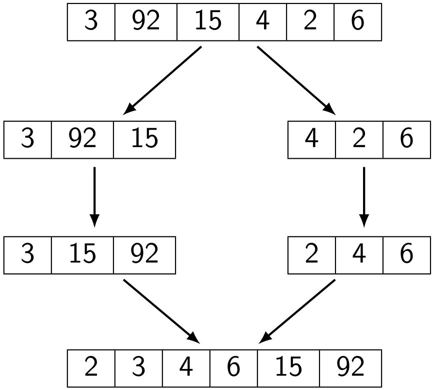




- Some sorting algorithms do all of their work ‘inside’ the input array.
- Selection sort (left) does not allocate any extra arrays.

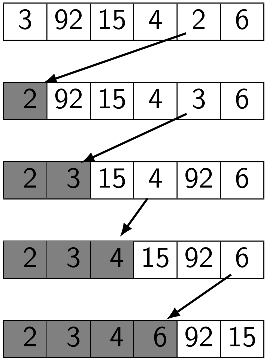
In-place algorithms (2)

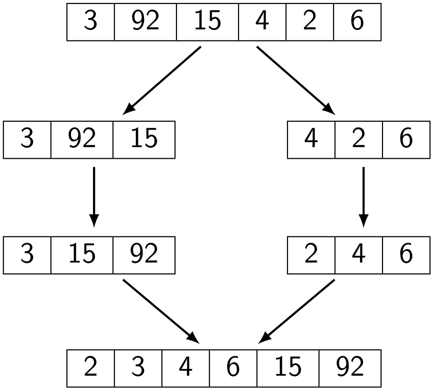




- Merge sort (right) allocates new arrays to facilitate recursive sorting and merging.

In-place algorithms (3)





- A sorting algorithm is said to be **in-place** if it requires $O(1)$ extra storage space.

Sorting Algorithms

	Running Time			Extra	Stable?
	Best Case	Expected Case	Worst Case	Space	
Selection Based					
Heap Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(1)$	No
Insertion Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	Yes
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	No
Divide and Conquer					
Merge Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$	Yes
Quicksort	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n)$	Yes
Other					
Bubble Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	Yes
Radix Sort[1]	$\Theta(dn + b)$	$\Theta(dn + b)$	$\Theta(dn + b)$	$\Theta(n + b)$	Yes

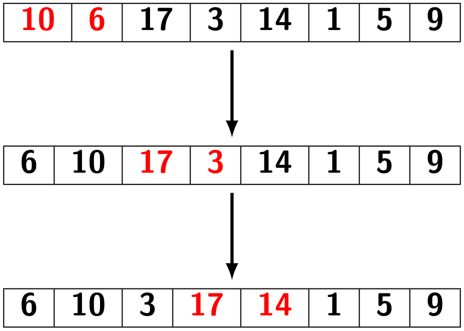
[1] Integers only: d-digit values in base b

Sorting Algorithms

	Running Time			Extra	Stable?
	Best Case	Expected Case	Worst Case	Space	
Selection Based					
Heap Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(1)$	No
Insertion Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	Yes
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	No
Divide and Conquer					
Merge Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$	Yes
Quicksort	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n)$	Yes
Other					
Bubble Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	Yes
Radix Sort[1]	$\Theta(dn + b)$	$\Theta(dn + b)$	$\Theta(dn + b)$	$\Theta(n + b)$	Yes

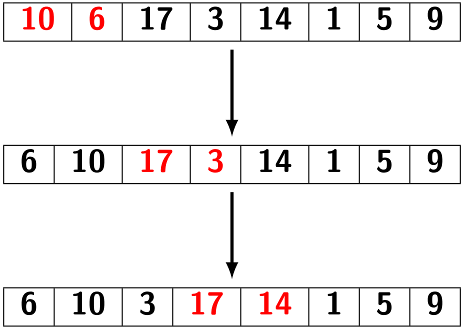
[1] Integers only: d-digit values in base b

Bubble Sort (1)



- **Observation**: If an array is not sorted, there must be a pair of neighbouring elements which are in the wrong order.
- Exchanging this pair seems to make the array ‘more sorted’.

Bubble Sort (2)



Algorithm Idea: Repeatedly scan through the array and swap any out-of-order pairs. When no out-of-order pairs are found in an entire scan, the array must be sorted.

This is the basis for Bubble Sort.

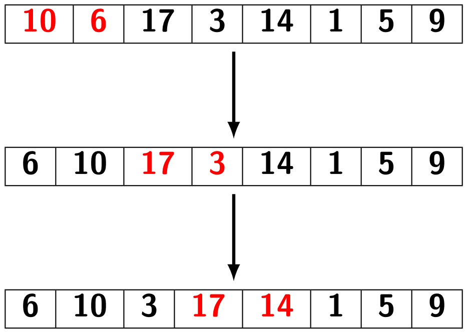
“ In short, the bubble sort seems

to have nothing to recommend it,

except a catchy name... ”

- D. Knuth
The Art of Computer Programming, Volume 3

Bubble Sort (1)



- **Editorial Opinion**: Bubble Sort has no redeeming qualities whatsoever, even the name.
- **Especially the name.**

Bubble Sort (2)

```
 1: procedure BubbleSort($A$)
 2:     $n \leftarrow$ Length($A$)
 3:     swap_made $\leftarrow$ true
 4:     while swap_made $=$ true do
 5:         swap_made $\leftarrow$ false
 6:         for $i \leftarrow 0, 1, \dots, n-2$ do
 7:             if $A[i] > A[i+1]$ then
 8:                 Swap $A[i]$ and $A[i+1]$.
 9:                 swap_made $\leftarrow$ true
10:             end if
11:         end for
12:     end while
13: end procedure
```

The inner loop (lines 6 - 11) is clearly $\Theta(n)$, but the number of iterations of the outer loop varies depending on the number of swaps needed.

Bubble Sort (3)

```
 1: procedure $\textsc{BubbleSort}(A)$
 2:     $n \leftarrow \textsc{Length}(A)$
 3:     swap_made $\leftarrow$ true
 4:     while swap_made $=$ true do
 5:         swap_made $\leftarrow$ false
 6:         for $i \leftarrow 0, 1, \ldots, n-2$ do
 7:             if $A[i] > A[i+1]$ then
 8:                 Swap $A[i]$ and $A[i+1]$.
 9:                 swap_made $\leftarrow$ true
10:             end if
11:         end for
12:     end while
13: end procedure
```

Any out-of-place element will be moved at least once by each iteration of the outer loop, so the total number of iterations of the outer loop can be bounded above by the maximum distance that a single element must travel.

Bubble Sort (4)

```
procedure BubbleSort($A$)
    $n \leftarrow$ Length($A$)
    swap_made $\leftarrow$ true
    while swap_made $=$ true do
        swap_made $\leftarrow$ false
        for $i \leftarrow 0, 1, \ldots, n-2$ do
            if $A[i] > A[i+1]$ then
                Swap $A[i]$ and $A[i+1]$.
                swap_made $\leftarrow$ true
            end if
        end for
    end while
end procedure
```

Therefore, the outer loop may iterate at most $n - 1$ times, so the algorithm is in $O(n^2)$.

Bubble Sort (5)

```
procedure BubbleSort($A$)
    $n \leftarrow$ Length($A$)
    swap_made $\leftarrow$ true
    while swap_made $=$ true do
        swap_made $\leftarrow$ false
        for $i \leftarrow 0, 1, \ldots, n-2$ do
            if $A[i] > A[i+1]$ then
                Swap $A[i]$ and $A[i+1]$.
                swap_made $\leftarrow$ true
            end if
        end for
    end while
end procedure
```

Exercise: Show that there is input for any *n* such that the algorithm requires $\Theta(n^2)$ operations.

Bubble Sort (6)

```
procedure BubbleSort($A$)
    $n \leftarrow$ Length($A$)
    for $j \leftarrow 0, 1, \ldots, n-1$ do
        for $i \leftarrow 0, 1, \ldots, n-2$ do
            if $A[i] > A[i+1]$ then
                Swap $A[i]$ and $A[i+1]$.
            end if
        end for
    end for
end procedure
```

Since at most *n* passes are needed, we can also write the algorithm with nested `for` loops.

“ In short, 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. ”

- D. Knuth

The Art of Computer Programming, Volume 3

Bubble Sort (7)

Exercise 1: Prove that Bubble Sort requires $\Theta(n^2)$ operations on the array

$$A = [2, \quad 3, \quad \ldots, \quad n-2, \quad n-1, \quad n, \quad 1]$$

Exercise 2: Prove that Bubble Sort requires $\Theta(n)$ operations on the array

$$A = [n, \quad 1, \quad 2, \quad 3, \quad \ldots, \quad n-2, \quad n-1]$$

Sorting Algorithms

	Running Time			Extra	Stable?
	Best Case	**Expected Case**	**Worst Case**	**Space**	
Selection Based					
Heap Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(1)$	No
Insertion Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	Yes
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	No
Divide and Conquer					
Merge Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$	Yes
Quicksort	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n)$	Yes
Other					
Bubble Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	Yes
Radix Sort[1]	$\Theta(dn + b)$	$\Theta(dn + b)$	$\Theta(dn + b)$	$\Theta(n + b)$	Yes

[1] Integers only: d-digit values in base b

Merge sort vs. Quicksort (1)

Merge sort is a divide and conquer algorithm with three phases.

Split: (simple)
Divide the array into two (roughly) equally sized parts.
Recurse:
Sort the two subarrays.
Merge: (complicated)
Combine the two sorted subarrays into a single sorted array.

Merge sort vs. Quicksort (2)

Observation: In merge sort, the merge step is complicated because the two sorted arrays must be shuffled together.

Observation: If A_1 and A_2 are sorted arrays where everything in A_1 is less than everything in A_2, then the two arrays can be combined into a sorted array by concatenation (i.e. joining them together).

Merge sort vs. Quicksort (3)

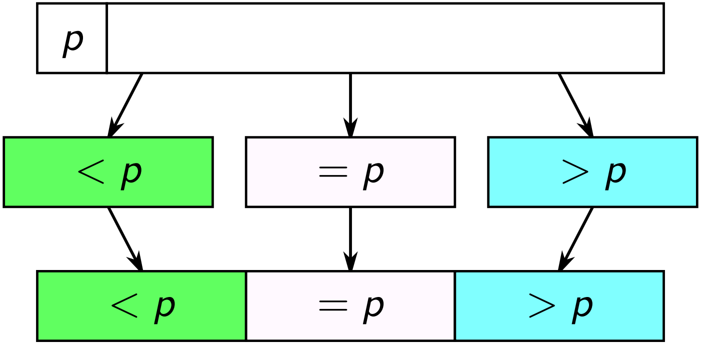
Quicksort is a divide and conquer algorithm with three phases.

Split: (complicated)
Choose some **pivot** value p and split the elements of the input array A into a subarray L containing elements less than p and a subarray G containing elements greater than p.
Recurse:
Sort the two subarrays L and G.
Merge: (simple)
Concatenate the sorted L with the element p and the sorted G.

Quicksort (1)

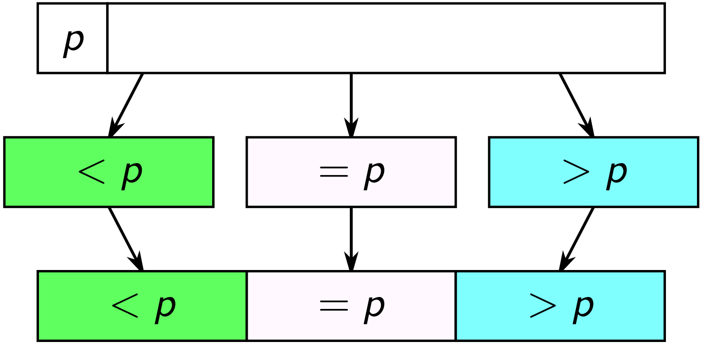
```
 1: procedure QUICKSORT($S$)
 2:     $n \leftarrow$ LENGTH($S$)
 3:     if $n = 0$ or $n = 1$ then
 4:         //A sequence of size 0 or 1 is already sorted.
 5:         return
 6:     end if
 7:     $p \leftarrow$ A pivot value from $S$
 8:     Create empty lists $L$, $E$ and $G$
 9:     for each element $x$ in $S$ do
10:         if $x < p$ then
11:             Add $x$ to the end of $L$
12:         else if $x > p$ then
13:             Add $x$ to the end of $G$
14:         else
15:             Add $x$ to the end of $E$
16:         end if
17:     end for
18:     QUICKSORT($L$)
19:     QUICKSORT($G$)
20:     $S \leftarrow$ Concatenation of $L$, $E$ and $G$
21: end procedure
```

Quicksort (2)



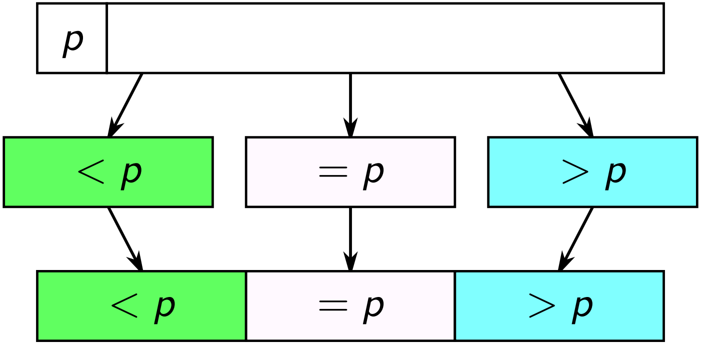
- At each step, the array is partitioned around a pivot value p into sequences L (elements less than p), E (elements equal to p) and G (elements greater than p)

Quicksort (3)



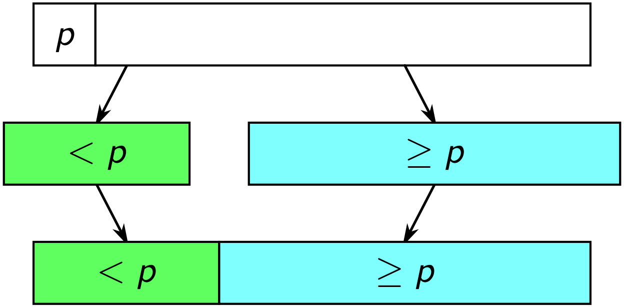
- The pivot is chosen based on some *pivoting rule*.
- A common pivoting rule is to use the first element as a pivot.

Quicksort (4)



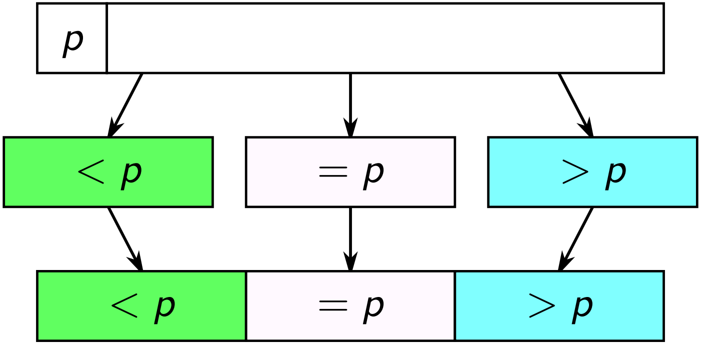
- Other common pivoting rules:
  - Choose the middle element.
  - Randomly select an element.

Quicksort (5)



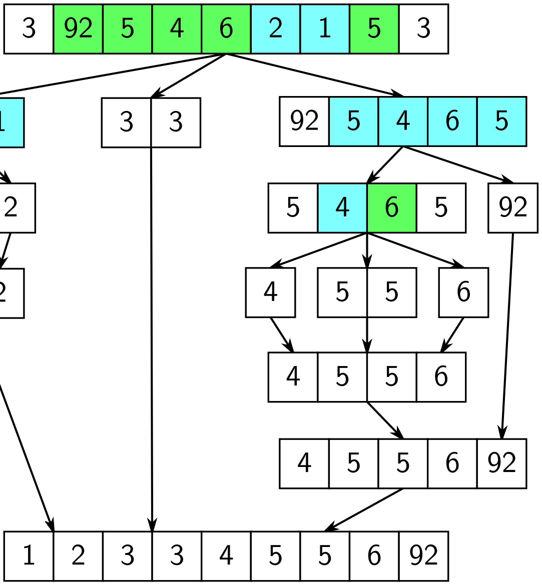
- Some variants of the algorithm do not use a sequence E (and instead put all elements equal to p in either L or G).
- This does not affect the worst case performance, but does increase the best case running time.

Quicksort (6)



- The behaviour of Quicksort is often represented with a diagram called a ‘Quicksort tree’, similar to the one on the next slide (which uses the first element as the pivot).

Quicksort (7)



3 92 5 4 6 2 1 5 3
2 1
3 3
92 5 4 6 5
1
2
5 4 6 5
92
1 2
4
5 5
6
4 5 5 6
4 5 5 6 92
1 2 3 3 4 5 5 6 92

Quicksort Analysis (1)

Splitting the input array into L, E and G requires $\Theta(n)$ operations. Joining the three arrays together after recursion also requires $\Theta(n)$ operations.

The running time of the recursive calls depends on the sizes of L and G. If L has size n_L and G has size n_G, then the following recurrence models the running time of Quicksort.

$$\begin{aligned} T(n) &= 1 && \text{if } n = 1 \\ &= T(n_L) + T(n_G) + 2n + 1 && \text{if } n \geq 2 \end{aligned}$$

Question: What are the maximum sizes of L and G?

Quicksort Analysis (2)

Question: What are the maximum sizes of L and G?

The size of L can be as small as 0 (if the pivot is the smallest element of the array) and as large as $n - 1$ (if the pivot is the largest element of the array).

In the worst case, the pivot at *every* step results in L having the maximum size of $n - 1$, so the worst case running time of Quicksort is modelled by

$$\begin{aligned} T(n) &= 1 && \text{if } n = 1 \\ &= T(n-1) + 2n + 1 && \text{if } n \geq 2 \end{aligned}$$

Quicksort Analysis (3)

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + 2n + 1 & \text{if } n \geq 2 \end{cases}$$

Solution by repeated substitution:

$$\begin{aligned}
T(n) &= T(n-1) + 2n + 1 \\
&= [T(n-2) + 2(n-1) + 1] + 2n + 1 \\
&= T(n-2) + 2(n-1) + 2n + 2 \\
&= [T(n-3) + 2(n-2) + 1] + 2(n-1) + 2n + 2 \\
&= T(n-3) + 2(n-2) + 2(n-1) + 2n + 3 \\
&\ \ \vdots \\
&= T(n-i) + \left[2 \sum_{j=n-(i-1)}^{n} j \right] + i
\end{aligned}$$

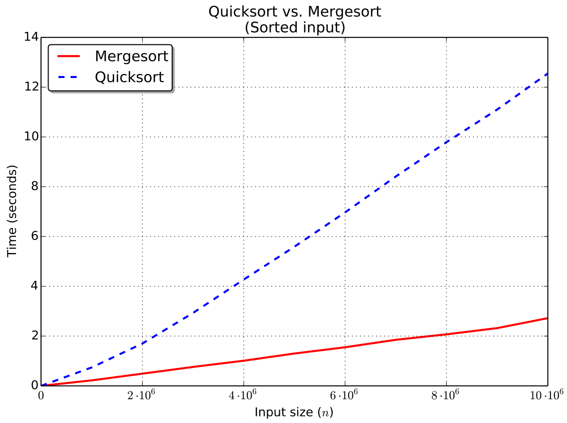
Quicksort Analysis (4)

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + 2n + 1 & \text{if } n \geq 2 \end{cases}$$

Solution by repeated substitution:

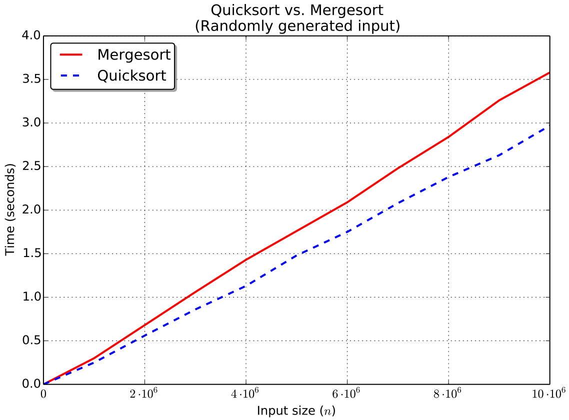
$$\begin{aligned}
T(n) &= T(n-i) + \left[2 \sum_{j=n-(i-1)}^{n} j\right] + i \\
&= T(n-(n-1)) + \left[2 \sum_{j=n-((n-1)-1)}^{n} j\right] + (n-1) \\
&= T(1) + \left[2 \sum_{j=2}^{n} j\right] + (n-1) \\
&= 1 + \left[2 \sum_{j=1}^{n} j\right] - 2 + n - 1 \\
&= 2\frac{n(n+1)}{2} + n - 2 = n^2 + 2n - 2
\end{aligned}$$

Practical Performance (1)



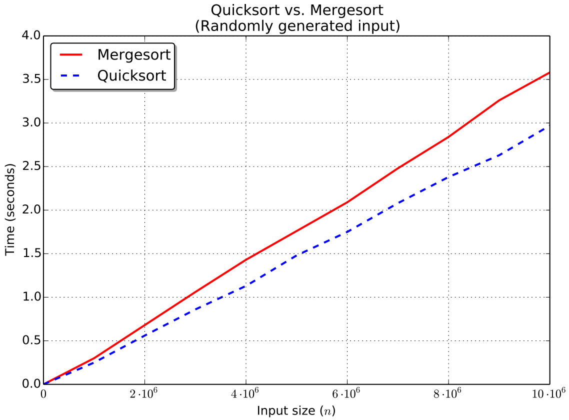
- Since Quicksort is $\Theta(n^2)$ in the worst case, Merge sort has a better worst case running time.

Practical Performance (2)



- The expected case running time of Quicksort is $\Theta(n \log n)$, and Quicksort tends to be faster in practice on most inputs.

Practical Performance (3)



- When the pivot at each step is a randomly-selected element, or if the input values are randomly shuffled before sorting, the running time will be $\Theta(n \log n)$ with very high probability.

Sorting Algorithms

	Running Time			Extra	Stable?
	Best Case	Expected Case	Worst Case	Space	
Selection Based					
Heap Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(1)$	No
Insertion Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	Yes
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	No
Divide and Conquer					
Merge Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$	Yes
Quicksort	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n)$	Yes
Other					
Bubble Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	Yes
Radix Sort[1]	$\Theta(dn + b)$	$\Theta(dn + b)$	$\Theta(dn + b)$	$\Theta(n + b)$	Yes

[1]Integers only: d-digit values in base b