Data Structures and Algorithms

Recursive Sorting

Chris Brooks

Department of Computer Science
University of San Francisco
Basic sorting algorithms all run in $\Theta(n^2)$ time.

We can do better by sorting sublists and combining results.
12-1: Merge Sort – Recursive Sorting

6 Base Case:
   △ A list of length 1 or length 0 is already sorted

6 Recursive Case:
   △ Split the list in half
   △ Recursively sort two halves
   △ Merge sorted halves together
51826437

5128

51286437

5128

51286437

5128

51286437

1528

15284637

1258

12583467

12345678
12-3: Merging

6 Merge lists into a new list, $T$
6 Maintain three pointers (indices) $i$, $j$, and $n$
   △ $i$ is index of left hand list
   △ $j$ is index of right hand list
   △ $n$ is index of destination list list $T$
6 If $A[i] < A[j]$
   △ $T[n] = A[i]$, increment $n$ and $i$
6 else
   △ $T[n] = A[j]$, increment $n$ and $j$
12-4: $\Theta()$ for Merge Sort

\begin{align*}
T(0) &= c_1 & \text{for some constant } c_1 \\
T(1) &= c_2 & \text{for some constant } c_2 \\
T(n) &= nc_3 + 2T(n/2) & \text{for some constant } c_3 \\
T(n) &= nc_3 + 2T(n/2)
\end{align*}
12-5: \( \Theta() \) for Merge Sort

\[
T(0) = c_1 \quad \text{for some constant } c_1 \\
T(1) = c_2 \quad \text{for some constant } c_2 \\
T(n) = nc_3 + 2T(n/2) \quad \text{for some constant } c_3
\]

\[
T(n) = nc_3 + 2T(n/2) \\
= nc_3 + 2(n/2c_3 + 2T(n/4)) \\
= 2nc_3 + 4T(n/4)
\]
$T(0) = c_1$ for some constant $c_1$

$T(1) = c_2$ for some constant $c_2$

$T(n) = nc_3 + 2T(n/2)$ for some constant $c_3$

$T(n) = nc_3 + 2T(n/2)$

$= nc_3 + 2(n/2c_3 + 2T(n/4))$

$= 2nc_3 + 4T(n/4)$

$= 2nc_3 + 4(n/4c_3 + 2T(n/8))$

$= 3nc_3 + 8T(n/8)$
$T(0) = c_1$ for some constant $c_1$

$T(1) = c_2$ for some constant $c_2$

$T(n) = nc_3 + 2T(n/2)$ for some constant $c_3$

$T(n)$

$= nc_3 + 2T(n/2)$

$= nc_3 + 2(n/2c_3 + 2T(n/4))$

$= 2nc_3 + 4T(n/4)$

$= 2nc_3 + 4(n/4c_3 + 2T(n/8))$

$= 3nc_3 + 8T(n/8))$

$= 3nc_3 + 8(n/8c_3 + 2T(n/16))$

$= 4nc_3 + 16T(n/16)$
\( T(0) = c_1 \) for some constant \( c_1 \)
\( T(1) = c_2 \) for some constant \( c_2 \)
\( T(n) = nc_3 + 2T(n/2) \) for some constant \( c_3 \)

\[
T(n) = nc_3 + 2T(n/2) \\
= nc_3 + 2(n/2c_3 + 2T(n/4)) \\
= 2nc_3 + 4T(n/4) \\
= 2nc_3 + 4(n/4c_3 + 2T(n/8)) \\
= 3nc_3 + 8T(n/8) \\
= 3nc_3 + 8(n/8c_3 + 2T(n/16)) \\
= 4nc_3 + 16T(n/16) \\
= 5nc_3 + 32T(n/32)
\]
12-9: $\Theta()$ for Merge Sort

\[ T(0) = c_1 \quad \text{for some constant } c_1 \]
\[ T(1) = c_2 \quad \text{for some constant } c_2 \]
\[ T(n) = nc_3 + 2T(n/2) \quad \text{for some constant } c_3 \]

\[
T(n) = nc_3 + 2T(n/2) \\
= nc_3 + 2(n/2c_3 + 2T(n/4)) \\
= 2nc_3 + 4T(n/4) \\
= 2nc_3 + 4(n/4c_3 + 2T(n/8)) \\
= 3nc_3 + 8T(n/8) \\
= 3nc_3 + 8(n/8c_3 + 2T(n/16)) \\
= 4nc_3 + 16T(n/16) \\
= 5nc_3 + 32T(n/32) \\
= knc_3 + 2^kT(n/2^k) \]
12-10: $\Theta()$ for Merge Sort

\[
T(0) = c_1 \\
T(1) = c_2 \\
T(n) = knc_3 + 2^k T(n/2^k)
\]

Pick a value for $k$ such that $n/2^k = 1$:

\[
\begin{align*}
\frac{n}{2^k} & = 1 \\
n & = 2^k \\
\lg n & = k \\
T(n) & = (\lg n)nc_3 + 2^{\lg n} T(n/2^{\lg n}) \\
& = c_3 n \lg n + nT(n/n) \\
& = c_3 n \lg n + nT(1) \\
& = c_3 n \lg n + c_2 n \\
& \in O(n \lg n)
\end{align*}
\]
$T(n)$
12-12: $\Theta()$ for Merge Sort

\[
\begin{align*}
T(n/2) & \quad \text{c*n} \quad \text{T(n/2)} \\
\end{align*}
\]
12-13: $\Theta()$ for Merge Sort

$$
c^*n
$$

$$
c^*(n/2)
$$

$$
T(n/4) \quad T(n/4)
$$

$$
c^*(n/2)
$$

$$
T(n/4) \quad T(n/4) \quad T(n/4) \quad T(n/4)
$$
12-14: $\Theta()$ for Merge Sort
12-15: \( \Theta() \) for Merge Sort

\[
\begin{align*}
\text{c*}n \\
\text{c*}(n/2) & \quad \text{c*}(n/2) \\
\text{c*}(n/4) & \quad \text{c*}(n/4) & \quad \text{c*}(n/4) & \quad \text{c*}(n/4) \\
\text{...} & \quad \text{...} & \quad \text{...} & \quad \text{...} \\
\end{align*}
\]
12-16: \( \Theta() \) for Merge Sort

Total time = \( c^* n \ lg \ n \)

\( \Theta(n \ lg \ n) \)
Merge Sort:

- Divide the list two parts
  - No work required – just calculate midpoint
- Recursively sort two parts
- Combine sorted lists into one list
  - Some work required – need to merge lists
Quick Sort:

- Divide the list two parts
  - Some work required – Small elements in left sublist, large elements in right sublist
- Recursively sort two parts
- Combine sorted lists into one list
  - No work required!
12-19: Quick Sort

6 Pick a pivot element
6 Reorder the list:
   △ All elements < pivot
   △ Pivot element
   △ All elements > pivot
6 Recursively sort elements < pivot
6 Recursively sort elements > pivot
Example: 3 7 2 8 1 4 6

Suppose 3 is our pivot

- Split into 3 2 1 7 8 4 6
  - Sort left half - suppose 2 is the pivot
  - 1 2 3
  - Sort right half - suppose 1 is the pivot
  - 4 6 7 8
  - Recurse and merge
Basic Idea:

1. Swap pivot element out of the way (we’ll swap it back later)

2. Maintain two pointers, $i$ and $j$
   - $i$ points to the beginning of the list
   - $j$ points to the end of the list

3. Move $i$ and $j$ in to the middle of the list – ensuring that all elements to the left of $i$ are $<$ the pivot, and all elements to the right of $j$ are greater than the pivot

4. Swap pivot element back to middle of list
12-22: Quick Sort - Partitioning

Pseudocode:

6 Pick a pivot index
6 Swap A[pivotindex] and A[high]
6 Set $i \leftarrow low$, $j \leftarrow high-1$
6 while ($i \leq j$)
   △ swap $A[i]$ and $A[j]$
   △ increment $i$, decrement $j$
6 swap $A[i]$ and $A[pivot]$
Coming up with a recurrence relation for quicksort is harder than mergesort.

How the problem is divided depends upon the data:

- Break list into:
  - size 0, size \( n - 1 \)
  - size 1, size \( n - 2 \)
  - \( \ldots \)
  - size \( \lceil (n - 1)/2 \rceil \), size \( \lfloor (n - 1)/2 \rfloor \)
  - \( \ldots \)
  - size \( n - 2 \), size 1
  - size \( n - 1 \), size 0
Worst case performance occurs when break list into size $n - 1$ and size 0

- $T(0) = c_1$ for some constant $c_1$
- $T(1) = c_2$ for some constant $c_2$
- $T(n) = nc_3 + T(n - 1) + T(0)$ for some constant $c_3$

$$T(n) = nc_3 + T(n - 1) + T(0)$$
$$= T(n - 1) + nc_3 + c_2$$
Worst case: \( T(n) = T(n - 1) + nc_3 + c_2 \)

\[
\begin{align*}
T(n) &= T(n - 1) + nc_3 + c_2
\end{align*}
\]
Worst case: \( T(n) = T(n - 1) + nc_3 + c_2 \)

\[
\begin{align*}
T(n) &= T(n - 1) + nc_3 + c_2 \\
&= [T(n - 2) + (n - 1)c_3 + c_2] + nc_3 + c_2 \\
&= T(n - 2) + (n + (n - 1))c_3 + 2c_2
\end{align*}
\]
Worst case: $T(n) = T(n - 1) + nc_3 + c_2$

$$
T(n) \\
= T(n - 1) + nc_3 + c_2 \\
= [T(n - 2) + (n - 1)c_3 + c_2] + nc_3 + c_2 \\
= T(n - 2) + (n + (n - 1))c_3 + 2c_2 \\
= [T(n - 3) + (n - 2)c_3 + c_2] + (n + (n - 1))c_3 + 2c_2 \\
= T(n - 3) + (n + (n - 1) + (n - 2))c_3 + 3c_2
$$
Worst case: \( T(n) = T(n - 1) + nc_3 + c_2 \)

\[
T(n) \\
= T(n - 1) + nc_3 + c_2 \\
= [T(n - 2) + (n - 1)c_3 + c_2] + nc_3 + c_2 \\
= T(n - 2) + (n + (n - 1))c_3 + 2c_2 \\
= [T(n - 3) + (n - 2)c_3 + c_2] + (n + (n - 1))c_3 + 2c_2 \\
= T(n - 3) + (n + (n - 1) + (n - 2))c_3 + 3c_2 \\
= T(n - 4) + (n + (n - 1) + (n - 2) + (n - 3))c_3 + 4c_2
\]
Worst case: \( T(n) = T(n - 1) + nc_3 + c_2 \)

\[
T(n) \\
= T(n - 1) + nc_3 + c_2 \\
= [T(n - 2) + (n - 1)c_3 + c_2] + nc_3 + c_2 \\
= T(n - 2) + (n + (n - 1))c_3 + 2c_2 \\
= [T(n - 3) + (n - 2)c_3 + c_2] + (n + (n - 1))c_3 + 2c_2 \\
= T(n - 3) + (n + (n - 1) + (n - 2))c_3 + 3c_2 \\
= T(n - 4) + (n + (n - 1) + (n - 2) + (n - 3))c_3 + 4c_2 \\
\ldots \\
= T(n - k) + (\sum_{i=0}^{k-1} (n - i)c_3) + kc_2
\]
Worst case:

\[ T(n) = T(n - k) + (\sum_{i=0}^{k-1} (n - i)c_3) + kc_2 \]

Set \( k = n \):

\[
T(n) = T(n - n) + (\sum_{i=0}^{n-1} (n - i)c_3) + kc_2 \\
= T(0) + (\sum_{i=0}^{n-1} (n - i)c_3) + kc_2 \\
= T(0) + (\sum_{i=0}^{n-1} ic_3) + kc_2 \\
= c_1 + c_3n(n + 1)/2 + kc_2 \\
\in \Theta(n^2)
\]
$T(n)$

12-31: $\Theta()$ for Quick Sort
12-32: $\Theta()$ for Quick Sort
12-33: \( \Theta() \) for Quick Sort

\[ \begin{align*}
&c^n \\
&c(n-1) \\
&T(n-2) & c2 & T(0)
\end{align*} \]
12-34: $\Theta()$ for Quick Sort

\[ c^n \]
\[ c^{n-1} \quad c^2 \]
\[ c^{n-2} \quad c^2 \]
\[ T(n-3) \quad T(0) \]
12-35: $\Theta()$ for Quick Sort

$$c^*n$$

$$c^*(n-1)$$

$$c^*(n-2)$$

$$c^*(n-3)$$

$$\ldots$$

$$c^*(n-1) + c^2$$

$$c^*(n-2) + c^2$$

$$c^*(n-3) + c^2$$

$$c^*(n-k) + c^2$$
12-36: $\Theta(\cdot)$ for Quick Sort

Total time = $c n (n+1)/2 + nc^2$

$\Theta(n^2)$
Best case performance occurs when break list into size \([n - 1]/2\) and size \([n - 1]/2\)

\[
T(0) = c_1 
\]

\[
T(1) = c_2 
\]

\[
T(n) = nc_3 + 2T(n/2) 
\]

This is the same as Merge Sort: \(\Theta(n \log n)\)
If Quicksort is $\Theta(n^2)$ on some lists, why is it called quick?

- Most lists give running time of $\Theta(n \lg n)$
  - Average case running time is $\Theta(n \lg n)$
- Constants are very small
  - Constants don’t matter when complexity is different
  - Constants do matter when complexity is the same

What lists will cause Quick Sort to have $\Theta(n^2)$ performance?
Quick Sort has worst-case performance when:

- The list is sorted (or almost sorted)
- The list is inverse sorted (or almost inverse sorted)

Many lists we want to sort are almost sorted!

How can we fix Quick Sort?
12-40: Better Partitions

- Pick the middle element as the pivot
  - Sorted and reverse sorted lists give good performance
- Pick a random element as the pivot
  - No single list always gives bad performance
- Pick the median of 3 elements
  - First, Middle, Last
  - 3 Random Elements
Insertion Sort runs faster than Quick Sort on small lists
  △ Why?

We can combine Quick Sort & Insertion Sort
  △ When lists get small, run Insertion Sort instead of a recursive call to Quick Sort
  △ When lists get small, stop! After call to Quick Sort, list will be almost sorted – finish the job with a single call to Insertion Sort
Build a heap out of the data

Repeat:
  △ Remove the largest element from the list, place it at end of heap

Until all elements have been removed from the heap

The list is now sorted

Example: 3 1 7 2 5 4
Building the heap takes time $\Theta(n)$

Each of the $n$ RemoveMax calls takes time $O(lg n)$

Total time: $(n \ lg \ n)$ (also $\Theta(n \ lg \ n)$)