CS1800

Admin:

- hw8 due today
- hw9 & "exam3" next Tuesday
- I hope to finish a few minutes early today and handle hw / exam content questions like we do in recitation.

Content:

- merge sort & runtime analysis (counting comparisons in the worst case)
- skill: solving recurrence relations via substitution

Quantifying runtime (search algorithms):

Runtime: how many "operations" required to complete algorithm for input of size n

To simplify our analysis of algorithms:

- lets only count comparisons (is item0 less than, equal to, or greater than item1?)
- lets assume the worst possible input for a given algorithm (requiring the most comparisons)

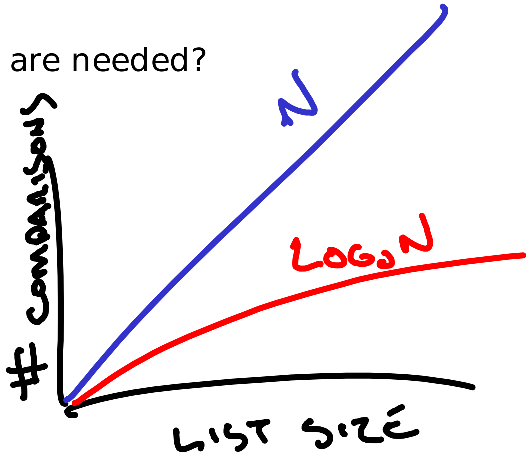
In the worst case, for an input list with n items how many comparisons are needed?

- unordered linear search

$$T_{LINEAR}(N) = N$$

- binary search

$$T_{BINARY}(N) = LOG_2 N$$



Quantifying runtime (sort algorithms):

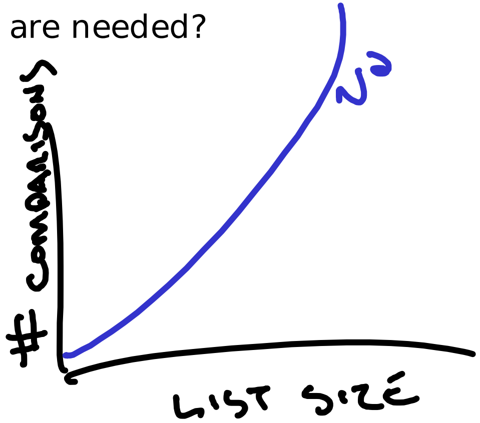
Runtime: how many "operations" required to complete algorithm for input of size n

To simplify our analysis of algorithms:
- lets only count comparisons (is item0 less than, equal to, or greater than item1?)
- lets assume the worst possible input for a given algorithm (requiring the most comparisons)

In the worst case, for an input list with n items how many comparisons are needed?
- insertion sort

$$T_{insertion}(N) = N^2$$

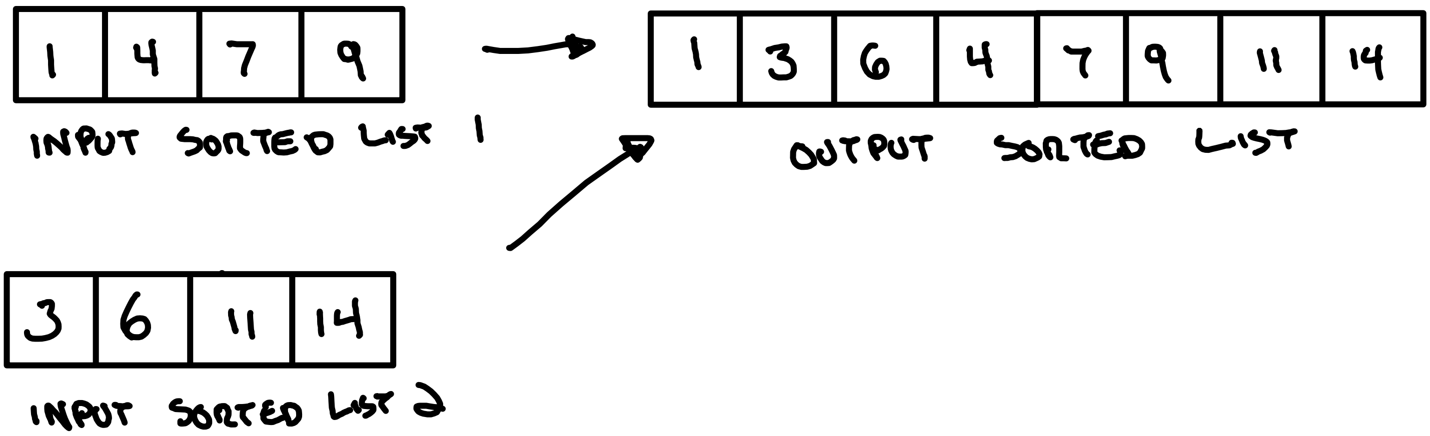
SOME OTHER SORT METHOD?



Merge Operation: Combining Two Sorted Lists Into One Sorted List

Approach: comparison: find the starting list whose current item is smallest
- move this smallest item into final list
- move current index of this list to the right

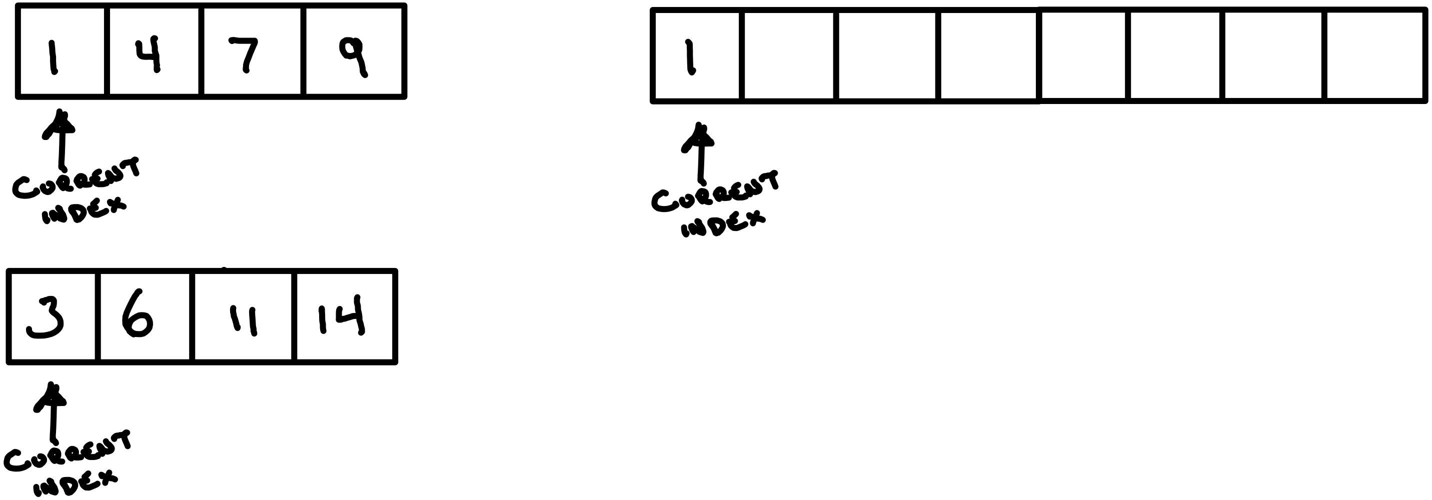
repeat above until one starting list is out of items and then:
- place all items in other list into output list, in same order



Merge Operation: Combining Two Sorted Lists Into One Sorted List

Approach: comparison: find the starting list whose current item is smallest
- move this smallest item into final list
- move current index of this list to the right

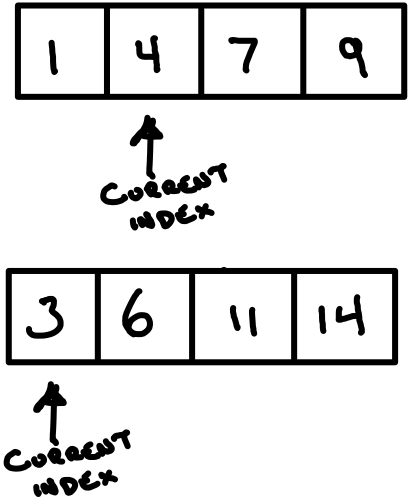
repeat above until one starting list is out of items and then:
- place all items in other list into output list, in same order

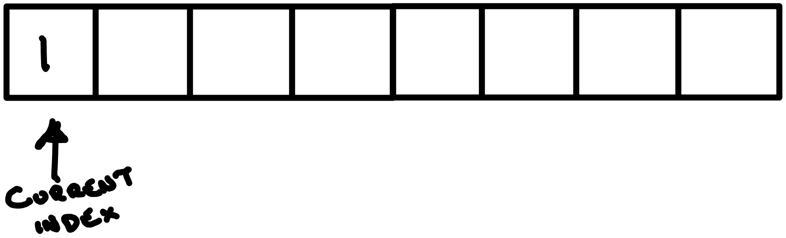


Merge Operation: Combining Two Sorted Lists Into One Sorted List

Approach: comparison: find the starting list whose current item is smallest
- move this smallest item into final list
- move current index of this list to the right

repeat above until one starting list is out of items and then:
- place all items in other list into output list, in same order



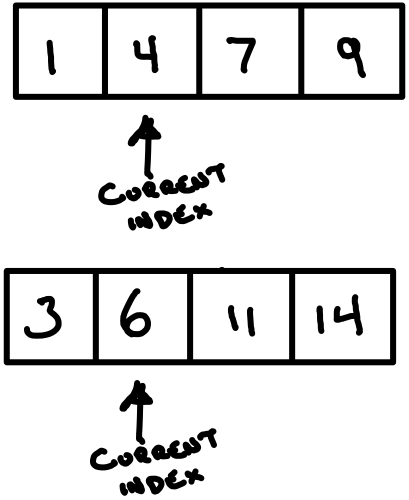


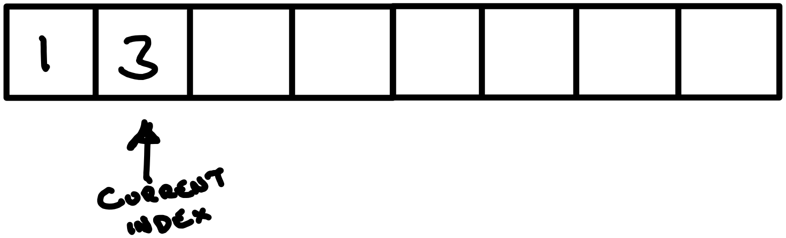
SINCE 1 < 3 WE MOVE 1 TO OUTPUT AND EXAMINE NEXT ITEM IN LIST IT CAME FROM

Merge Operation: Combining Two Sorted Lists Into One Sorted List

Approach: comparison: find the starting list whose current item is smallest
- move this smallest item into final list
- move current index of this list to the right

repeat above until one starting list is out of items and then:
- place all items in other list into output list, in same order



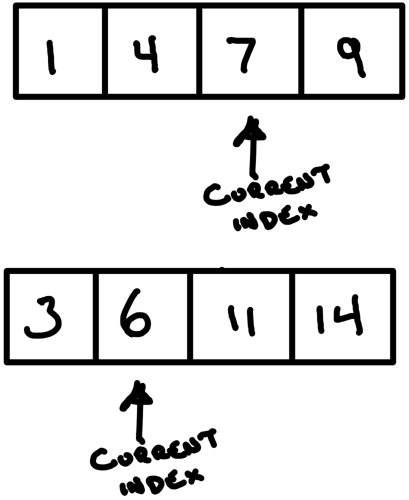


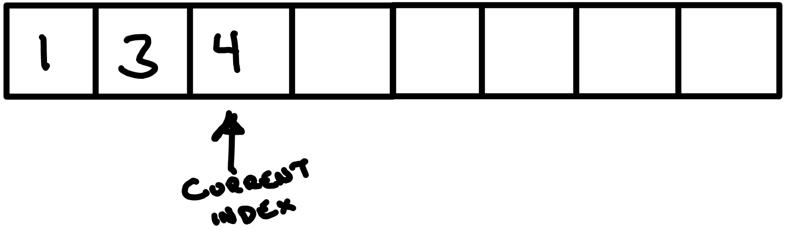
SINCE 3 < 4 WE MOVE 3 TO OUTPUT AND EXAMINE NEXT ITEM IN LIST IT CAME FROM

Merge Operation: Combining Two Sorted Lists Into One Sorted List

Approach: comparison: find the starting list whose current item is smallest
- move this smallest item into final list
- move current index of this list to the right

repeat above until one starting list is out of items and then:
- place all items in other list into output list, in same order



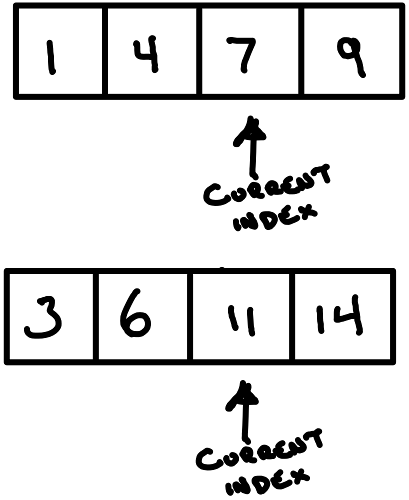


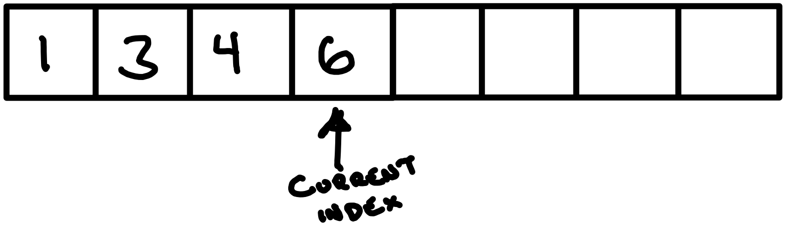
SINCE 4 < 6 WE MOVE 4 TO OUTPUT AND EXAMINE NEXT ITEM IN LIST IT CAME FROM

Merge Operation: Combining Two Sorted Lists Into One Sorted List

Approach: comparison: find the starting list whose current item is smallest
- move this smallest item into final list
- move current index of this list to the right

repeat above until one starting list is out of items and then:
- place all items in other list into output list, in same order



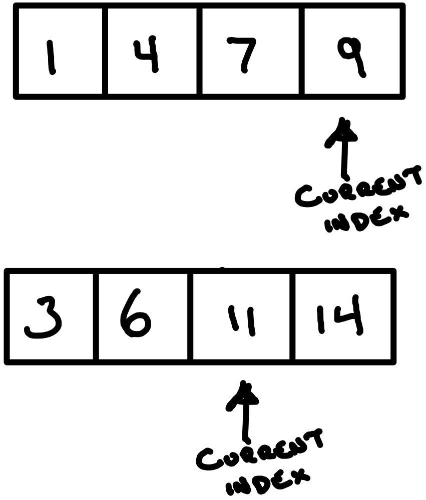


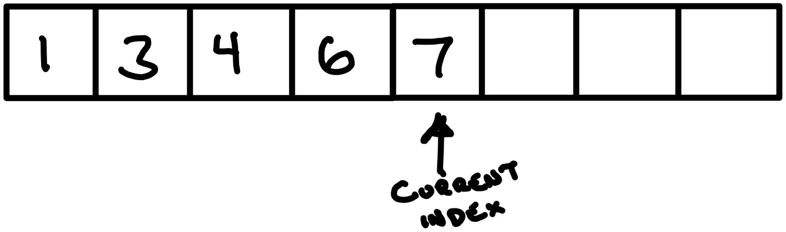
SINCE $6 < 7$ WE MOVE 6 TO OUTPUT AND EXAMINE NEXT ITEM IN LIST IT CAME FROM

Merge Operation: Combining Two Sorted Lists Into One Sorted List

Approach: comparison: find the starting list whose current item is smallest
- move this smallest item into final list
- move current index of this list to the right

repeat above until one starting list is out of items and then:
- place all items in other list into output list, in same order





SINCE $7 < 11$ WE MOVE 7 TO OUTPUT AND EXAMINE NEXT ITEM IN LIST IT CAME FROM

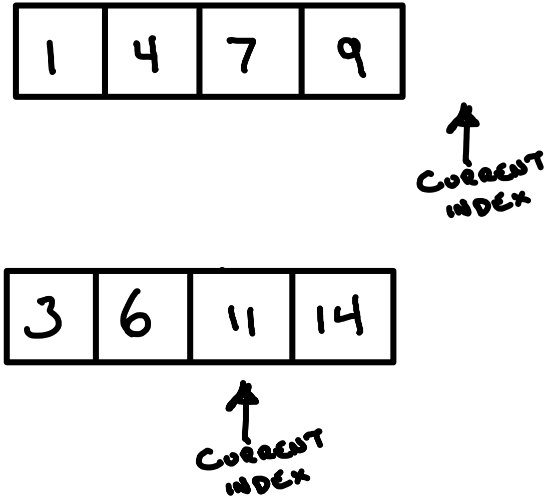
Merge Operation: Combining Two Sorted Lists Into One Sorted List

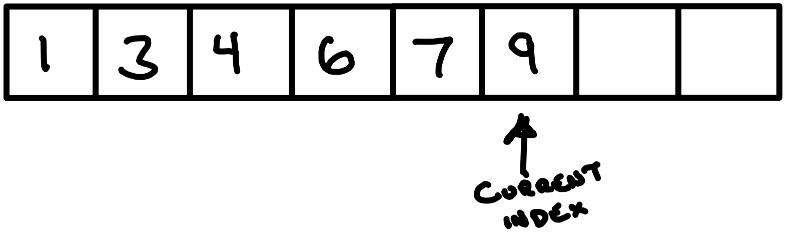
Approach: comparison: find the starting list whose current item is smallest

- move this smallest item into final list
- move current index of this list to the right

repeat above until one starting list is out of items and then:

- place all items in other list into output list, in same order



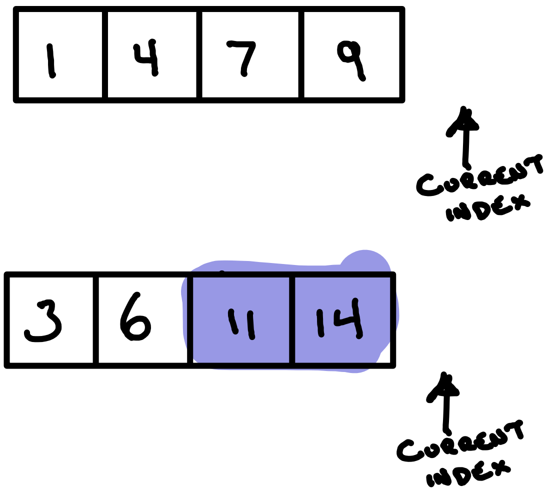


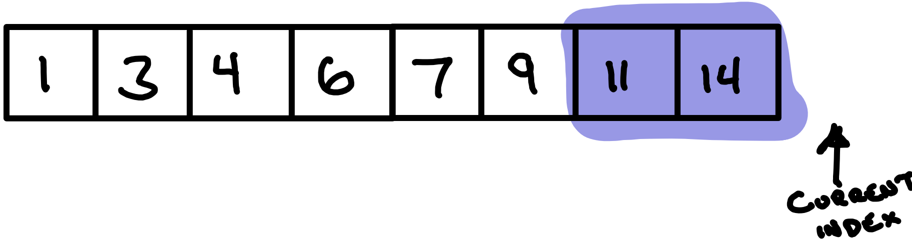
SINCE 9<11 WE MOVE 9 TO OUTPUT AND EXAMINE NEXT ITEM IN LIST IT CAME FROM

Merge Operation: Combining Two Sorted Lists Into One Sorted List

Approach: comparison: find the starting list whose current item is smallest
- move this smallest item into final list
- move current index of this list to the right

repeat above until one starting list is out of items and then:
- place all items in other list into output list, in same order





SINCE ONE INPUT LIST RAN OUT OF ITEMS WE MOVE ALL REMAINING ITEMS IN OTHER LIST TO OUTPUT (IN SAME ORDER)

In Class Activity:

Build a worst case (requiring the most comparisons) example of merge sort which combines two sorted lists (each of length 4) into an output list of size 8.

How many comparisons, in the worst case, will it take to combine two sorted lists (each of length n/2) into an output list of size n?

Merging: Worst Case Scenario (requiring most comparisons)

Every comparison moves a single item to the output list

When one list runs out of items, the whole remaining list is moved into output (see blue highlights @ end of example a few slides ago). No comparisons are required for these remaining items!

The worst case scenario is when we move only a single item from the "remaining list" to the output.

(That is, the last items of each input list become the last two items in the output list).

< demonstrate this with cards>

Worst Case Scenario of Merge Operation: N - 1 comparisons to merge two lists of size n/2

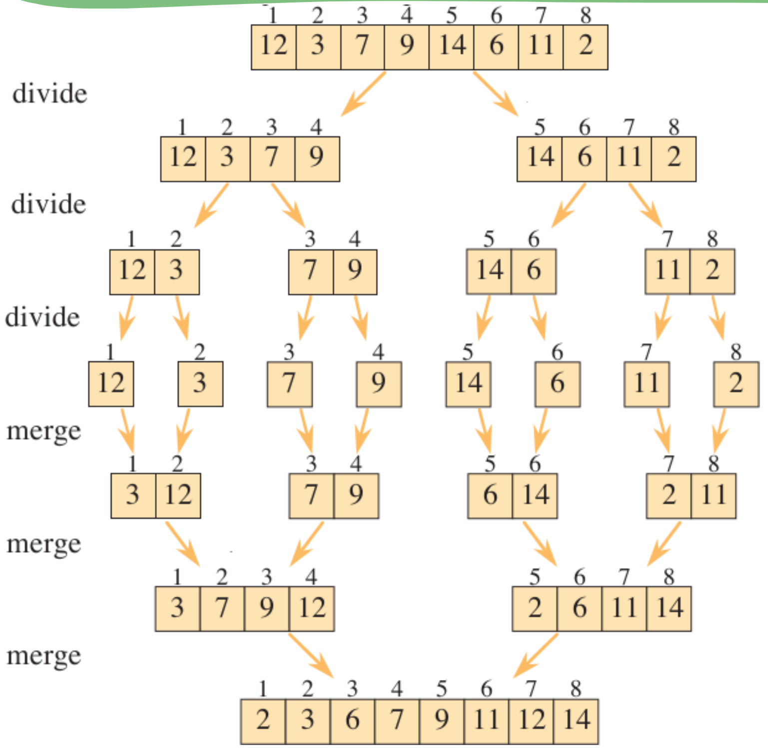
LETS IGNORE THIS -1, IT WILL SIMPLIFY ANALYSIS WITHOUT CHANGING RESULT (SLOWER BIG-O GROWTH THAN N)

Merging: Worst Case Scenario (requiring most comparisons)

Punchline:

Merging so the output list has N items requires (at worst) N comparisons

Merge Sort: How do we sort a list with this merge operation?



Approach:

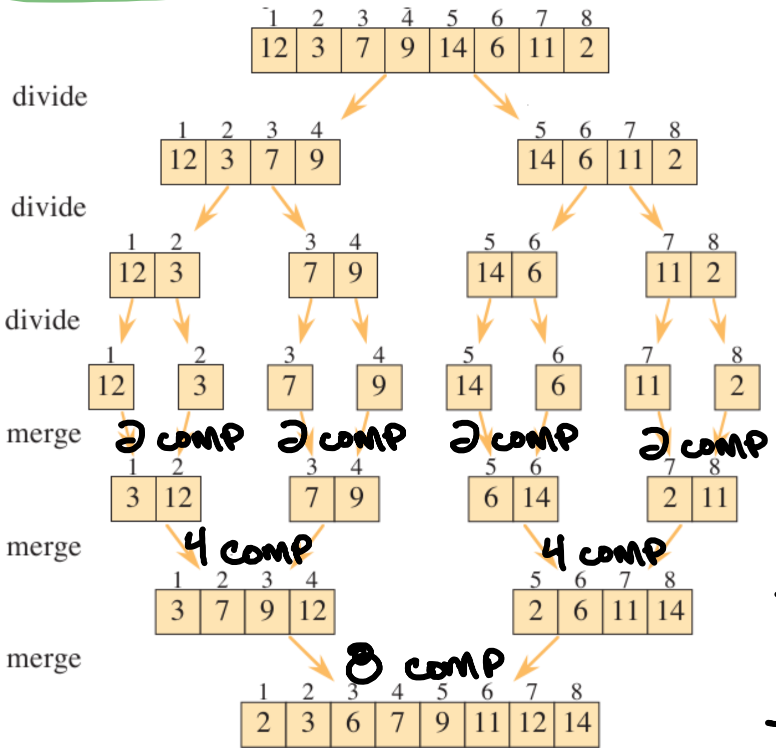
Divide the input list in half until they're all length 1 lists (which are sorted!)

Merge lists back together

Super simple, right? There are many algorithms which fit this pattern:

Divide-and-conquer: split problem into sub-problems until sub-problems easily solved

Merge Sort: Runtime Analysis (comparisons in worst case scenario) on this example



Observe:

- there are no comparisons in divide

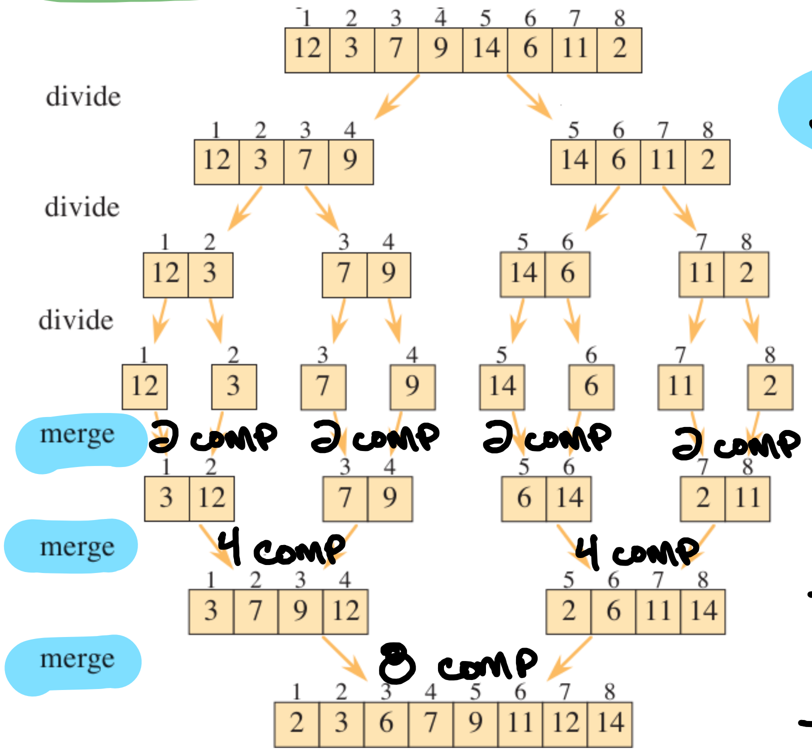
- each merge operation (in worst case) uses as many comparison as output list size (see previous slide's "punchline")

$\rightarrow$ 4 lists $\cdot$ 2 $\frac{\text{comp}}{\text{list}}$ = 8 comp

$\rightarrow$ 2 lists $\cdot$ 4 $\frac{\text{comp}}{\text{list}}$ = 8 comp

$\rightarrow$ 1 list $\cdot$ 8 $\frac{\text{comp}}{\text{list}}$ = 8 comp

Merge Sort: Runtime Analysis (comparisons in worst case scenario) on this example



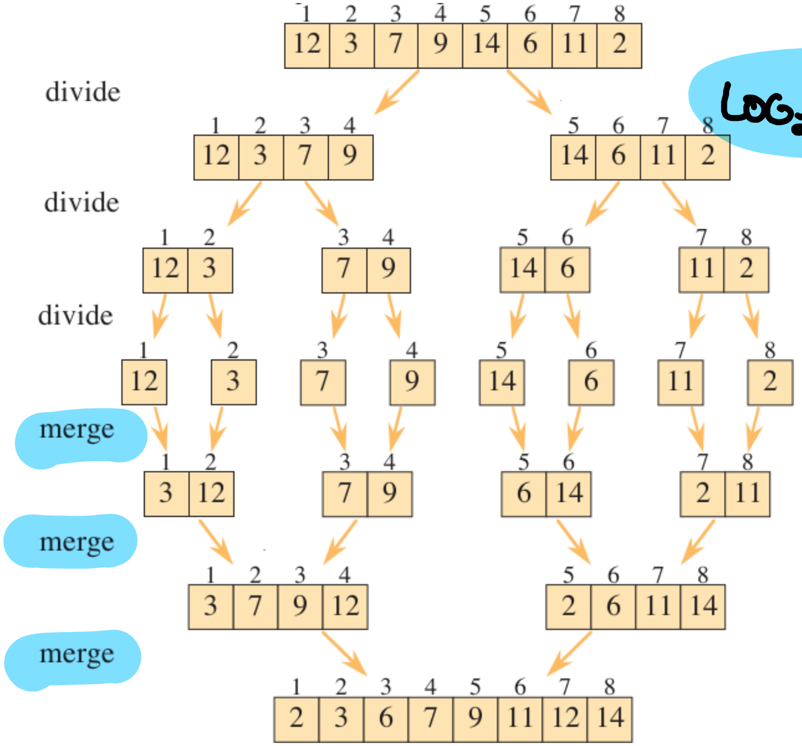
3 levels merging $\cdot$ $8 \frac{\text{comp}}{\text{level}}$ = 24 comp

→ 4 lists $\cdot$ $2 \frac{\text{comp}}{\text{list}}$ = 8 comp

→ 2 lists $\cdot$ $4 \frac{\text{comp}}{\text{list}}$ = 8 comp

→ 1 list $\cdot$ $8 \frac{\text{comp}}{\text{list}}$ = 8 comp

Merge Sort: Runtime Analysis (comparisons in worst case scenario) for list with n items



$\log_2 N$ LEVELS MERGING $\cdot$ $\frac{N \text{ COMP}}{\text{LEVEL}} = \Theta(N \log N)$

EVERY LEVEL USES N COMPARISONS

$N = 2^{\#\text{ LEVELS}} \longleftrightarrow \#\text{ LEVELS} = \log_2 N$

FROM EXAMPLE:

$8 = 2^3 \longleftrightarrow 3 = \log_2 8$

Quantifying runtime (sort algorithms):

Runtime: how many "operations" required to complete algorithm for input of size n

To simplify our analysis of algorithms:

- lets only count comparisons (is item0 less than, equal to, or greater than item1?)
- lets assume the worst possible input for a given algorithm (requiring the most comparisons)

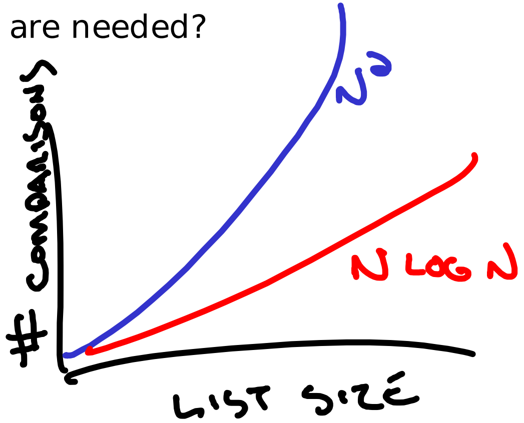
In the worst case, for an input list with n items how many comparisons are needed?

- insertion sort

$$T_{\text{INSERTION}}(N) = N^2$$

- merge sort

$$T_{\text{MERGE}}(N) = N \log N$$



Recurrence Relations:

Another way of analyzing worst case comparisons in merge sort

(Why learn another way? There are many other divide and conquer methods which don't have a fun little analysis picture like merge sort did a few slides ago ... recurrences are a tool which will work for these!)

Building a Recurrence Relation for Merge Sort:

T(n) = number of comparisons it takes to run merge sort on a list of size n in worst case

$$T(n) = 2\,T(n/2) + n$$

To run merge sort:

- split input list of size n into two lists of size n/2, run merge sort on each

    worst case cost: 2 * T(n/2)

- merge these two (now sorted) lists of size n/2 together via merge operation

    worst case cost: n operations   (see previous "punchline")

Whats a recurrence relation?

T(n) = number of comparisons it takes to run merge sort on a list of size n

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

RECURRENCE RELATION = AN EQUALITY WHICH EXPRESSES EACH ITEM OF A SEQUENCE AS A FUNCTION OF PREVIOUS TERMS

Bad news: recurrences not easily understood (is this fast or slow growing?)

Solving a Recurrence: Substitution Method

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$= 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$$

$$= 2^2 T\left(\frac{n}{4}\right) + 2n$$

$$= 2^2\left(2T(n/8) + n/4\right) + 2n$$

$$= 2^3 T(n/8) + 3n$$

$$\rightarrow \quad T(\quad) = 2T(\quad/2) + \quad$$

$$T\left(\frac{n}{2}\right) = 2T(n/4) + \frac{n}{2}$$

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4}$$

Each time we see the T function on the right hand side:

- substitute for equivilent expression using recurrence (i.e. green / red)

- simplify resulting expression

Solving a Recurrence: Substitution Method

$$T(n) = 2T\left(\frac{n}{2}\right) + n \quad \longleftarrow K=1$$

$$= 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$$

$$= 2^2 T\left(\frac{n}{4}\right) + 2n \quad \longleftarrow K=2$$

$$= 2^2\left(2T(n/8) + n/4\right) + 2n$$

$$= 2^3 T(n/8) + 3n \quad \longleftarrow K=3$$

$$= 2^K T(n/2^K) + Kn$$

BY NOTICING A PATTERN WE CAN GET AN EXPRESSION FOR ALL TERMS IN SEQUENCE

A helpful insight about merge sort

T(n) = number of comparisons it takes to run merge sort on a list of size n

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

$$T(1) = 0$$

It takes 0 operations to sort a list of size 1 (its already sorted, right?)

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$$

Which k (number of substitutions) provides n / 2^k = 1?

$$\frac{n}{2^k} = 1 \longleftrightarrow n = 2^k \longleftrightarrow k = \log_2 n$$

Lets use that k as it'll give T(1) on the right hand side (helpful since T(1) = 0)

$$T(n) = 2^{\log_2 n} \cdot T\left(\frac{n}{2^{\log_2 n}}\right) + n \log_2 n$$

$$= n \cdot T(1) + n \log_2 n$$

$$= n \log_2 n$$

In Class Activity:

Solve the following recurrences (answers to all are given)

i. $T(n) = T(n-1) + 1$ where $T(1) = 1$
solution: $T(n) = n$

In case you'd like some practice, here's a few more examples too:

ii. $T(n) = T(n-3) + 4$ where $T(1) = 1$
solution: $T(n) = (4n - 1) / 3$

iii. $T(n) = 7 * T(n-2)$ where $T(0) = 1$
solution: $T(n) = 7^{n/2}$

FULL SOLUTION ON FOLLOWING SLIDES

i. T(n) = T(n-1) + 1 where T(1) = 1
solution: T(n) = n

$$T(\cancel{\ }) = T(\cancel{\ } - 1) + 1$$

$$T(n-1) = T(n-1-1) + 1$$

$$T(n-2) = T(n-2-1) + 1$$

$$n - k = 1 \rightarrow k = n-1$$

$$T(n) = T(n-1) + 1 \quad \leftarrow k=1$$

$$= T(n-2) + 1 + 1$$

$$= T(n-2) + 2 \quad \leftarrow k=2$$

$$= T(n-3) + 1 + 2$$

$$= T(n-3) + 3 \quad \leftarrow k=3$$

$$= T(n-k) + k \overset{k=n-1}{=} T(1) + (n-1) = 1 + n - 1$$

$$= n$$

ii. $T(n) = T(n-3) + 4$ where $T(1) = 1$
solution: $T(n) = (4n - 1) / 3$

T(n) = 7 * T(n-2) where T(0) = 1
solution: T(n) = 7^{n/2}

$$T(\blacksquare) = 7\,T(\blacksquare - 2)$$

$$T(n-2) = 7\,T(n-4)$$

$$T(n-4) = 7\,T(n-6)$$

$$
\begin{aligned}
T(n) &= 7\,T(n-2) \quad \leftarrow k=1 \\
&= 7\big(7\,T(n-4)\big) \\
&= 7^2\,T(n-4) \quad \leftarrow k=2 \\
&= 7^2\big(7\,T(n-6)\big) \\
&= 7^3\,T(n-6) \quad \leftarrow k=3 \\
&= 7^k\,T(n-2k)
\end{aligned}
$$

T(n) = 7 * T(n-2) where T(0) = 1
solution: T(n) = 7^{n/2}

WHAT K BRINGS ME TO MY BASE CASE T(0)=1?

$$T(n) = 7^k \, T(n-2k)$$

$$n - 2k = 0 \longleftrightarrow k = \frac{n}{2}$$

LETS SUBSTITUTE THAT K IN!

$$= 7^{\frac{n}{2}} \, T\left(n - 2 \cdot \frac{n}{2}\right)$$

$$= 7^{n/2} \, T(n-n)$$

$$= 7^{n/2} \, T(0) = 7^{n/2}$$