1) Admin

- Hw 8 due today
- Hw 9 due next Tuesday
- Exam3 next Tuesday

2) Review

3) Merge Sort

4) Recurrences

Professor Hamlin
Day 21

Review

Search		Sorting	
Linear	$O(n)$	Insertion	$O(n^2)$
Binary	$O(\log_2 n)$		

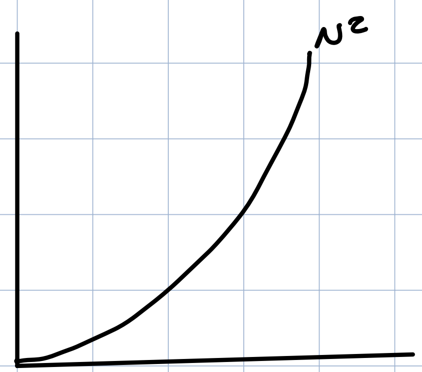
Runtime: # of comparison in worst case

Exercise How many comparisons needed to sort the list below? insertion sort

2 3 4 8 7 6 8 comparisons

Recall Insertion sort requires $O(n^2)$ runtime

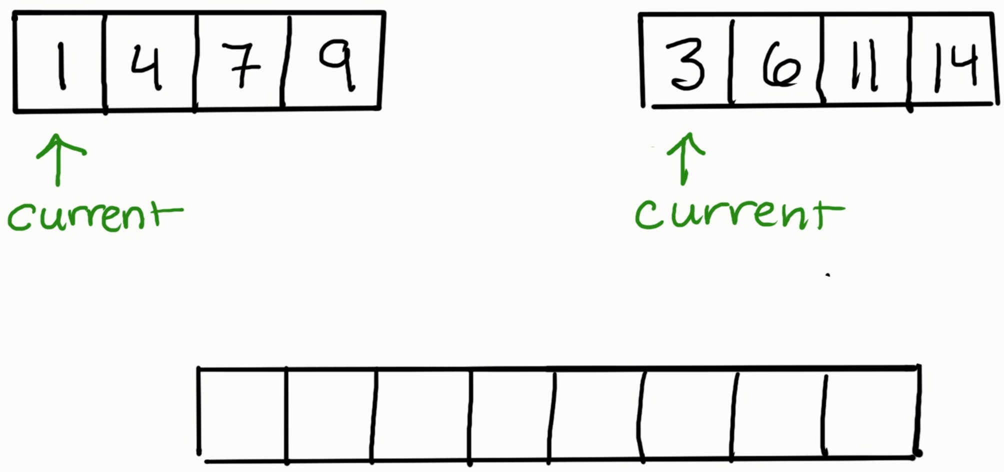
$$T(n) = n^2$$



Can we do better?

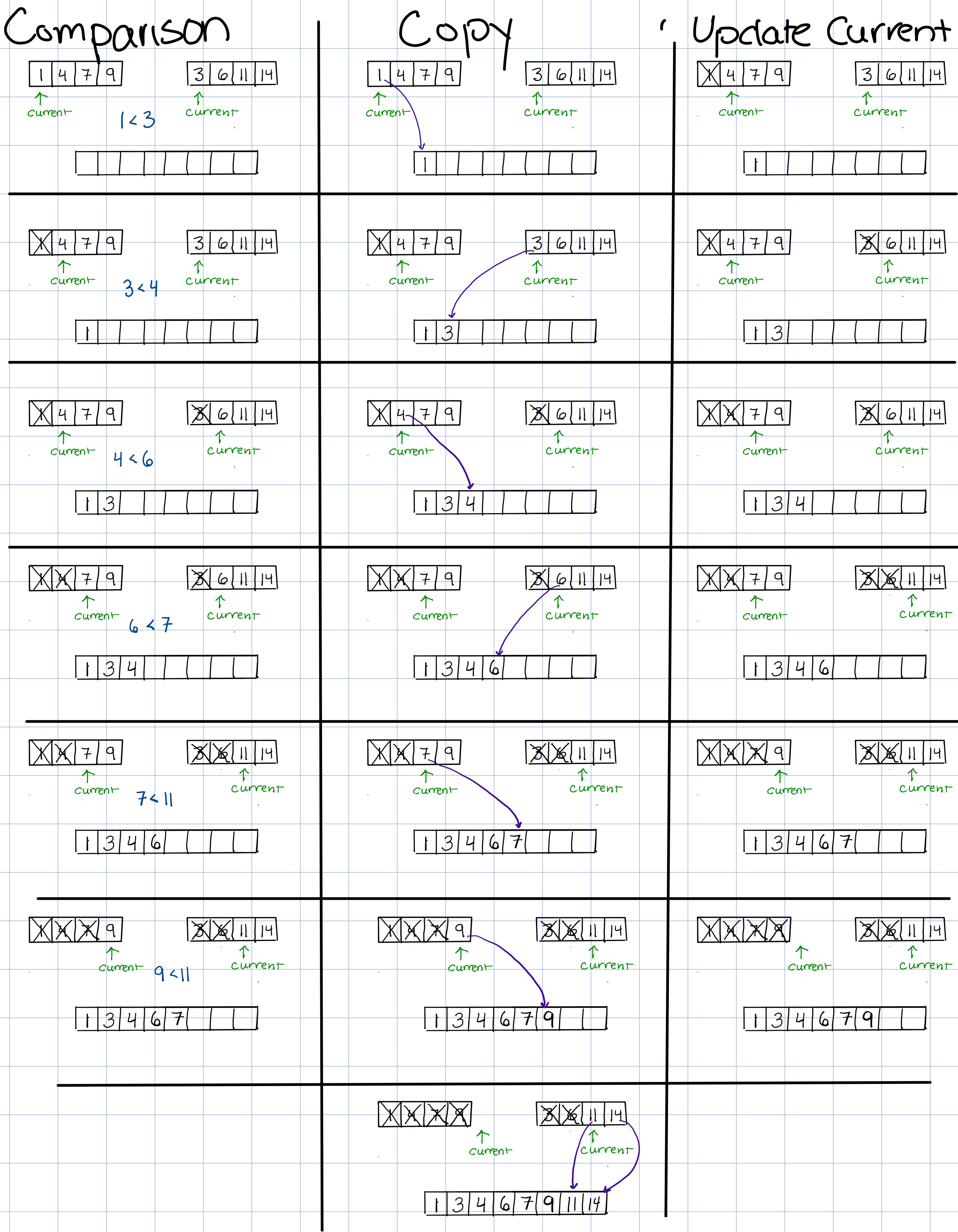
Merge Sort

Basic Building Block: merging sorted lists



1) Find current smallest, move to final list
2) increment current

Repeat until one list is gone → then just place all others into final list



Note: when reaching 11, 14 left can just add them directly

Exercise: Build a worst case (the most comparisons) of merging two sorted lists of length 4.

(How many comparisons for two length $n/2$ lists?)

$\boxed{1\ 3\ 5\ 7}$ $\boxed{2\ 4\ 6\ 8}$ 7

$\underline{1}<2$, $3>\underline{2}$, $\underline{3}<4$, $5>\underline{4}$, $\underline{5}<6$, $7>\underline{6}$, $\underline{7}<8$, 8 free!

7 comparisons

$5 \to 9$

$6 \to 11$

$\boxed{N-1}$

Worst Case: merging two sorted lists

Every comparison moves one element to final list

If one list runs out no comparisons needed for elements left in other list!

Worst case: $N/2$ length (N total)

$\boxed{N-1 \text{ comparisons}}$

Last element we get for "free"!

But let's just say $O(N)$ comparisons to merge two $N/2$ length sorted lists!

How do we use this to sort?

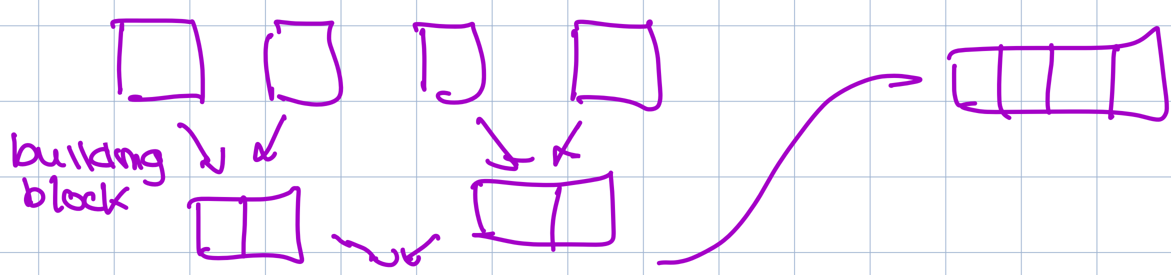
Remember from insertion sort, a single element is sorted!

We know how to merge sorted lists!

12	3	7	9	14	6	11	2

Approach: divide list in half until all length 1 (they are sorted!)

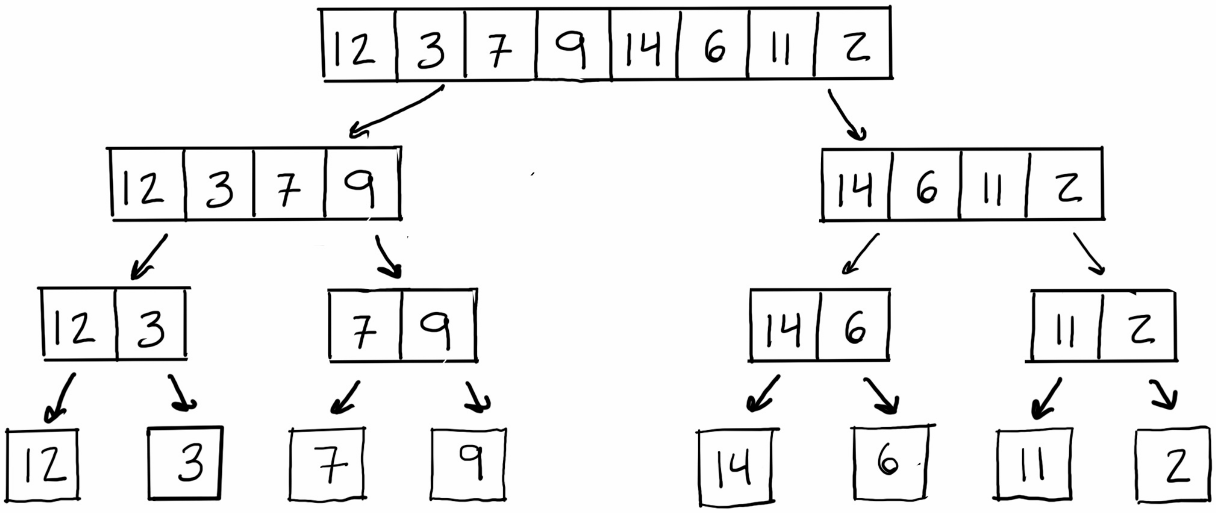
Then merge sorted lists back together with our earlier technique



divide

divide

divide



merge

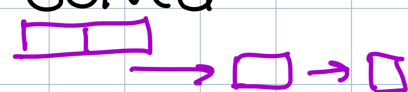
merge

merge

Observations

1) This is a type of algorthm knowns as divide and conquer; divide until sub problems are easily solved

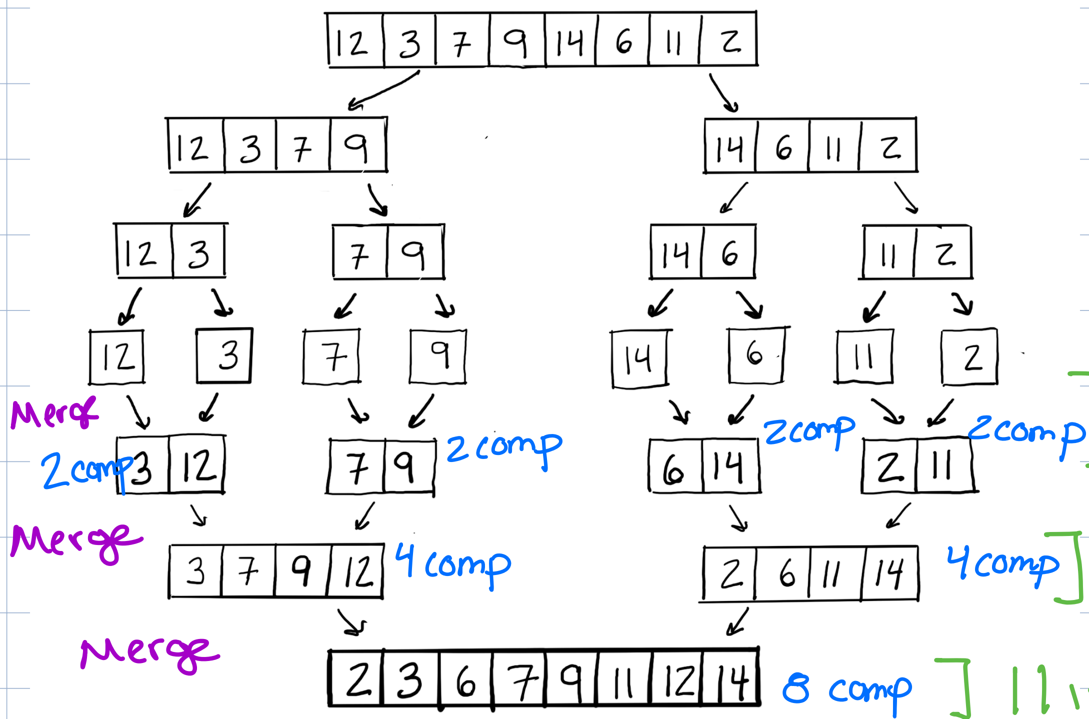
Binary Search



2) Divide steps require no comparisons

3) Each merge step in worst case requires $O(N)$ comparisons

(see next page)



4 list · 2 comp = 8 comp

2 list · 4 comp = 8 comp

1 list · 8 comp = 8 comp

Total runtime:

3 levels merging · 8 comparisons per level = $\boxed{24 \text{ comp}}$

$\boxed{24}$

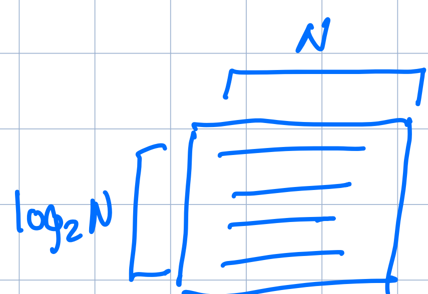
$1 \to 2 \to 4 \to 8 \to 16$

For $N = 16$: 4 levels · 16 comp per level = $\boxed{64}$

For general N:

$\log_2 N$ levels merging · N comparisons per level = $N \log N$ comp

$2^0 \to 2^1 \to 2^2 \to 2^x = N$



Mergesort has runtime of: $\Theta(n \log n)$

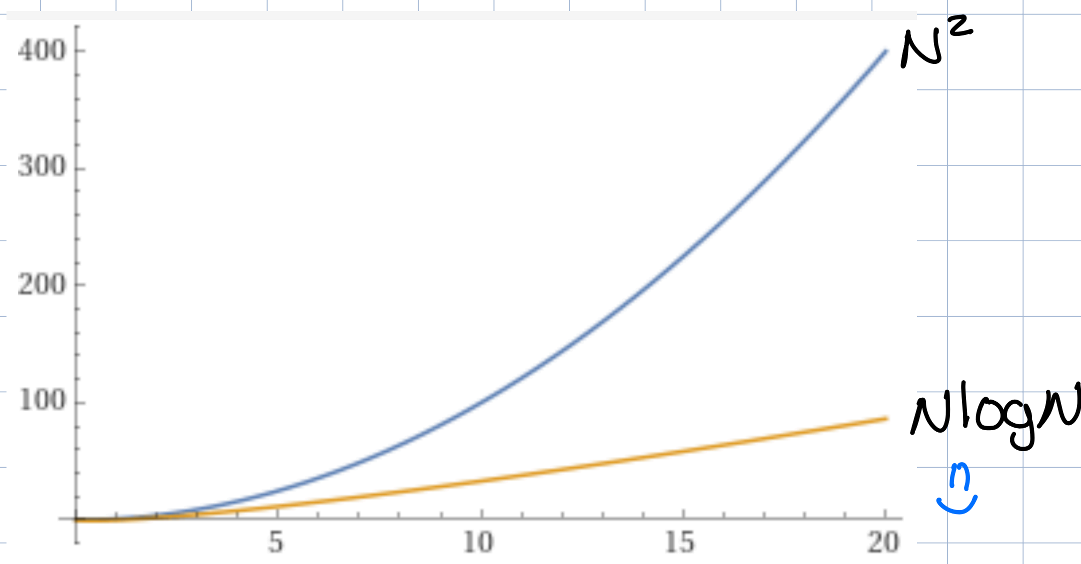
Comparing sorting algorithms

Insertion Sort:

$O(N^2)$

MergeSort:

$O(N\log N)$



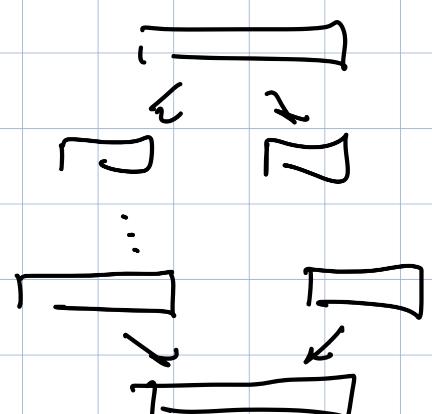
Recurrence Relations

Another way of analyzing mergesort's runtime

Why bother? Because other divide and conquer algorithms are harder to think about than mergesort

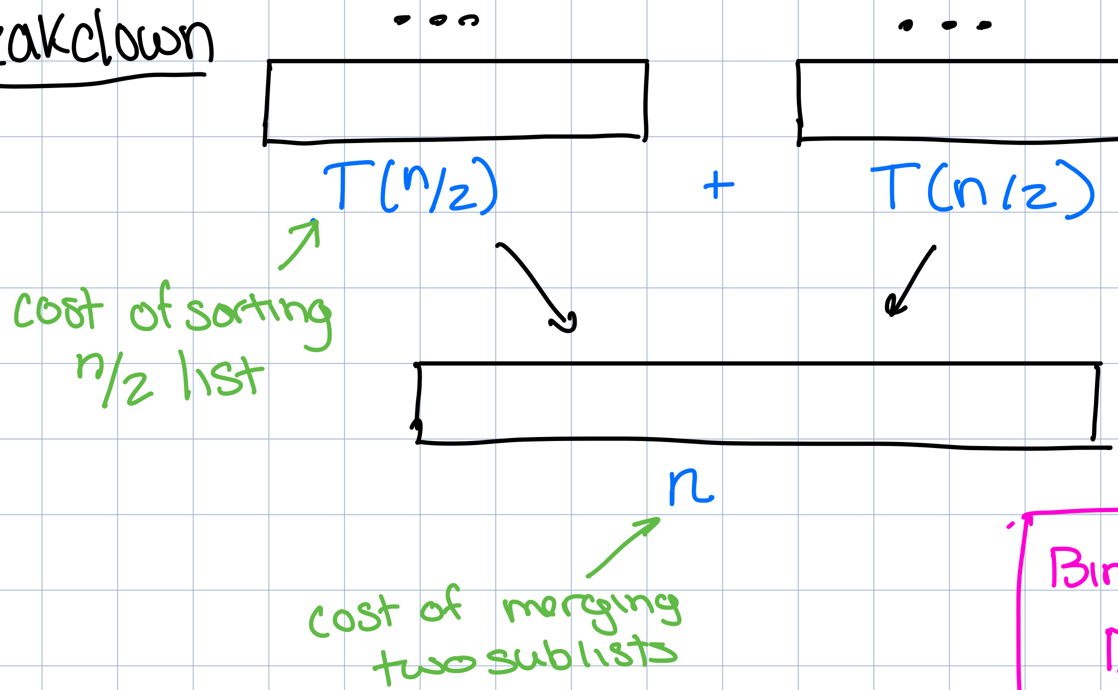
$$T(n) = 2T(n/2) + n$$

Intuition:



Call $T(n)$ the cost of doing merge sort on list of size n. Can break down this cost ...

Breakdown

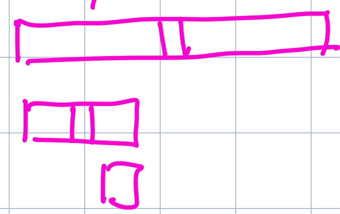


$$T(n) = T(n/2) + T(n/2) + n$$

~or~

$$\boxed{T(n) = 2T(n/2) + n}$$

Binary Search



1. Comparison
2. Run binary search on half

$$T(n) = 1 + T(\tfrac{n}{2})$$

This is a recurrance relation, each item is expressed in terms of previous item

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

This doesn't exactly give us a runtime though.

How can we get an expression w/ $T(\cdot)$ only on one side? (closed form)

Substitution Method

aka... expand a few layers and notice a pattern

Expanding one layer:

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

What is $T(n/2)$:

For clarity call $x = n/2$

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

Now if we plug $n/2$ for x

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

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

can substitute in $T(n/2)$:

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

$$T(n) = 2^2\, T(n/4) + 2n$$

Expanding Second layer:

$$T(n) = 2^2\, T(n/4) + 2n$$

What is $T(n/4)$?

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

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

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

Now let's see if we put our expressions side by side if we spot a pattern...

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

$$T(n) = 2^2 T(n/4) + 2n$$ two expansions

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

More formally:

$$T(n) = 2^1 T(n/2) + 1n$$ one expansion
$k=1$

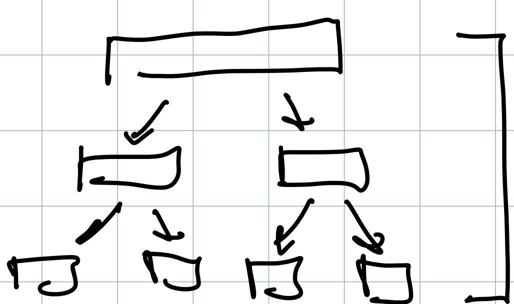
$$T(n) = 2^2 T(n/4) + 2n$$ two expansions
$k=2$

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

$$= 2^k T(n/2^k) + kn$$

Okay but what is k?

k is the number of expansions we can do before we reach the "bottom" or base case



In merge sort we can only divide list until they are 1 element!

What is our base case? And how much work do we do there?

In Mergesort, our smallest list is size 1 and it is already sorted so we do no work e.g.

$$T(1) = 0$$

Solving for k:

Once we have our base case we can find how many substitutions we can make before reaching it

$$T(n) = 2^k T(n/2^k) + kn$$

aka what value of k makes this $= 1$

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

So lets eliminate k in our previous expression:

$$\begin{aligned} T(n) &= 2^k T(n/2^k) + kn \\ &= 2^{\log_2 n} T(n/2^{\log_2 n}) + n \log_2 n \\ &= n \cdot T(1) + n \log_2 n \\ &= n \log_2 n \end{aligned}$$

we know $T(1) = 0$!

same as before!

Condensed solution for reference:

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

$$T(n) = 2(2T(n/4) + \frac{n}{2}) + n$$
$$= 2^2 T(n/4) + 2n \qquad T(n/4) = 2T(n/8) + n/4$$

$$T(n) = 2^2(2T(n/8) + n/4) + 2n$$
$$= 2^3 T(n/8) + 3n$$

After k substitutions:

$$T(n) = 2^k T(n/2^k) + kn$$

Solving for k when $T(1) = 0$:

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

Substituting k back in:

$$T(n) = 2^k T(n/2^k) + kn$$
$$= 2^{\log_2 n} T(n/2^{\log_2 n}) + n\log_2 n$$
$$= n \cdot T(1) + n \log n$$
$$= n \log_2 n$$

$$\boxed{T(n) = n\log_2 n}$$

Exercise:

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

$\boxed{T(n) = n}$

$$T(n) = \overbrace{T(n-1)} + 1$$
$$T(n) = (T(n-2) + 1) + 1$$
$$= \underbrace{T(n-2)} + 2$$
$$= (T(n-3) + 1) + 2$$
$$= T(n-3) + 3$$

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

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

Solve for k

$$n - k = 1$$
$$k = n - 1$$

Plug in k

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

Recurrence for linear:

$$T(n) = \overbrace{1}^{1} + \overbrace{T(n-1)}^{\text{Linear}}$$

2) $T(n) = T(n-3) + 4$, $T(1) = 1$ $T(n) = \frac{4n-1}{3}$

$T(n) = T(n-3) + 4$ $T(n-3) = T(n-6) + 4$

$T(n) = T(n-6) + 4 + 4$

$= T(n-2\cdot 3) + 2\cdot 4$

$T(n) = T(n-6) + 2\cdot 4$ $T(n-6) = T(n-9) + 4$

$= (T(n-9) + 4) + 2\cdot 4$

$T(n) = T(n-3k) + 4k$

Solve for k:

$n - 3k = 1 \Rightarrow 3k = n-1 \Rightarrow k = \frac{n-1}{3}$

Plug back in

$$T(n) = T\left(n - 3\left(\frac{n-1}{3}\right)\right) + 4\left(\frac{n-1}{3}\right)$$

$$= T(n - n + 1) + 4\left(\frac{n-1}{3}\right)$$

$$= 1 + 4\left(\frac{n-1}{3}\right)$$

$$= \frac{4n-4}{3} + \frac{3}{3}$$

$$= \boxed{\frac{4n-1}{3}}$$

3) $T(n) = 7T(n-2)$ $\quad T(0)=1$ $\quad T(n) = 7^{n/2}$

$$T(n) = 7T(n-2)$$
$$= 7\cdot(7T(n-4))$$
$$= 7^2 T(n-4)$$
$$= 7^2(7T(n-6))$$
$$= 7^3 T(n-6)$$

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

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

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

Solve for k:

$$n-2k=0 \Rightarrow k = n/2$$

Plug back in:

$$T(n) = 7^{n/2} T(n-2(\tfrac{n}{2}))$$
$$7^{n/2}\cdot 1$$
$$\boxed{= 7^{n/2}}$$