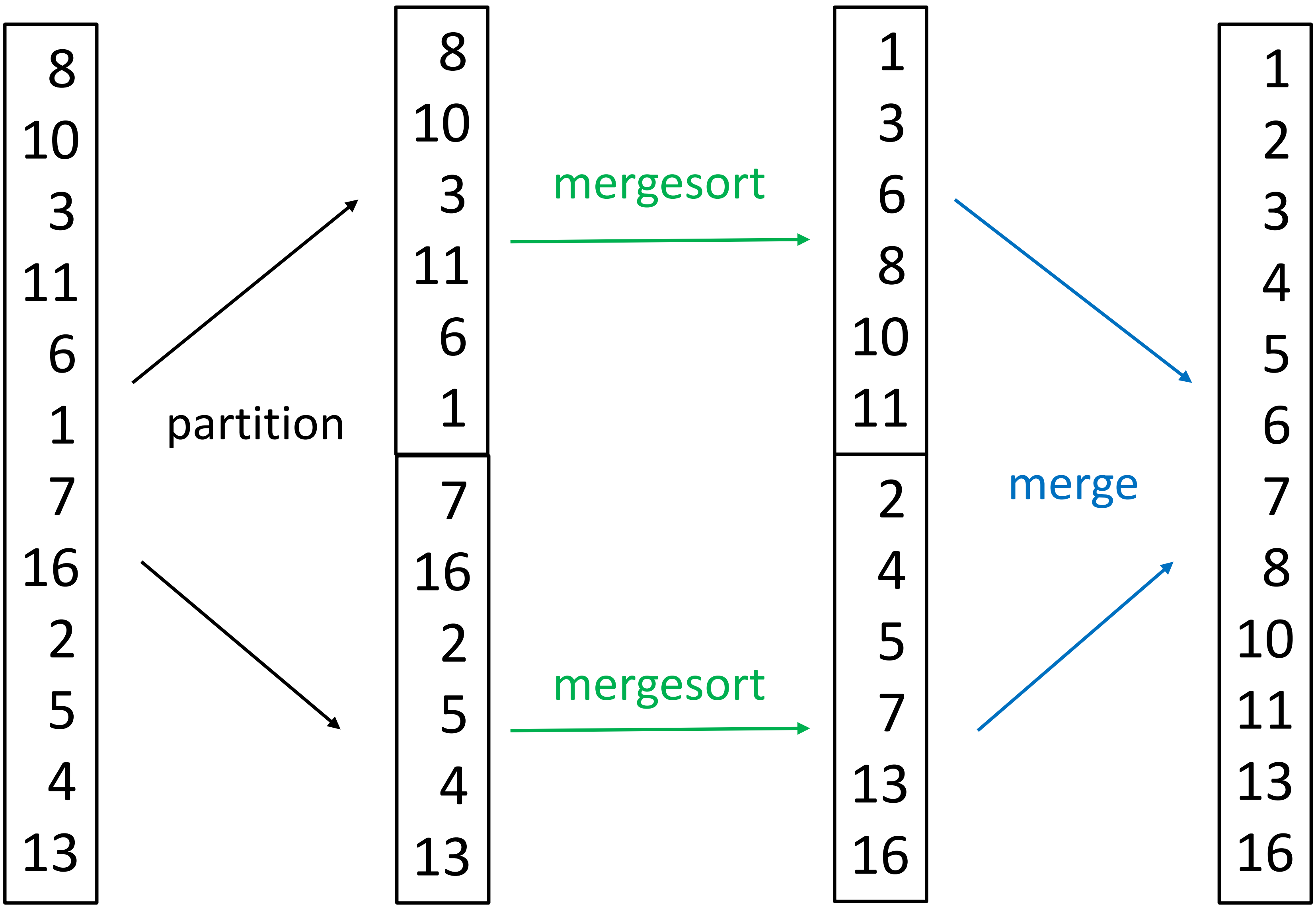
COMP 250

Lecture 34

recurrences 2: mergesort & quicksort

Nov. 28, 2018

Example 5: Mergesort



```
mergesort(list){
   if  list.length == 1
      return list
   else{
      mid = (list.size - 1) / 2
      list1 =  list.getElements(0,mid)
      list2 =  list.getElements(mid+1, list.size-1)
      list1  =  mergesort(list1)
      list2  =  mergesort (list2)
      return   merge( list1, list2 )
   }
}
```

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

We are ignoring a constant term for simplicity.

```
mergesort(list){
    if  list.length == 1
       return list
    else{
       mid = (list.size - 1) / 2
       list1 =  list.getElements(0,mid)
       list2 =  list.getElements(mid+1, list.size-1)
       list1  =  mergesort(list1)
       list2  =  mergesort (list2)
       return   merge( list1, list2 )
    }
}
```

Base case   n = 1

getElements could take time proportional to n.

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

We are ignoring a constant term for simplicity.

Recall same issue with binary search last lecture:

What if n is not even ?

e.g. $t(13) = c * 13 + t(6) + t(7)$

In general, one should write the recurrence as:

$$t(n) = c\,n + t\left(floor\left(\frac{n}{2}\right)\right) + t\left(ceiling\left(\frac{n}{2}\right)\right)$$

round down   round up

In COMP 250, one typically assumes $n = 2^k$ for recurrences that involve $t(\frac{n}{2})$.
The more general recurrence has roughly the same solution.

$$t(n) \quad = \quad c\ n + 2\ t(\frac{n}{2})$$

$$
\begin{aligned}
t(n) &= c\ n + 2\ t(\frac{n}{2}) \\
&= c\ n + 2\ (\ c\frac{n}{2} + \ 2t(\frac{n}{4}))
\end{aligned}
$$

$$
\begin{aligned}
t(n) &= c\ n + 2\ t(\frac{n}{2}) \\
&= c\ n + 2\ (\ c\frac{n}{2} +\ 2t(\frac{n}{4})) \\
&= c\ n + c\ n + 4\ t(\frac{n}{4})
\end{aligned}
$$

$$
\begin{aligned}
t(n) &= c\;n + 2\;t(\frac{n}{2}) \\
&= c\;n + 2\;(\;c\frac{n}{2} + \;2t(\frac{n}{4})) \\
&= c\;n + c\;n + 4\;t(\frac{n}{4}) \\
&= c\;n + c\;n + 4\;(c\frac{n}{4}\; + \;2\;t(\;\frac{n}{8}\;))
\end{aligned}
$$

$$
\begin{aligned}
t(n) &= c\,n + 2\;t(\frac{n}{2}) \\
&= c\,n + 2\;(\;c\frac{n}{2} + \;2t(\frac{n}{4})) \\
&= c\,n + c\,n + 4\;t(\frac{n}{4}) \\
&= c\,n + c\,n + 4\;(c\frac{n}{4}\; + \;2\;t(\;\frac{n}{8}\;)) \\
&= c\,n + \;c\,n + \;c\,n + 8\;t(\frac{n}{8})
\end{aligned}
$$

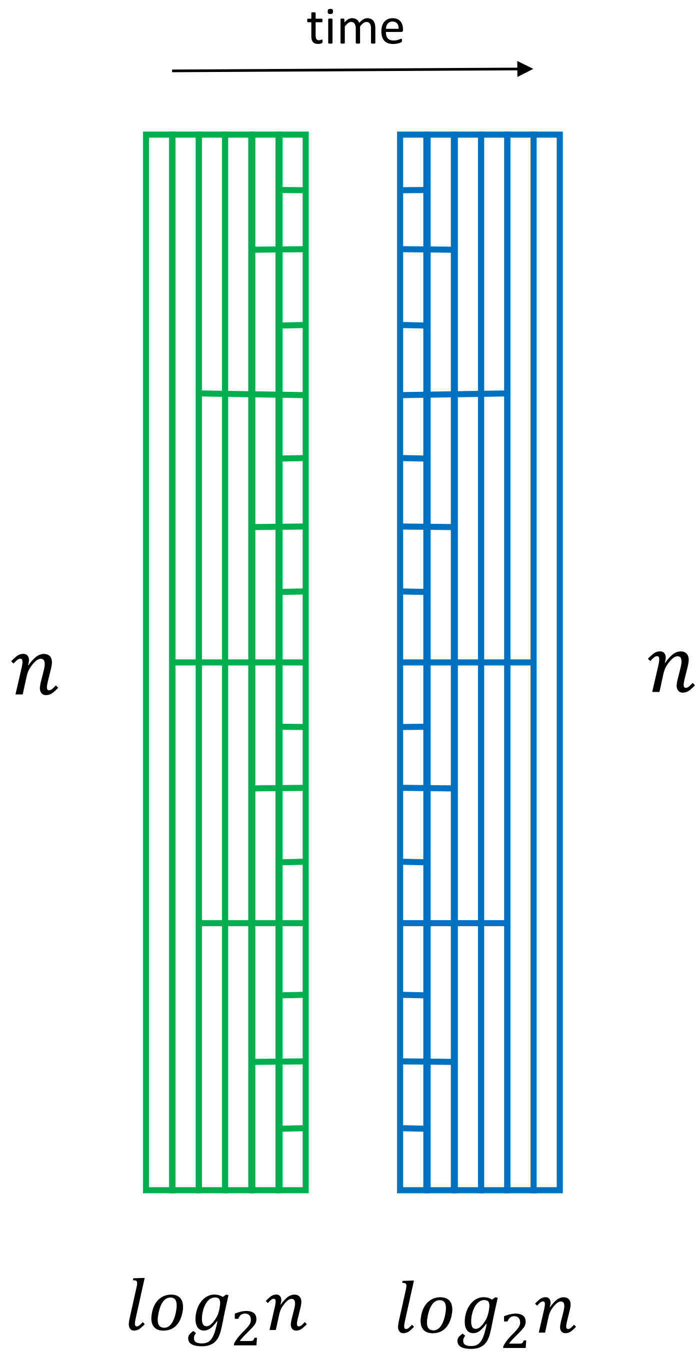
$$
\begin{aligned}
t(n) &= c\ n + 2\ t(\frac{n}{2}) \\
&= c\ n + 2\ (\ c\frac{n}{2} +\ 2t(\frac{n}{4})) \\
&= c\ n + c\ n + 4\ t(\frac{n}{4}) \\
&= c\ n + c\ n + 4\ (c\frac{n}{4}\ +\ 2\ t(\ \frac{n}{8}\ )) \\
&= c\ n +\ c\ n +\ c\ n + 8\ t(\frac{n}{8}) \\
&= c\ n\ k\ +\ 2^k\ t(\frac{n}{2^k})
\end{aligned}
$$

$$
\begin{aligned}
t(n) &= c\,n + 2\;t(\frac{n}{2}) \\
&= c\,n + 2\;(\;c\frac{n}{2} + \;2t(\frac{n}{4})) \\
&= c\,n + c\,n + 4\;t(\frac{n}{4}) \\
&= c\,n + c\,n + 4\;(c\frac{n}{4}\; + \;2\;t(\;\frac{n}{8}\;)) \\
&= c\,n + \;c\,n + \;c\,n + 8\;t(\frac{n}{8}) \\
&= c\,n\,k\;+\;2^k\;t(\frac{n}{2^k}) \\
&= c\,n\log_2 n + \;n\;t(1), \quad \text{when } n = 2^k
\end{aligned}
$$

Base case

```
mergesort(list){
   if  list.length == 1
      return list
   else{
      mid = (list.size - 1) / 2
      list1 =  list.getElements(0,mid)
      list2 =  list.getElements(mid+1, list.size-1)
      list1  =  mergesort(list1)
      list2  =  mergesort (list2)
      return   merge( list1, list2 )
   }
}
```

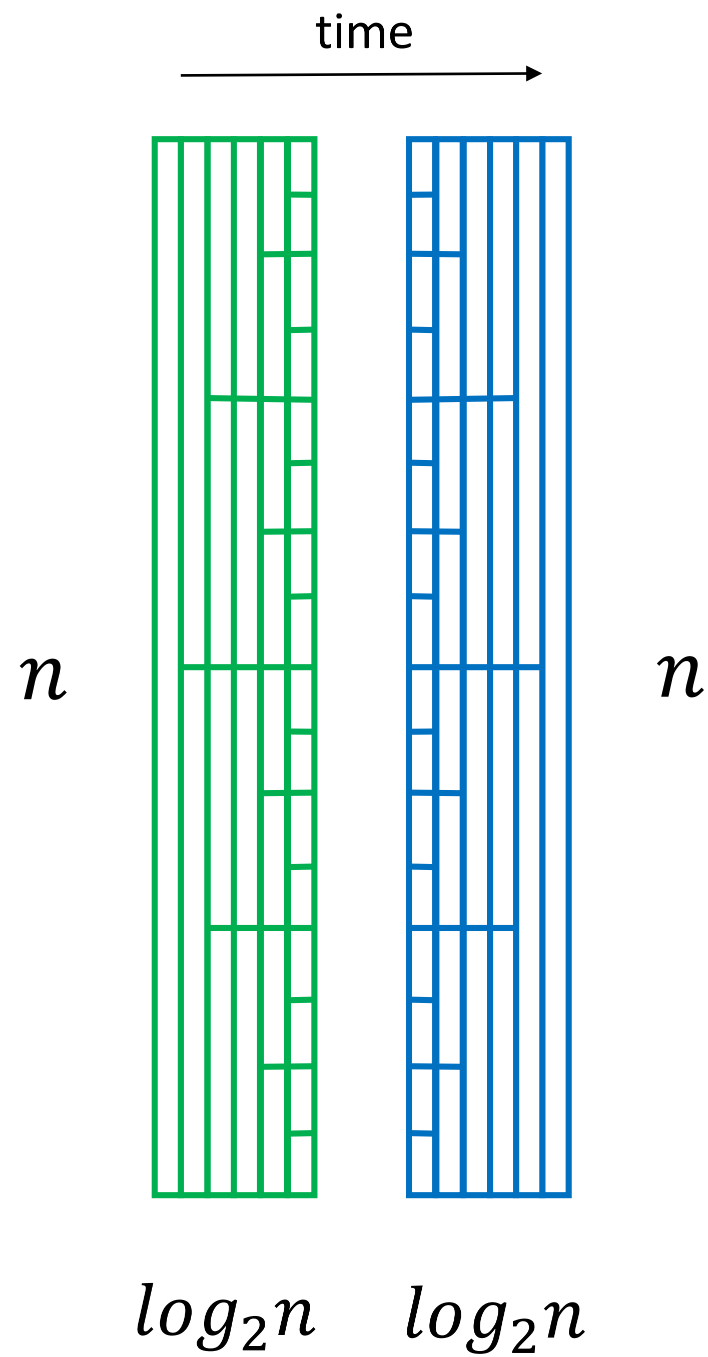
Q: How many recursive calls are made to mergesort?



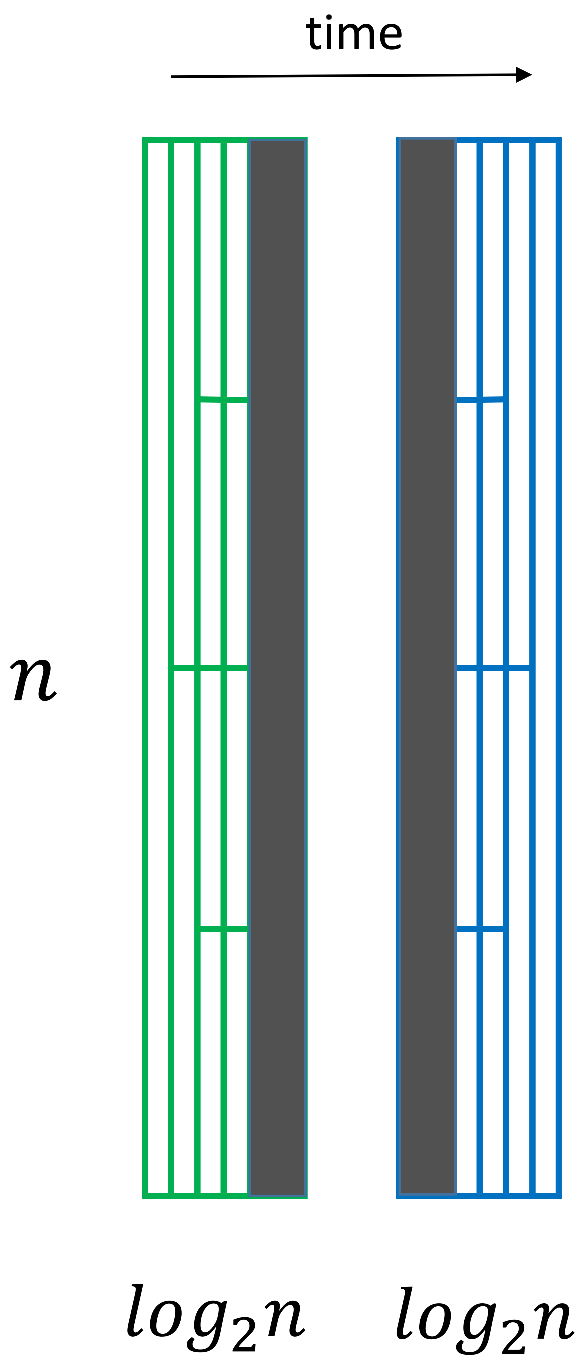
```
mergesort(list){
   if  list.length == 1
      return list
   else{
      mid = (list.size - 1) / 2
      list1 =  list.getElements(0,mid)
      list2 =  list.getElements(mid+1, list.size-1)
      list1  =  mergesort(list1)
      list2  =  mergesort (list2)
      return   merge( list1, list2 )
   }
}
```

Q: How many recursive calls are made to mergesort?

A: $n - 1$.



What if we change the base case and stop the recursion at a larger list size $n_0 > 1$?



For example, if we stop at $n_0 > 4$, then we don't have to do the recursions in the grayed out part.

But we still need to sort and merge.

How does that change the total time?

```
mergesort(list){
   if  list.length <  5                    //   or some other base case
      return bubblesort(list)
   else{
      mid = (list.size - 1) / 2
      list1 =  list.getElements(0,mid)
      list2 =  list.getElements(mid+1, list.size-1)
      list1  =  mergesort(list1)
      list2  =  mergesort (list2)
      return   merge( list1, list2 )
   }
}
```

$$t(n) = c\,n + 2\,t\left(\frac{n}{2}\right), \qquad n \geq 5$$

```
if  list.length <  5
       return bubbleSort(list)
```

How does this affect the back substitution and solution?

$$t(n) \;=\; c\,n + 2\,t(\frac{n}{2})$$

if list.length < 5

    return bubbleSort(list)

k times

$$
\begin{aligned}
t(n) &= c\,n + 2\,t(\frac{n}{2}) \\
&= \dots \\
&= c\,n\,k \;+\; 2^k\,t(\frac{n}{2^k})
\end{aligned}
$$

Stop back substitution when $\frac{n}{2^k} = 4$

```
if  list.length <  5
      return bubbleSort(list)
```

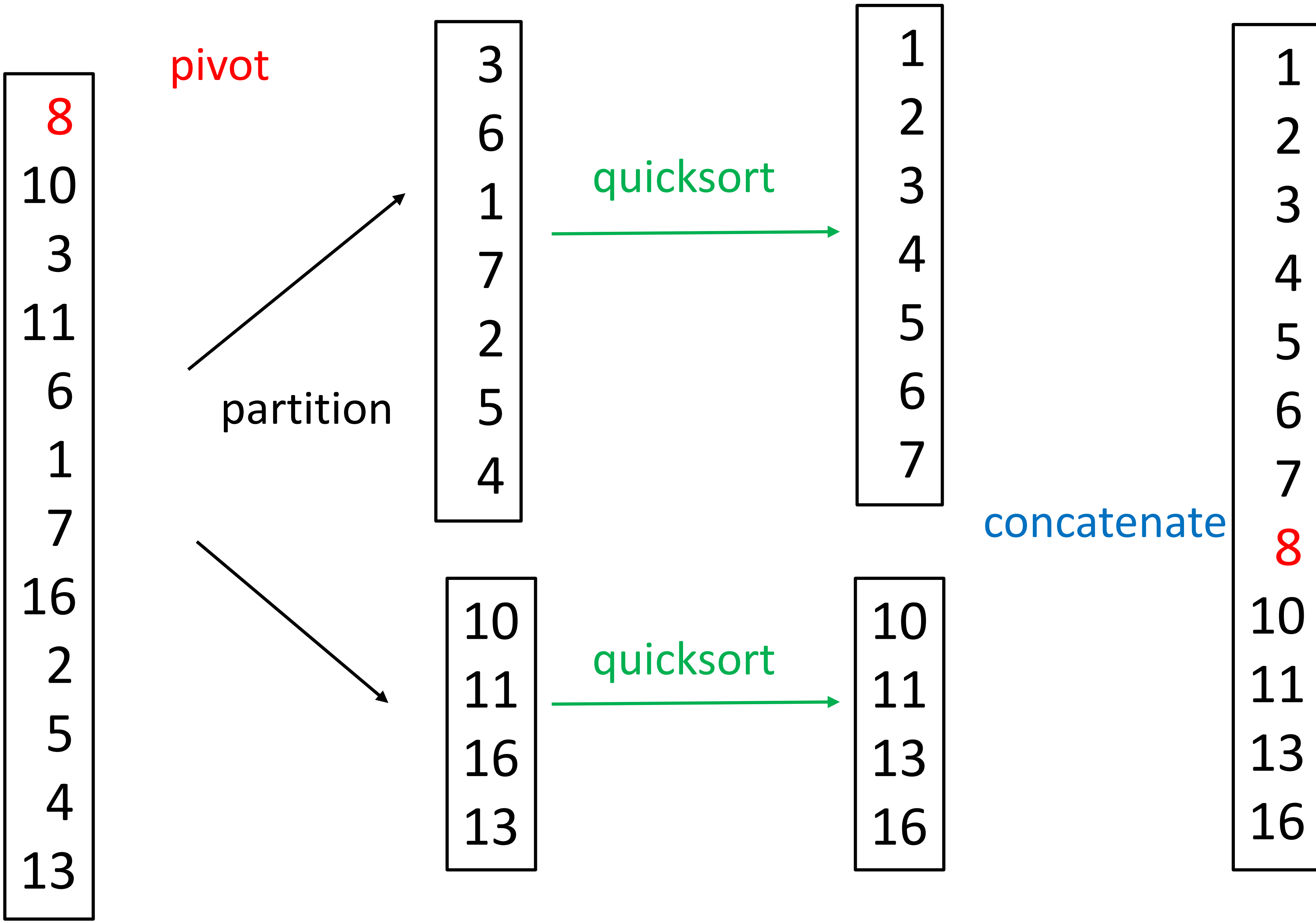
$$
\begin{aligned}
t(n) &= c\,n + 2\,t(\frac{n}{2}) \\
&= \dots \\
&= c\,n\,k \;+\; 2^k\,t(\frac{n}{2^k}), \quad \text{and letting } 2^k = \frac{n}{4} \text{ gives...} \\
&= c\,n(\log_2 n - 2) + \;\frac{n}{4}\,t(4)
\end{aligned}
$$

if list.length < 5
    return bubbleSort(list)

$$
\begin{aligned}
t(n) &= c\,n + 2\,t(\frac{n}{2}) \\
&= \dots \\
&= c\,n\,k \;+\; 2^k\,t(\frac{n}{2^k}), \quad \text{and letting } 2^k = \frac{n}{4} \text{ gives...} \\
&= c\,n(\log_2 n - 2) + \frac{n}{4}\,t(4)
\end{aligned}
$$

For large n, the dominant term is still $n\,log_2 n$.

Quicksort


pivot
8
10
3
11
6
1
7
16
2
5
4
13
partition
3
6
1
7
2
5
4
10
11
16
13
quicksort
quicksort
1
2
3
4
5
6
7
10
11
13
16
concatenate
1
2
3
4
5
6
7
8
10
11
13
16

Quicksort Best Case

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

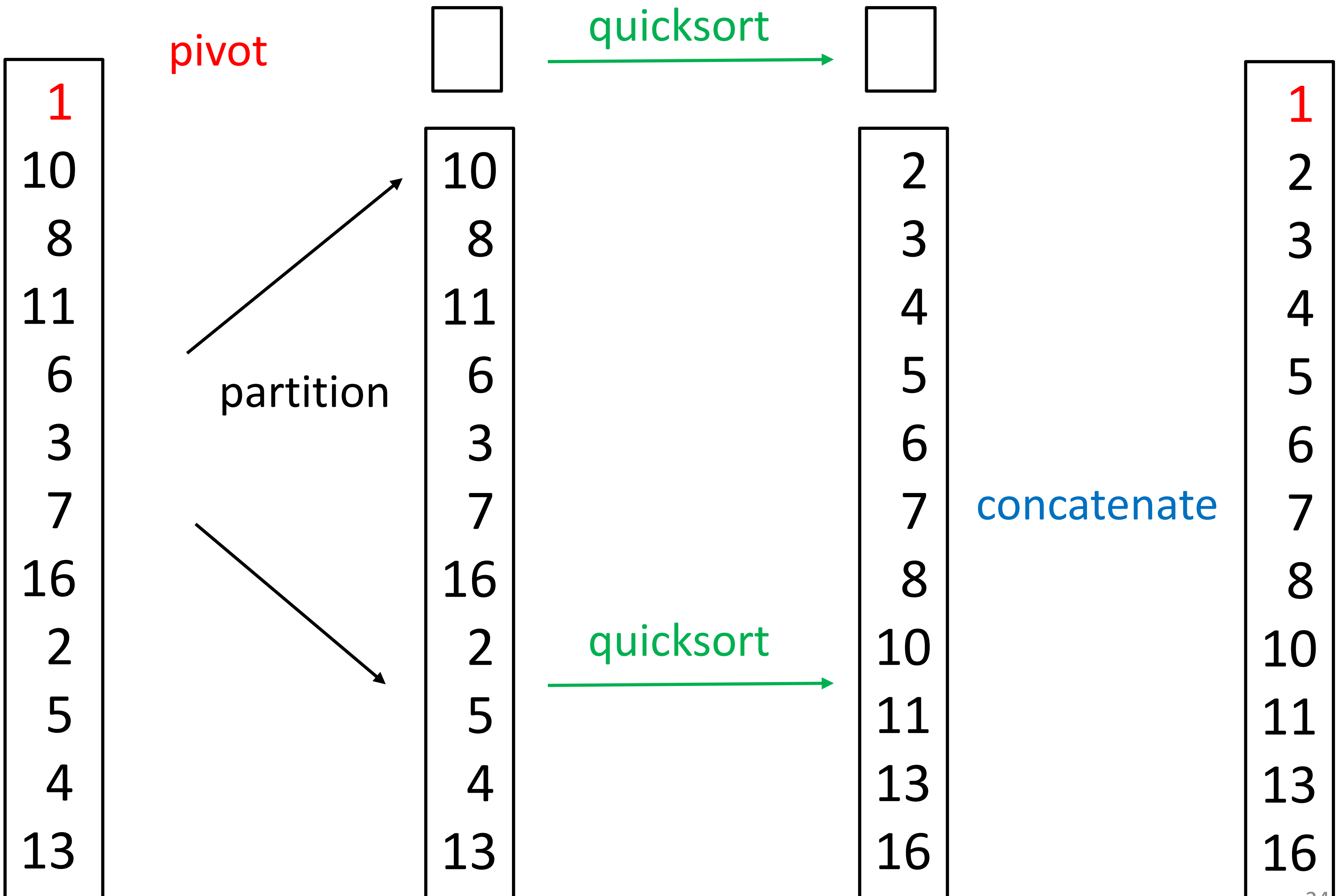
Partition based on the pivot

Two sublists have the same size

This is the same recurrence as mergesort.

Again, we are ignoring a constant term for simplicity.

Quicksort ***worst*** case ?



Quicksort worst case

$$t(n) = c\, n + t(n - 1)$$

From last lecture….

Quicksort worst case

$$t(n) = c\,n + t(n-1)$$

From last lecture....   (algorithm was similar to selection sort)

$$t(n) = c\,\frac{n\,(n+1)}{2}$$

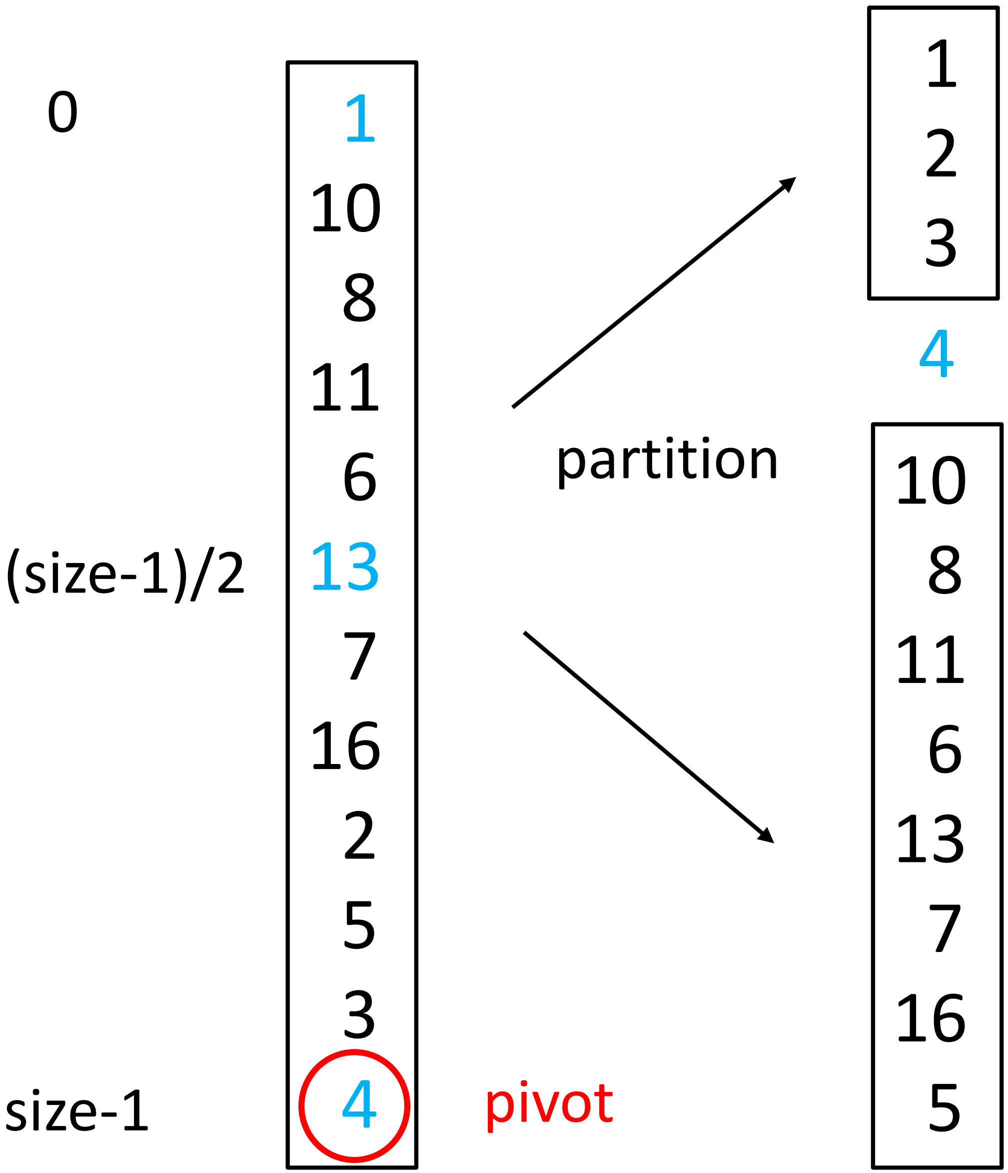
How to reduce the chance of an unbalanced partition at each step of Quicksort?

0	1
	10
	8
	11
	6
(size-1)/2	13
	7
	16
	2
	5
	3
size-1	4

"Median of three" technique:

Let the pivot be the *median* of first, middle, and last elements in list.

e.g. pivot = median(1, 13, 4) = 4



Works well in practice, especially if the list is already nearly sorted.

Example 6

$$t(n) \;=\; n + t(\frac{n}{2})$$

What do you expect?

(BTW, using cn for the first term instead of n won't change the following derivation.)

$$
\begin{aligned}
t(n) &= n + t(\frac{n}{2}) \\
&= n + \frac{n}{2} + t(\frac{n}{4})
\end{aligned}
$$

$$
\begin{aligned}
t(n) &= n + t(\frac{n}{2}) \\
&= n + \frac{n}{2} + t(\frac{n}{4}) \\
&= n + \frac{n}{2} + \frac{n}{4} + t(\frac{n}{8})
\end{aligned}
$$

$$
\begin{aligned}
t(n) &= n + t(\frac{n}{2}) \\
&= n + \frac{n}{2} + t(\frac{n}{4}) \\
&= n + \frac{n}{2} + \frac{n}{4} + t(\frac{n}{8}) \\
&= n + \frac{n}{2} + \frac{n}{4} + \ldots \frac{n}{2^{k-1}} + t(\frac{n}{2^k})
\end{aligned}
$$

$$
\begin{aligned}
t(n) &= n + t(\frac{n}{2}) \\
&= n + \frac{n}{2} + t(\frac{n}{4}) \\
&= n + \frac{n}{2} + \frac{n}{4} + t(\frac{n}{8}) \\
&= n + \frac{n}{2} + \frac{n}{4} + \ldots \frac{n}{2^{k-1}} + t(\frac{n}{2^k}) \\
&= n + \frac{n}{2} + \frac{n}{4} + \cdots + 4 + 2 + t(1), \quad \text{when } 2^k = n
\end{aligned}
$$

$$
\begin{aligned}
t(n) &= n + t(\frac{n}{2}) \\
&= n + \frac{n}{2} + t(\frac{n}{4}) \\
&= n + \frac{n}{2} + \frac{n}{4} + t(\frac{n}{8}) \\
&= n + \frac{n}{2} + \frac{n}{4} + \ldots \frac{n}{2^{k-1}} + t(\frac{n}{2^k}) \\
&= n + \frac{n}{2} + \frac{n}{4} + \cdots + 4 + 2 + t(1), \quad \text{when } 2^k = n \\
&= \underbrace{n + \frac{n}{2} + \frac{n}{4} + \cdots + 4 + 2 + 1}_{?} - 1 + t(1)
\end{aligned}
$$

$$
\begin{aligned}
t(n) &= n + t(\frac{n}{2}) \\
&= n + \frac{n}{2} + t(\frac{n}{4}) \\
&= n + \frac{n}{2} + \frac{n}{4} + t(\frac{n}{8}) \\
&= n + \frac{n}{2} + \frac{n}{4} + \ldots \frac{n}{2^{k-1}} + t(\frac{n}{2^k}) \\
&= n + \frac{n}{2} + \frac{n}{4} + \cdots + 4 + 2 + t(1), \quad \text{when } 2^k = n \\
&= \underbrace{n + \frac{n}{2} + \frac{n}{4} + \cdots + 4 + 2 + 1} - 1 + t(1)
\end{aligned}
$$

$$\sum_{i=0}^{log_2 n} 2^i = \ ?$$

$$
\begin{aligned}
t(n) &= n + t(\frac{n}{2}) \\
&= n + \frac{n}{2} + t(\frac{n}{4}) \\
&= n + \frac{n}{2} + \frac{n}{4} + t(\frac{n}{8}) \\
&= n + \frac{n}{2} + \frac{n}{4} + \dots \frac{n}{2^{k-1}} + t(\frac{n}{2^k}) \\
&= n + \frac{n}{2} + \frac{n}{4} + \cdots + 4 + 2 + t(1), \quad \text{when } 2^k = n \\
&= n + \frac{n}{2} + \frac{n}{4} + \cdots + 4 + 2 + 1 - 1 + t(1)
\end{aligned}
$$

$$
\sum_{i=0}^{log_2 n} 2^i = 2^{log_2 n + 1} - 1 = ?
$$

$$
\begin{aligned}
t(n) &= n + t(\frac{n}{2}) \\
&= n + \frac{n}{2} + t(\frac{n}{4}) \\
&= n + \frac{n}{2} + \frac{n}{4} + t(\frac{n}{8}) \\
&= n + \frac{n}{2} + \frac{n}{4} + \ldots \frac{n}{2^{k-1}} + t(\frac{n}{2^k}) \\
&= n + \frac{n}{2} + \frac{n}{4} + \cdots + 4 + 2 + t(1), \quad \text{when } 2^k = n \\
&= \underbrace{n + \frac{n}{2} + \frac{n}{4} + \cdots + 4 + 2 + 1}_{\sum_{i=0}^{log_2 n} 2^i} - 1 + t(1)
\end{aligned}
$$

$$\sum_{i=0}^{log_2 n} 2^i = 2^{log_2 n + 1} - 1 = 2n - 1$$

We have solved the following recurrences:

$$t(n) = c + t(n-1)$$

$$t(n) = c\,n + t(n-1)$$

$$t(n) = c + 2\,t(n-1)$$

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

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

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

In COMP 251, you will prove a general result called the Master Theorem which covers all these cases and more!

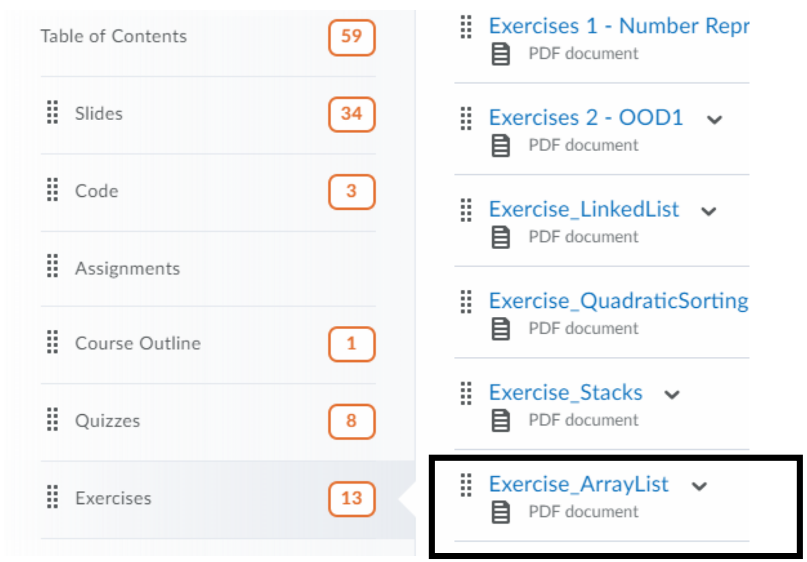
$$1 + 2 + 4 + \ \ldots + \ 2^{log_2 n} = \ \ 2n \ - 1$$

Look familiar? Where does this arise ?

$$1 + 2 + 4 + \ \ldots + \ 2^{log_2 n} = \ 2n \ - 1$$

Look familiar? Where does this arise ?

- Adding n elements to an empty arraylist and resizing $log_2 n$ times.



See Q6 for details.

$$1 + 2 + 4 + \ldots + 2^{log_2 n} = 2n - 1$$

Look familiar? Where does this arise ?

- Adding n elements to an empty arraylist.
- Putting n elements into an empty hash table. *(The rehashing takes the same amount of time as in the array list example. Why?)*

$$1 + 2 + 4 + \ \ldots + \ 2^{log_2 n} = \ 2n \ - 1$$

Look familiar? Where does this arise ?

- Adding n elements to an empty arraylist.
- Putting n elements into an empty hash table.

So, on average, each of these operations is O(1).

In COMP 251, this is called "amortized" analysis.

TODO

- lecture 35, 36, 37: Asymptotic Complexity
- Quiz 5 on Friday Nov 30
- Room change on Friday Nov 30 for Sec. 001. (10:35-11:25) ENGTR 0100