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)

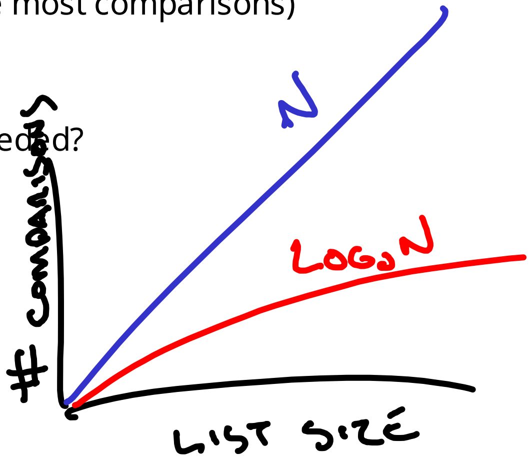
In the worst case, for an input list with n items how many comparisons are needed?

- 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:

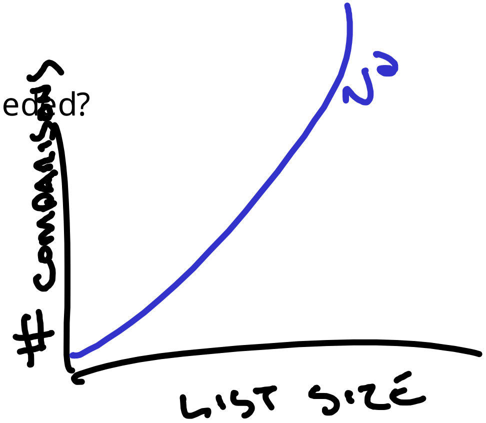
- lets only count comparisons (is item0 less than, equal to, or greater than item1?)
- lets assume the worst possible input for a given algorithm (requiring the most comparisons)

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?



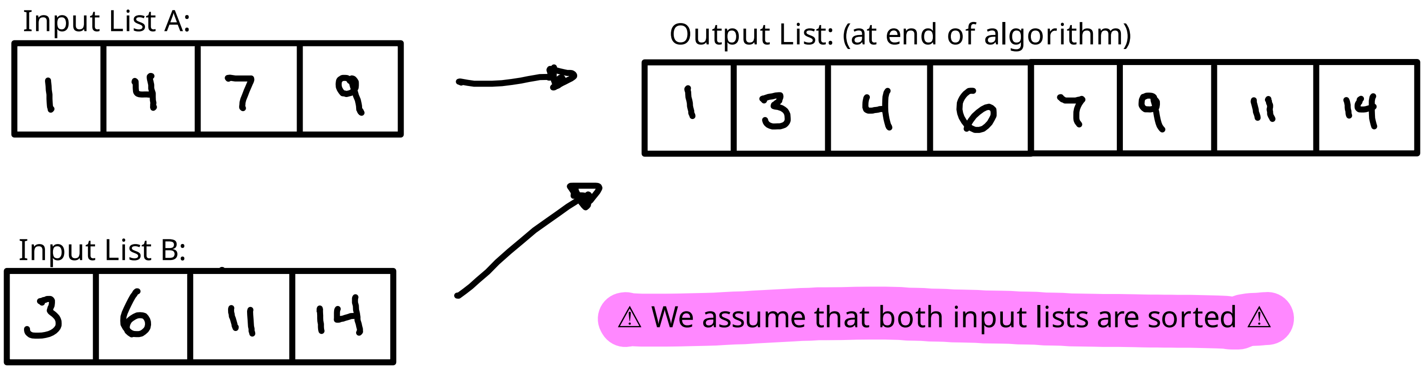
Merge Operation: Combining Two Sorted Lists Into One Sorted List

Approach:

compare: find input list whose current item is smallest

- move item to final list
- increment current index of this initial list

repeat above until one initial list is out of items, then place all items in other list into output list (in same order)



⚠ We assume that both input lists are sorted ⚠

Merge Operation: Combining Two Sorted Lists Into One Sorted List

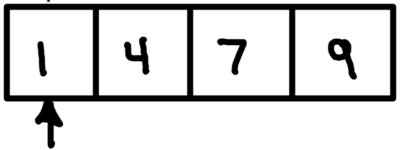
Approach:

compare: find input list whose current item is smallest

- move item to final list
- increment current index of this initial list

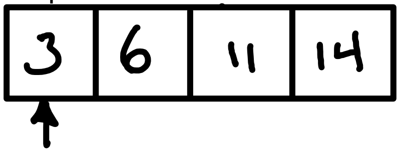
repeat above until one initial list is out of items, then place all items in other list into output list (in same order)

Input List A:



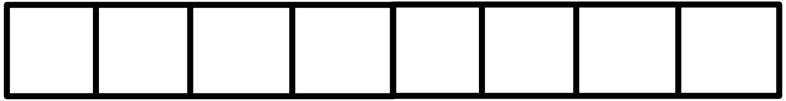
Current Index

Input List B:



Current Index

Output List:



Start of Algorithm:

- set current index to first item in each input list

Merge Operation: Combining Two Sorted Lists Into One Sorted List

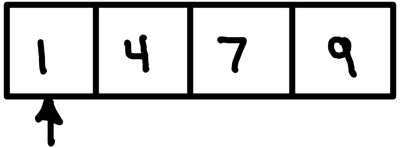
Approach:

compare: find input list whose current item is smallest

- move item to final list
- increment current index of this initial list

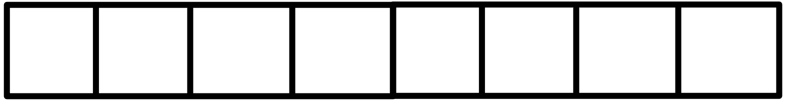
repeat above until one initial list is out of items, then place all items in other list into output list (in same order)

Input List A:

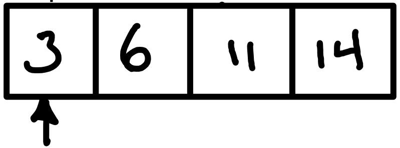


Current Index

Output List:



Input List B:



Current Index

Next Step:

Since 1 < 3, list A has smaller item:

- move this list's current item into the output list
- increment current index of this list (move to right one)

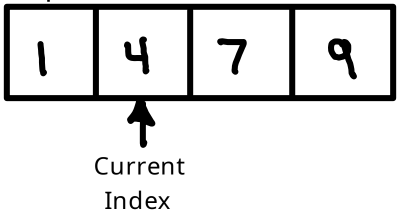
Merge Operation: Combining Two Sorted Lists Into One Sorted List

Approach:

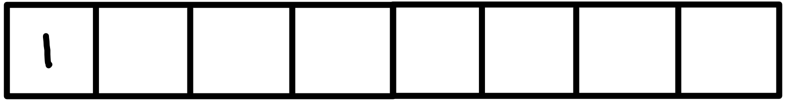
compare: find input list whose current item is smallest
- move item to final list
- increment current index of this initial list

repeat above until one initial list is out of items, then place all items in other list into output list (in same order)

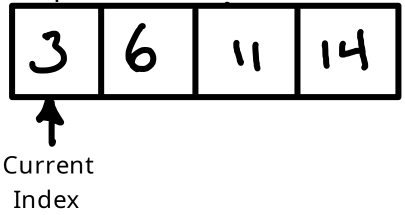
Input List A:



Output List:



Input List B:



Next Step:

Since 3 < 4, list B has smaller item:
- move this list's current item into the output list
- increment current index of this list (move to right one)

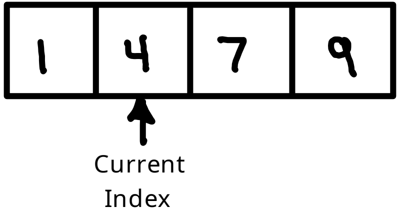
Merge Operation: Combining Two Sorted Lists Into One Sorted List

Approach:

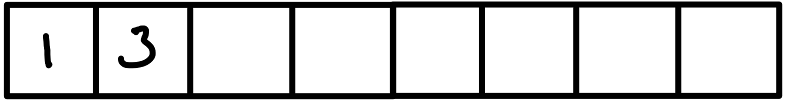
compare: find input list whose current item is smallest
- move item to final list
- increment current index of this initial list

repeat above until one initial list is out of items, then place all items in other list into output list (in same order)

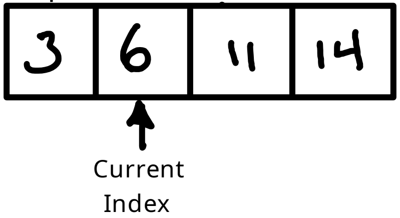
Input List A:



Output List:



Input List B:



Next Step:

Since 4 < 6, list A has smaller item:
- move this list's current item into the output list
- increment current index of this list (move to right one)

Merge Operation: Combining Two Sorted Lists Into One Sorted List

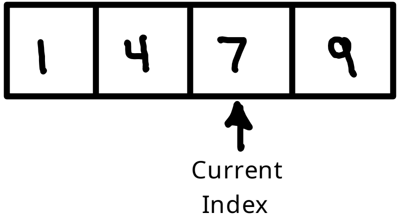
Approach:

compare: find input list whose current item is smallest

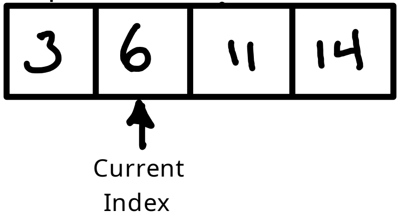
- move item to final list
- increment current index of this initial list

repeat above until one initial list is out of items, then place all items in other list into output list (in same order)

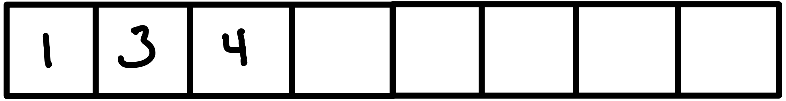
Input List A:



Input List B:



Output List:



Next Step:

Since 6 < 7, list B has smaller item:

- move this list's current item into the output list
- increment current index of this list (move to right one)

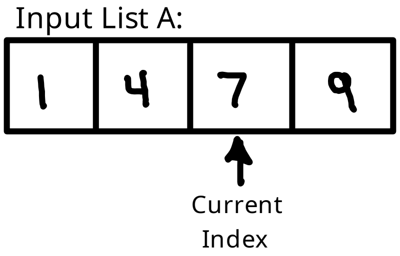
Merge Operation: Combining Two Sorted Lists Into One Sorted List

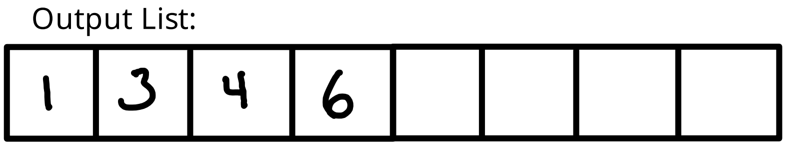
Approach:

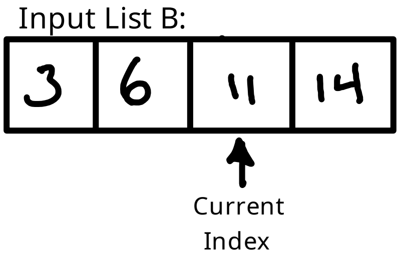
compare: find input list whose current item is smallest

- move item to final list
- increment current index of this initial list

repeat above until one initial list is out of items, then place all items in other list into output list (in same order)







Next Step:

Since 7 < 11, list A has smaller item:

- move this list's current item into the output list
- increment current index of this list (move to right one)

Merge Operation: Combining Two Sorted Lists Into One Sorted List

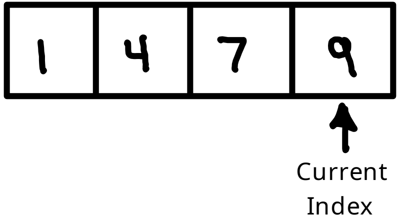
Approach:

compare: find input list whose current item is smallest

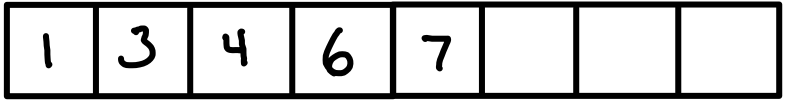
- move item to final list
- increment current index of this initial list

repeat above until one initial list is out of items, then place all items in other list into output list (in same order)

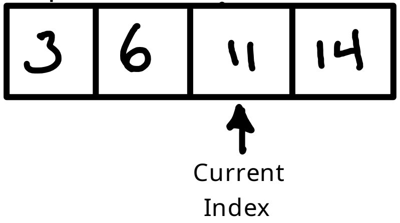
Input List A:



Output List:



Input List B:



Next Step:

Since 9 < 11, list A has smaller item:

- move this list's current item into the output list
- increment current index of this list (move to right one)

Merge Operation: Combining Two Sorted Lists Into One Sorted List

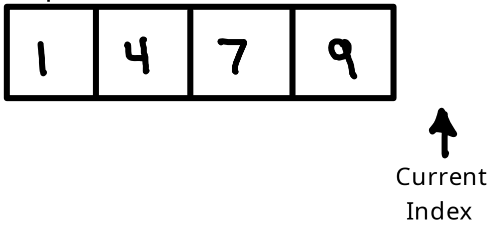
Approach:

compare: find input list whose current item is smallest

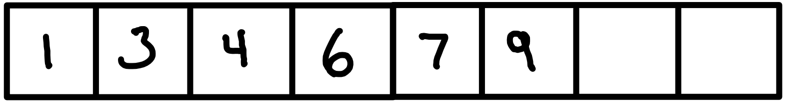
- move item to final list
- increment current index of this initial list

repeat above until one initial list is out of items, then place all items in other list into output list (in same order)

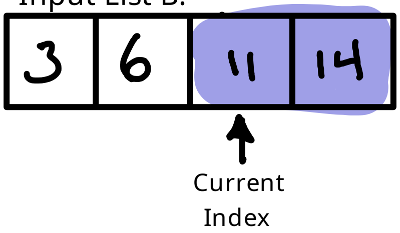
Input List A:



Output List:



Input List B:



Next Step:

We've exhausted one input list (A).

Move all remaining items in other list (B) to the output

(keep same order, they're already sorted)

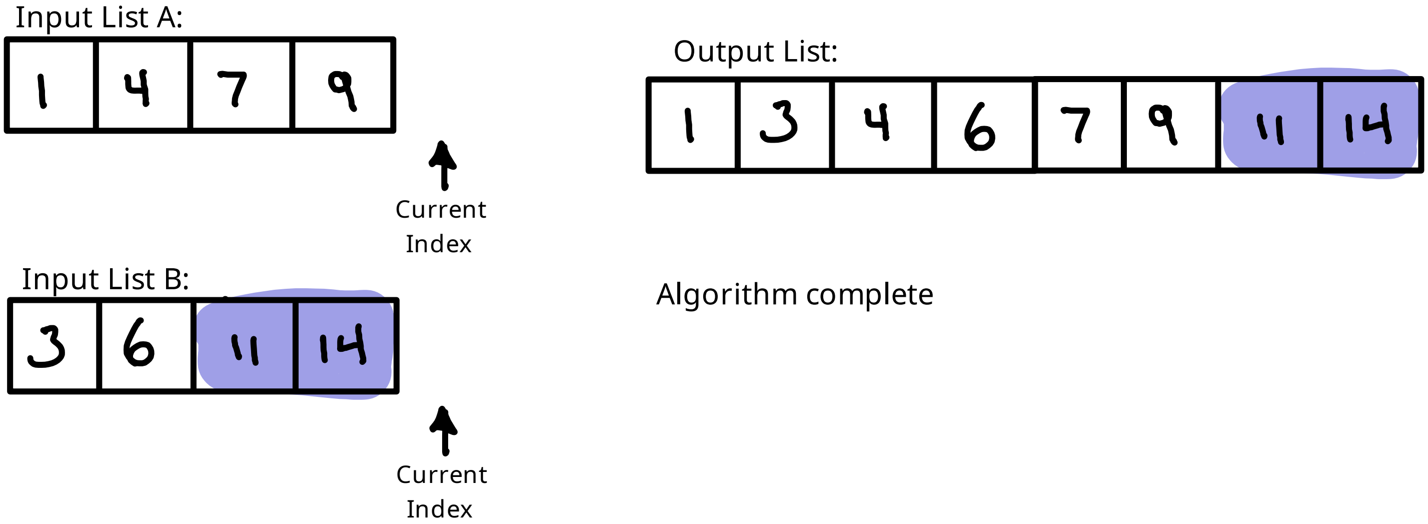
Merge Operation: Combining Two Sorted Lists Into One Sorted List

Approach:

compare: find input list whose current item is smallest

- move item to final list
- increment current index of this initial list

repeat above until one initial list is out of items, then place all items in other list into output list (in same order)



Algorithm complete

In Class Activity:

COMPARISONS: ~~||||~~ ||

Build a worst case (requiring the most comparisons) example of merge operation 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?

Input List A:

1	4	7	9

Input List B:

3	6	8	10

Output List:

1	3	4	6	7	8	9	10

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

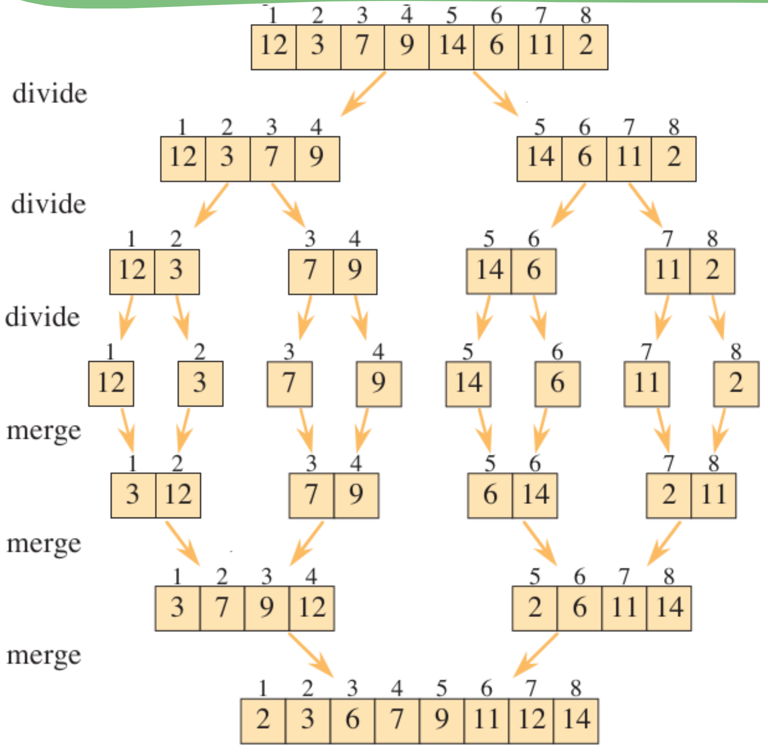
LETS IGNORE THIS -1, IT WILL SIMPLIFY ANALYSIS WITHOUT CHANGING RESULT (SLOWER BIG-O GROWTH THAN N)

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:

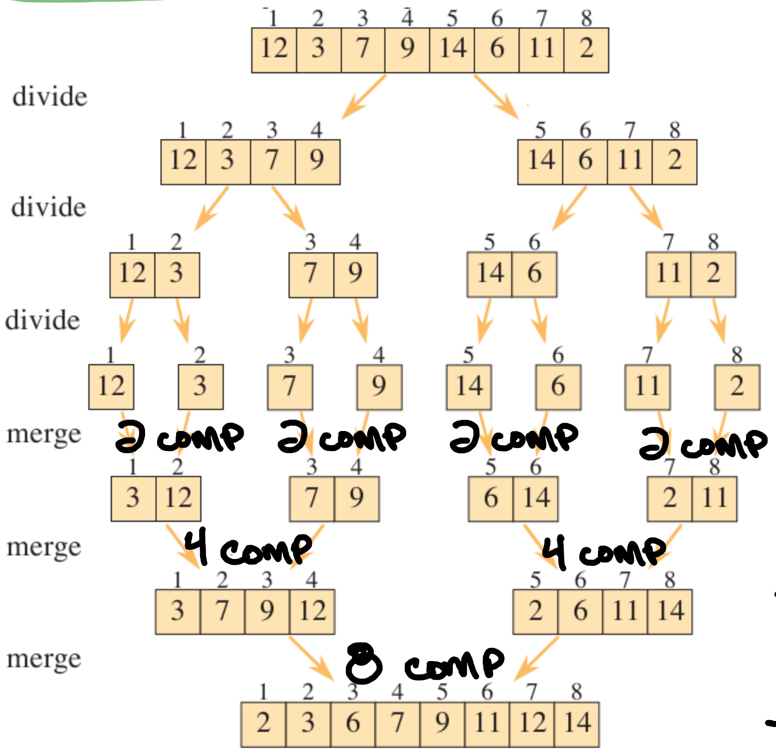
- Divide the input list in half until they're all length 1 lists (which are sorted!)

- 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 steps

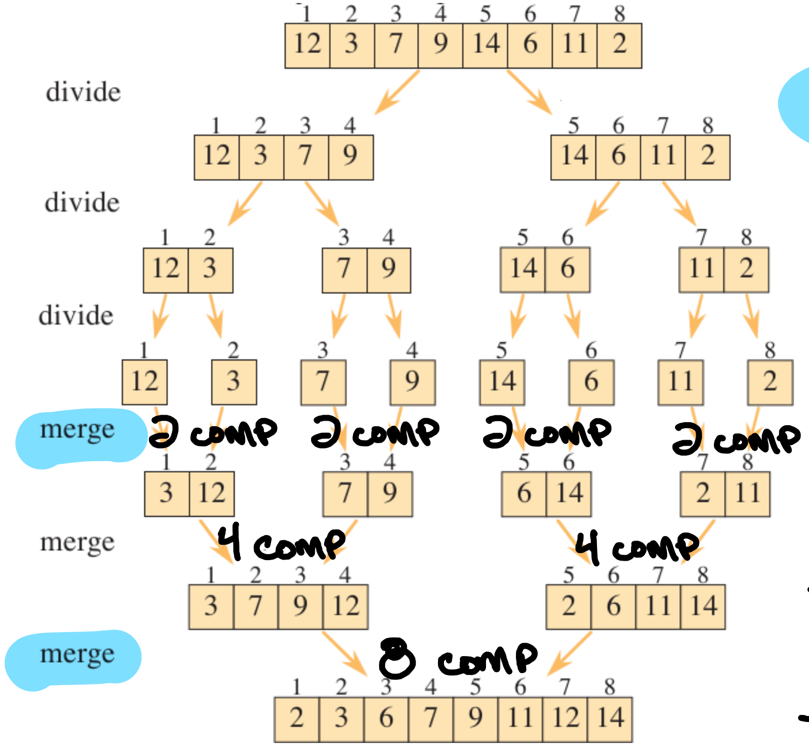
- each merge operation (in worst case) uses as many comparison as output list size (see previous slide's "punchline")

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

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

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

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



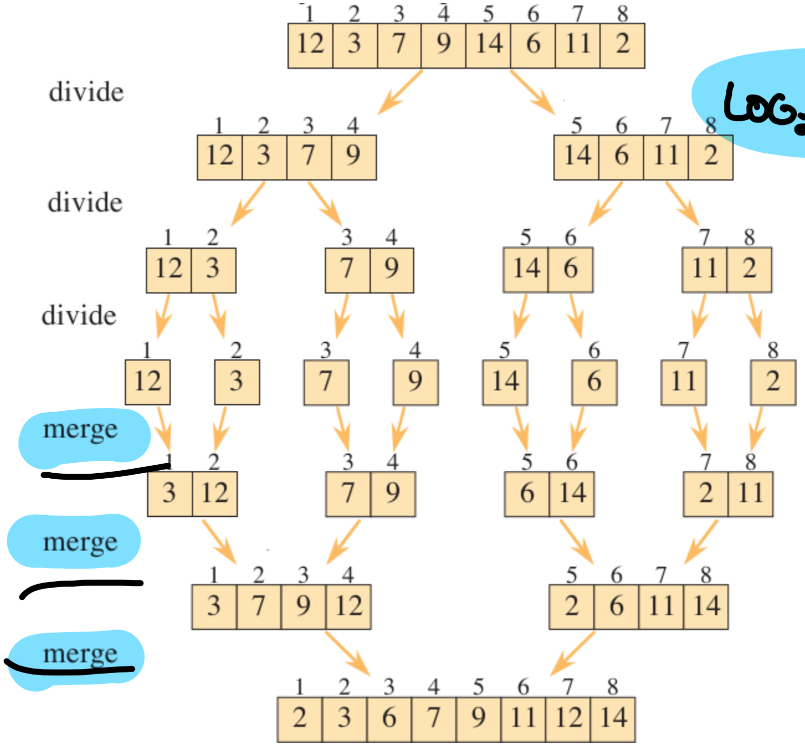
3 LEVELS MERGING $\cdot$ $8 \frac{\text{COMP}}{\text{LEVEL}}$ = 24 COMP

→ 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



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

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)

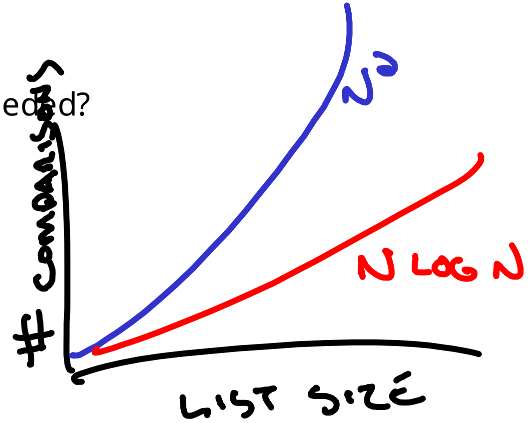
In the worst case, for an input list with n items how many comparisons are needed?

- insertion sort

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

- merge sort

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



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!)

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?)

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")

Merge Sort via Recurrences (part 1 of 2)

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

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

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

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

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

$$T(\bullet) = 2T(\bullet/2) + \bullet$$

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

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

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

Merge Sort via Recurrences (part 1 of 2)

$$
\begin{aligned}
T(n) &= 2T\left(\tfrac{n}{2}\right) + n && \leftarrow K=1 \\
&= 2\left(2T\left(\tfrac{n}{4}\right) + \tfrac{n}{2}\right) + n \\
&= 2^2\, T\left(\tfrac{n}{4}\right) + 2n && \leftarrow K=2 \\
&= 2^2\left(2T(n/8) + n/4\right) + 2n \\
&= 2^3\, T(n/8) + 3n && \leftarrow K=3 \\
&= 2^K\, T(n/2^K) + Kn
\end{aligned}
$$

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

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

$$T(1) = 0 \longleftarrow$$ It takes 0 operations to sort a list of size 1 (its already sorted, right?)

To help us build intuition, lets substitute forward … (in practice, this is less useful, but great for our intuition now)

A helpful insight about merge sort

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

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

$$\to \boxed{T(1) = 0} \leftarrow$$

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

To help us build intuition, lets substitute forward ... (in practice, this is less useful, but great for our intuition now)

$$T(2) = 2 \cdot T\left(\frac{2}{2}\right) + 2 = 2 \cdot 0 + 2 = 2$$

$$T(4) = 2 \cdot T\left(\frac{4}{2}\right) + 4 = 2 \cdot 2 + 4 = 6$$

$$T(8) = 2 \cdot T\left(\frac{8}{2}\right) + 8 = 2 \cdot 6 + 8 = 20$$

Merge Sort via Recurrences (part 2 of 2)

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

Goal: find the k (number of substitutions) so that n / 2^k = 1. (We can then apply our base case: T(1) = 0)

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

Plug that k in to "break" the recurrence relation (i.e. find a non-recurrence expression for T(n)):

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

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

$$= n \log_2 n \quad \leftarrow \text{SAME AS BEFORE}$$

This method of solving recurrences is called "substitution method"

(There are others, but this is the only one we'll study in CS1800)

In Class Activity:

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

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

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

$$T(n) = T(n-1) + 1 \leftarrow k=1$$
$$= T(n-2) + 1 + 1$$
$$= T(n-2) + 2 \leftarrow k=2$$
$$= T(n-3) + 1 + 2$$
$$= T(n-3) + 3 \leftarrow k=3$$
$$= \dots$$

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

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

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

WHAT K BRINGS ME TO BASE CASE?

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

$n - K = 1$ $\quad K = n - 1$

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

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

T(n) = 7 * T(n-2) where T(0) = 1

solution: T(n) = 7^{n/2}

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

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

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

$$T(n) = 7\,T(n-2) \leftarrow k=1$$

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

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

$$= 7^2\big(7\,T(n-6)\big)$$

$$= 7^3\,T(n-6) \leftarrow k=3$$

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

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!

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

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

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