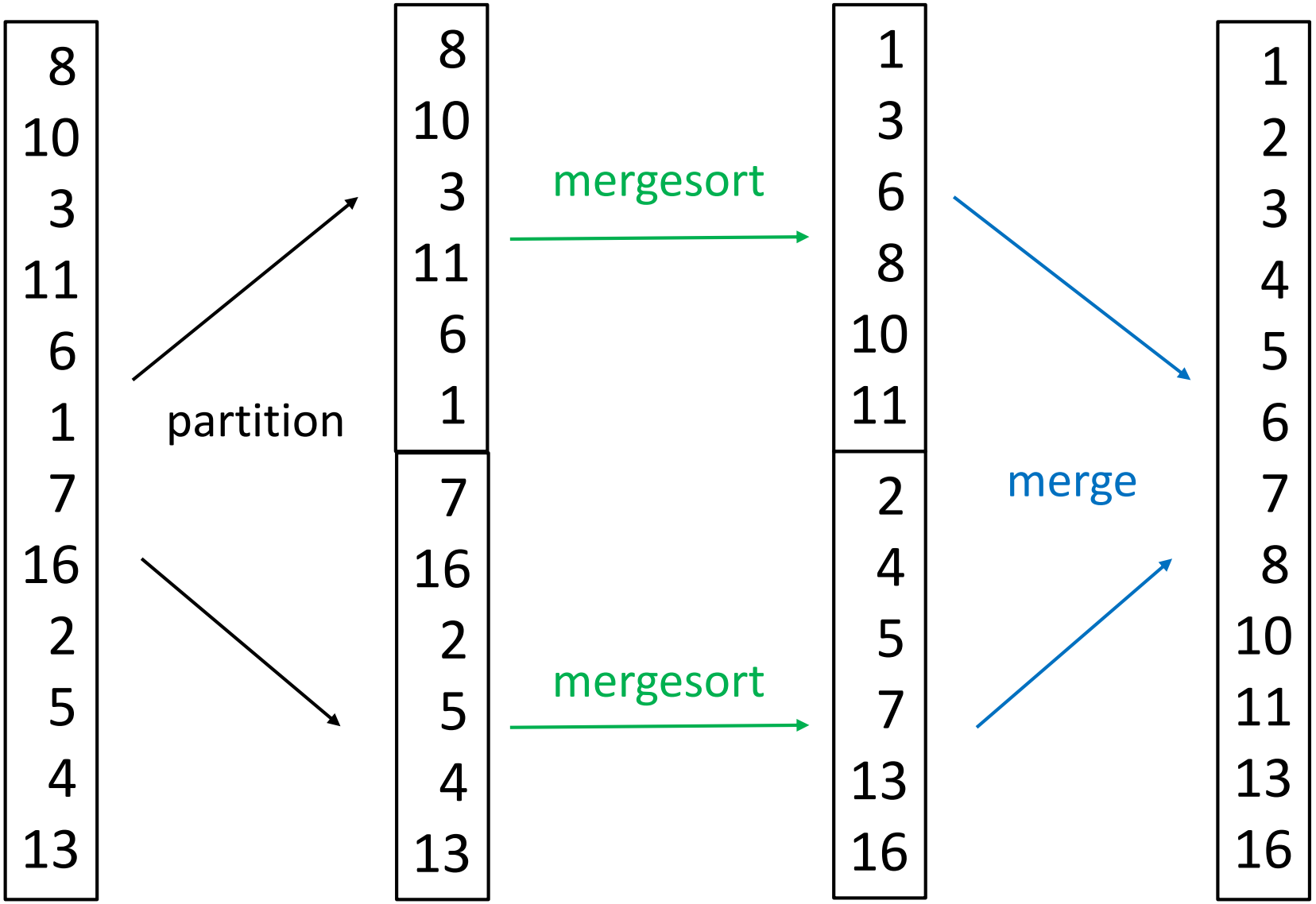
COMP 250

Lecture 34

recurrences 2:
mergesort & quicksort

April 1, 2022

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 )
    }
 }
```

In case you want to see code, ...

```
public static void mergeSort(int[] a) {

    int n = a.length;

    if (n < 2) {
        return;
    }
    int mid = n / 2;
    int[] left = new int[mid];
    int[] right = new int[n - mid];

    for (int i = 0; i < mid; i++) {
        left[i] = a[i];
    }
    for (int i = mid; i < n; i++) {
        right[i - mid] = a[i];
    }
    mergeSort(left);
    mergeSort(right);

    merge(a, left, right);
}
```

```
public static void merge(
    int[] a, int[] left, int[] right) {

    int n_left = left.length;
    int n_right = right.length;
    int i = 0, j = 0, k = 0;

    while (i < n_left && j < n_right) {
       if (left[i] <= right[j]) {
           a[k++] = left[i++];
       }
       else {
          a[k++] = right[j++];
       }
    }
    while (i < n_left) {
        a[k++] = left[i++];
    }
    while (j < n_right) {
       a[k++] = right[j++];
    }
}
```

Note: mid is defined slightly differently here from previous slide, namely the ranges here are [0, mid-1] and [mid, n]. This is necessary for correct array indexing.

The list has n elements.

```
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.

merge takes time proportional to n.

The list has n elements.

```
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 $n = 1$

We ignore the constant term for simplicity.

getElements and **merge** both take time proportional to n.

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

What if n is not even ?

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

In general, one could write this 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

But coming up with a closed form solution would be impossible.
So we assume $n = 2^k$ to solve the recurrence.
The more general recurrence has roughly the same solution.

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

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

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

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

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

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

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

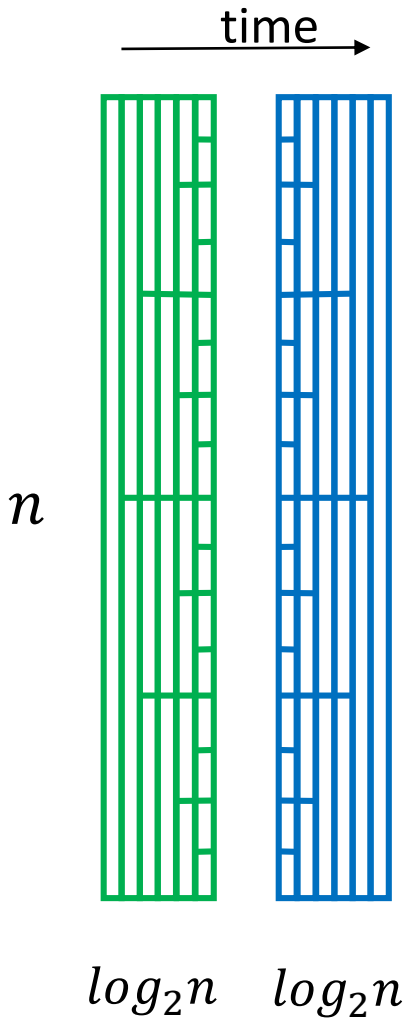
Base case

Dominant term, so $O(n \log n)$

Q: *How many calls* are made to mergesort? (including the original one)

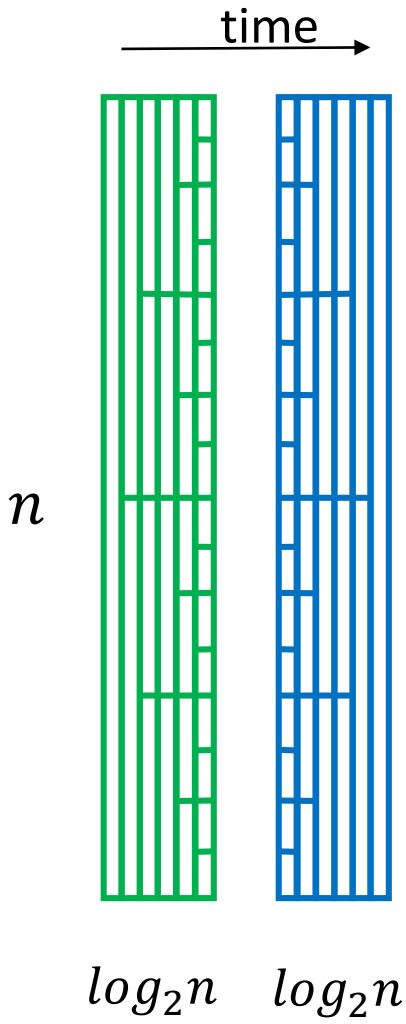
Hint: each vertical green rectangle is one call.

```
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 )
    }
}
```



A: $1 + 2 + 4 + 8 + 16 = 32 - 1 = 31$

Q: *How many calls* are made to mergesort? (including the original one)



Hint: each vertical green rectangle is one call.

```
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 )
   }
}
```

The recurrence is:

$$f(n) = 1 + 2\,f\left(\frac{n}{2}\right), \qquad f(1) = 1.$$

$$f(n) \quad = \quad 1 + 2\ f(\frac{n}{2})$$

$$
\begin{aligned}
f(n) &= 1 + 2\ f(\frac{n}{2}) \\
&= 1 + 2\ (1 + \ 2f(\frac{n}{4}))
\end{aligned}
$$

$$
\begin{aligned}
f(n) &= 1 + 2\ f(\frac{n}{2}) \\
&= 1 + 2\ (1 + \ 2f(\frac{n}{4})) \\
&= 1 + 2 + 4f(\frac{n}{4})
\end{aligned}
$$

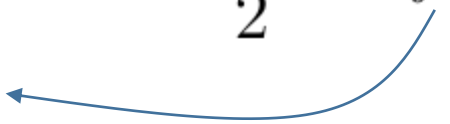
$$
\begin{aligned}
f(n) &= 1 + 2\ f(\frac{n}{2}) \\
&= 1 + 2\ (1 + \ 2f(\frac{n}{4})) \\
&= 1 + 2 + 4f(\frac{n}{4}) \\
&= 1 + 2 + 4(1 + 2f(\frac{n}{8}))
\end{aligned}
$$

$$
\begin{aligned}
f(n) &= 1 + 2\ f(\frac{n}{2}) \\
&= 1 + 2\ (1 + \ 2f(\frac{n}{4})) \\
&= 1 + 2 + 4f(\frac{n}{4}) \\
&= 1 + 2 + 4(1 + 2f(\frac{n}{8})) \\
&= 1 + 2 + 4 + 8f(\frac{n}{8})
\end{aligned}
$$

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

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

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



Q: *How many calls* are made to merge?

Notes: Each call to mergesort makes one call to merge *except in the base case*.
Although merge itself is not recursive, we can still define a recurrence here.

```
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 )
   }
 }
```

A: The recurrence for number of calls to merge is

$$f(n) = 1 + 2f\left(\frac{n}{2}\right), \qquad f(1) = 0.$$

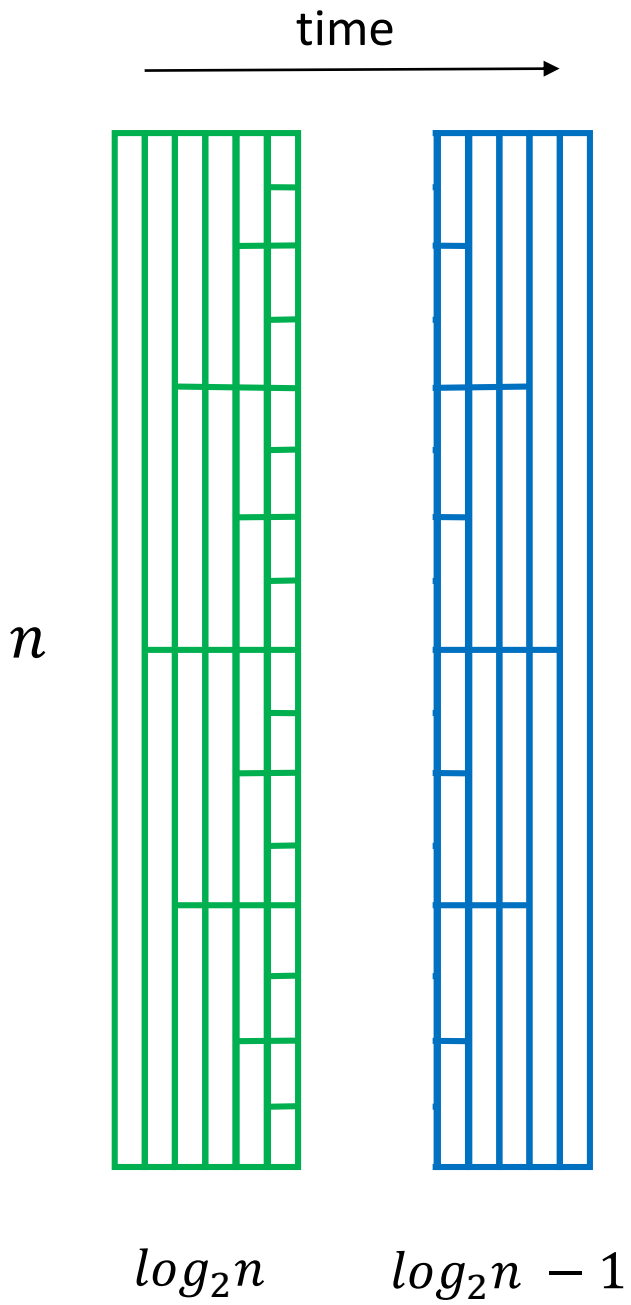
$$
\begin{aligned}
f(n) &= 1 + 2\ f(\frac{n}{2}) \\
&= 1 + 2\ (1 + \ 2f(\frac{n}{4})) \\
&= 1 + 2 + 4f(\frac{n}{4}) \\
&= 1 + 2 + 4(1 + 2f(\frac{n}{8})) \\
&= 1 + 2 + 4 + 8f(\frac{n}{8}) \\
&= 1 + 2 + 4 + \cdots + 2^{k-1} + 2^k f(\frac{n}{2^k}) \\
&= 1 + 2 + 4 + \cdots + \frac{n}{2} + nf(1) \quad \text{when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

Same as previous derivation.

$f(1) = 0$.

```
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 )
    }
}
```

Notice we do not have the n merge calls when n = 1.



Let's return to the mergesort recurrence from slide 6.

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

What if we change the base case and stop the recursion at a larger list size (some fixed $n_0 > 1$) and we do say bubblesort for the small lists ?

```
mergesort(list){
   if  list.length <=  4                    //   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 )
   }
 }
```

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

```
mergesort(list){
   if  list.length < = 4
        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 )
   }
 }
```

How does this affect the back substitution and solution?

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

```
mergesort(list){
   if  list.length <= 4
        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 )
   }
 }
```

How does this affect the back substitution and solution?

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

k times

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

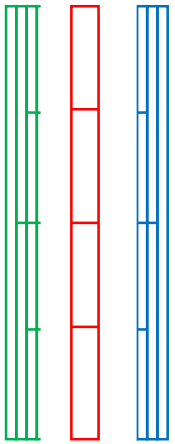
```
mergesort(list){
   if  list.length <= 4
        return bubblesort(list)
   else{
      mid = (list.size - 1) / 2
      list1 =  list.getElements(0,mid)
      list2 =  list.getElements(mid+1, list.size-1)
      list1  =  mergesort(list1)
      list2  =  mergesort (list2)
      return   merge( list1, list2 )
   }
 }
```

How does this affect the back substitution and solution?

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

```
mergesort(list){
   if  list.length <= 4
        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 )
   }
 }
```

How does this affect the back substitution and solution?

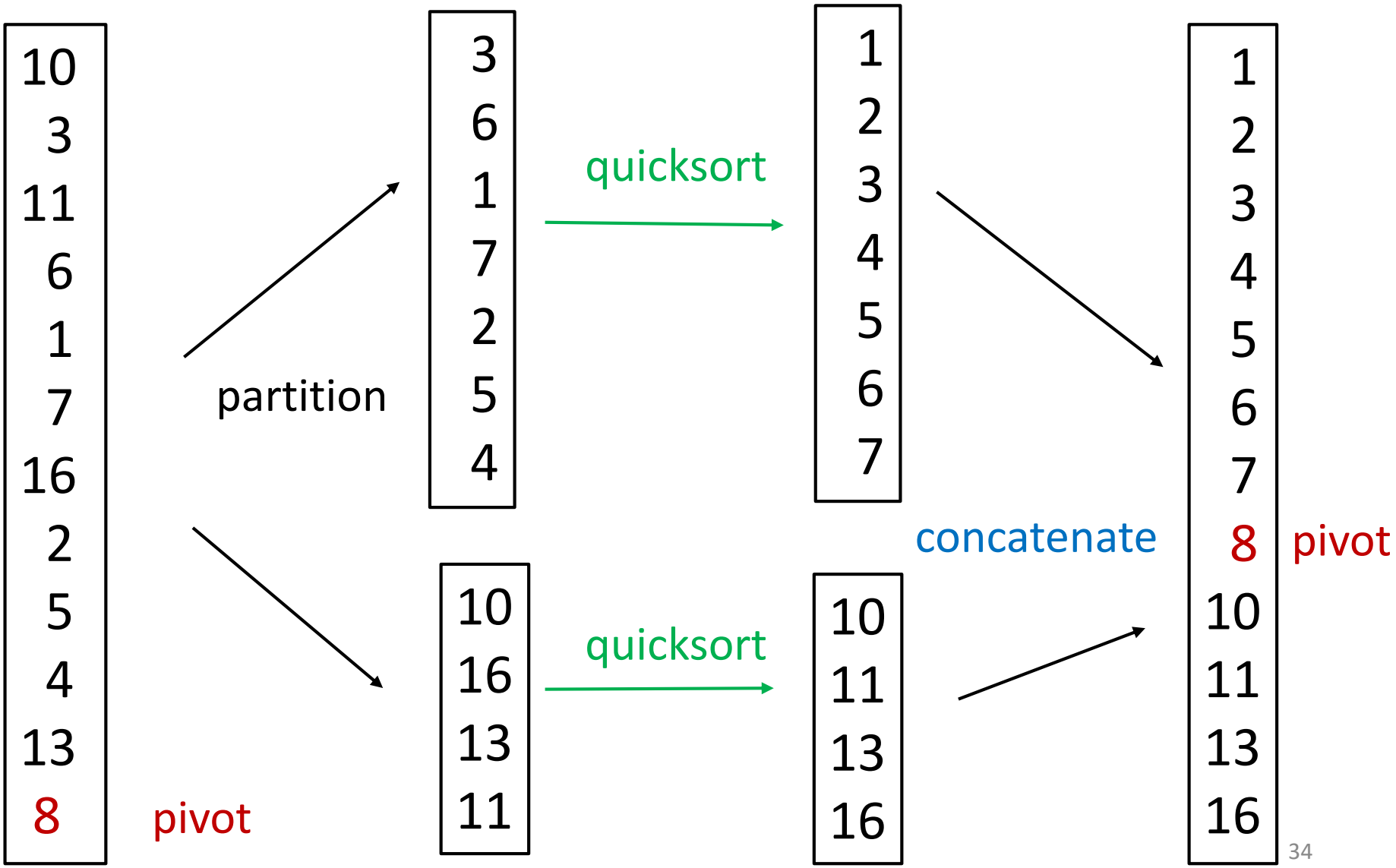


$$
\begin{aligned}
t(n) &= c\,n + 2\,t(\frac{n}{2}) \\
&= \ldots \\
&= c\,n\,k \;+\; 2^k\,t(\frac{n}{2^k}), \quad \text{and letting } 2^k = \frac{n}{4} \text{ gives...} \\
&= c\,n(\log_2 n - 2) + \frac{n}{4}\,t(4)
\end{aligned}
$$

$t_{bubble}(4)$

The second term grows only with n since $t_{bubble}(4)$ is a constant.
For large n, the first term dominates → $n\,log_2 n$.

Example 6: Quicksort



Quicksort Best Case

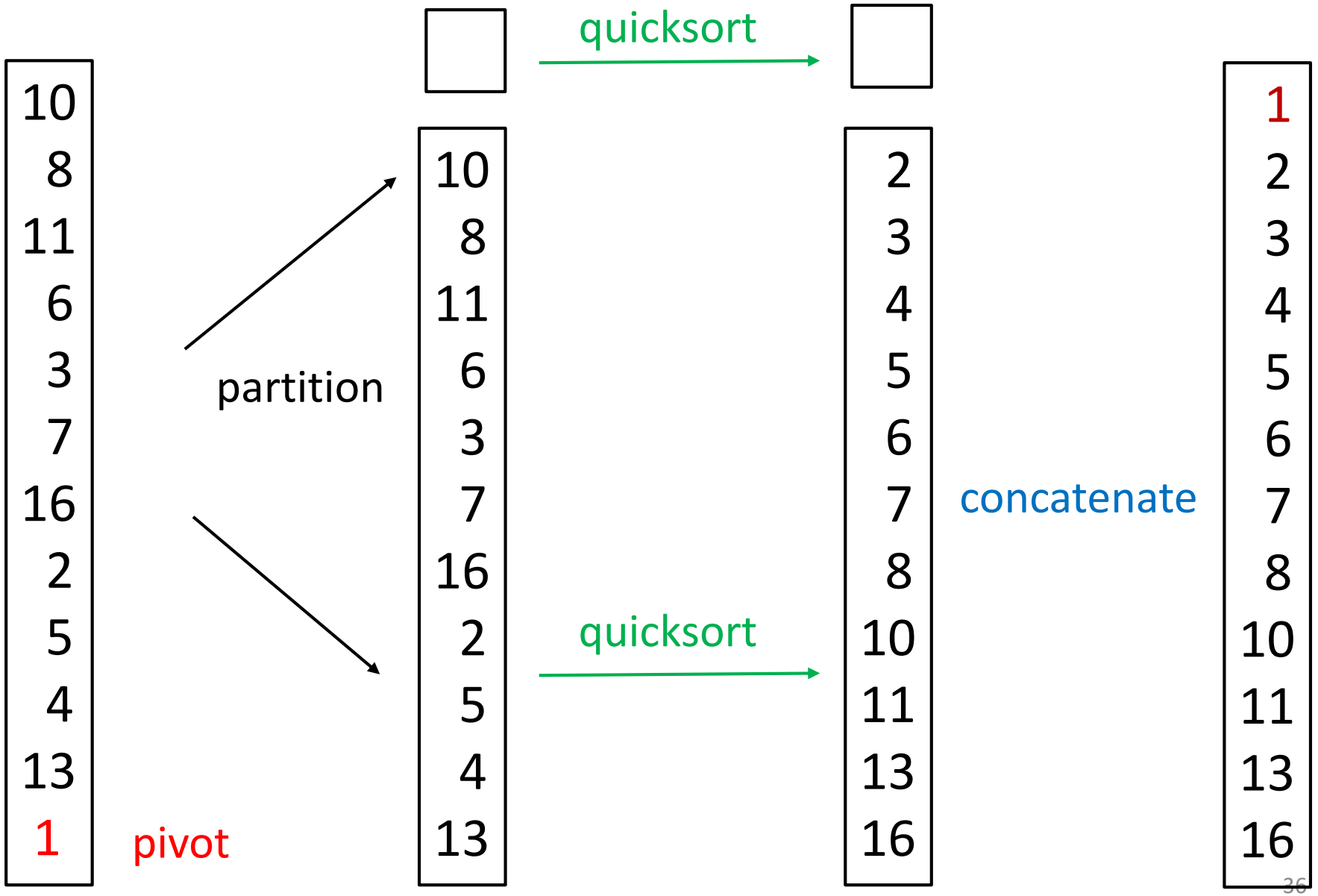
$$t(n) = c\,n + 2\,t\left(\frac{n}{2}\right)$$

Partition requires scanning the list, based on the pivot

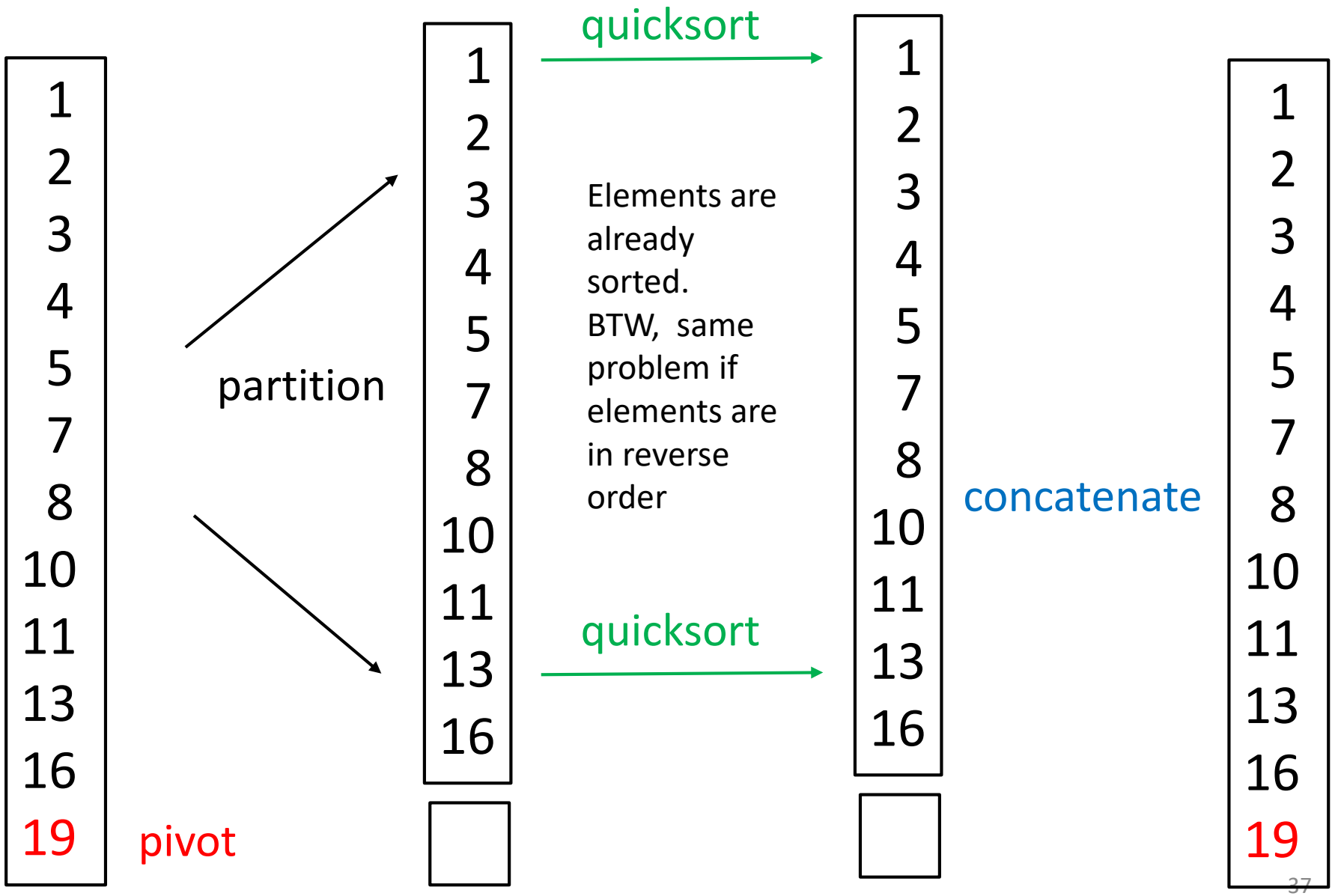
Best case: the two sublists have ~ the same size for each call

This is the same recurrence as mergesort → same solution.

Quicksort ***worst*** case (for one call) ?



Quicksort ***worst*** case (for all calls) ?



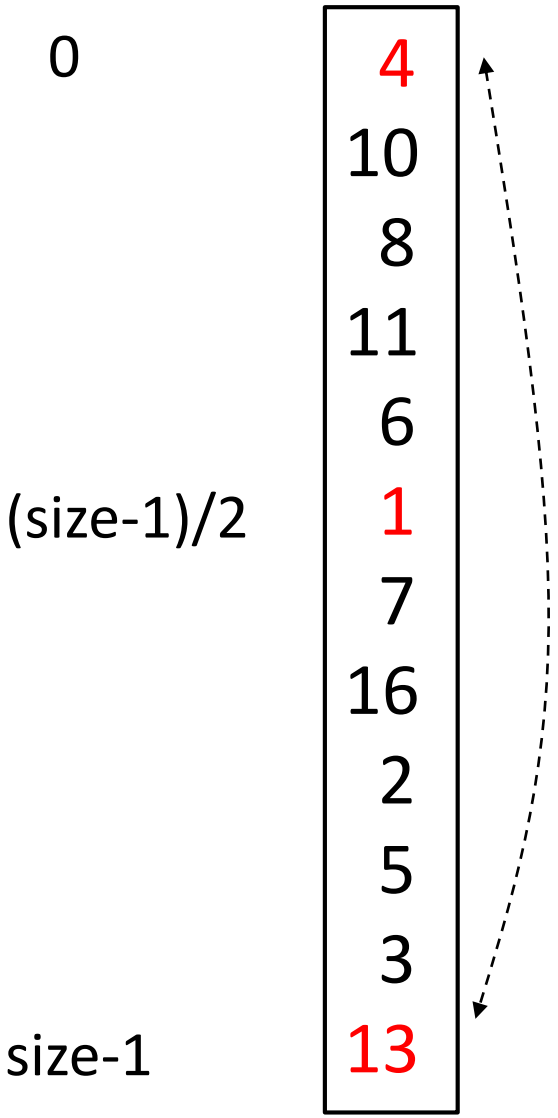
Quicksort worst case

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

Same recurrences as last lecture…. (sorting a list by successively removing minimum element)

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

Recall lecture 28 slide 48 : How to reduce the chance of an unbalanced partition at each step of Quicksort?

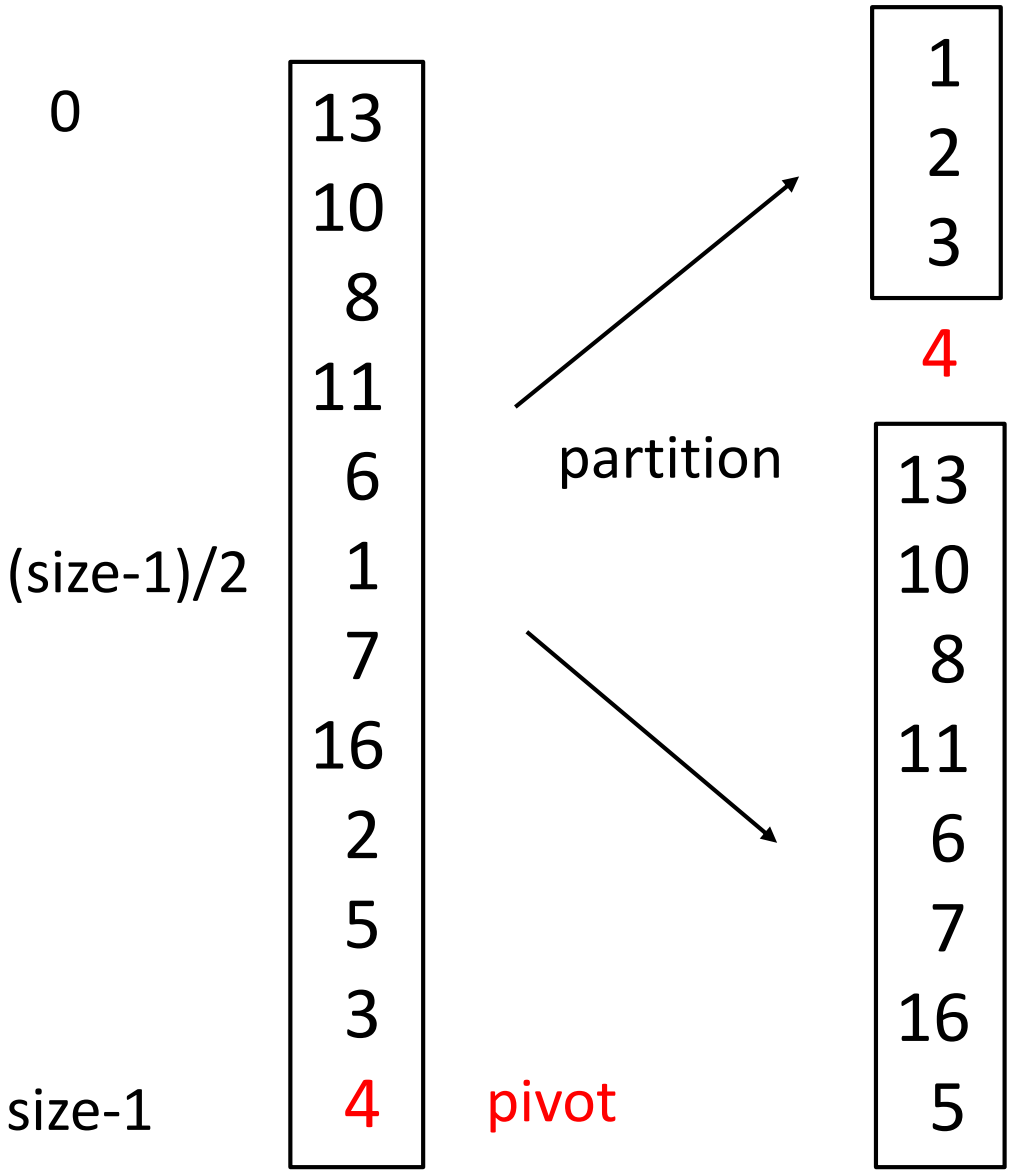


"Median of three" technique:

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

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

Swap the median with the element at the back so that the median is in the pivot position for quicksort.



Median of 3 works well in practice, especially if the list is already nearly sorted.

You may cover it in COMP 251.

We have solved the following recurrences:

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

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

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

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

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

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

Coming up…

Lectures

Mon, Wed, Fri : April 4, 6, 8

big O, big Omega, big Theta,

best and worst cases

Assessments

Quiz 5 is in Mon. April 4

Practice quizzes are available for Quiz 5 and for lectures after that (33-37)

Assignment 4 due Wed. April 6.