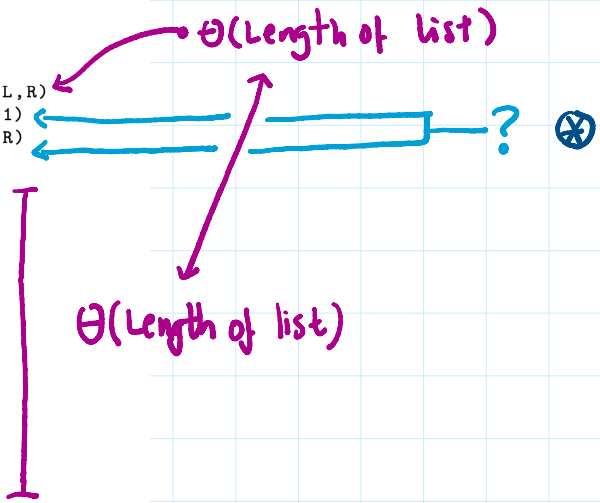
CMSC351-2023-11-13

Friday, October 13, 2023 8:11 AM

QUICKSORT - TIME COMPLEXITY, ETC!

```
\\ PRE: A is a list of length n.
\\ Note that A is considered global here.
function quicksort(A,L,R)
    if L < R then
         resultingpivotindex = partition(A,L,R)
         quicksort(A,L,resultingpivotindex-1)
         quicksort(A,resultingpivotindex+1,R)
    end
end
function partition(A,L,R)
    \\ To use a different pivotvalue
    \\ swap it with A[R] here.
    pivotvalue = A[R]
    t = L
    for i = L to R-1
        if A[i] <= pivotvalue
            A[t] <-> A[i]
            t = t + 1
        end
    end
    A[t] <-> A[R]
    return t
end
quicksort(A,0,n-1)
\\ POST: A is sorted.
```



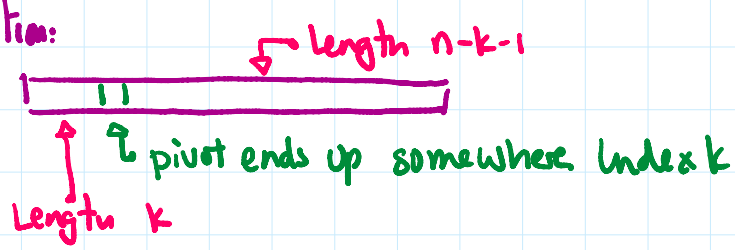
① looking at ⊛: we have two sublists we call partition on.

if k = resulting pivot index then:

first call to a list of length k

second call to a list of length n-k-1

Pic: After partition:



So for a rec. reln:

$$T(n) = T(k) + T(n-k-1) + \Theta(n)$$ where k = resulting pivot index.

WORST-CASE: $k=0$ or $k=n-1$: when pivot always ends up at an end.

WLOG if $k=0$ then we get:

$$T(n) = \underbrace{T(0)}_{0 \text{ or Const.}} + T(n-1) + \Theta(n)$$

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

If we solve this (fill in $\Theta(n)$ and dig down)

we get $T(n) = \Theta(n^2)$ ← wc!

we get $T(n) = \Theta(n^2)$ ← WC!

BEST-CASE: resulting pivot index $k = \frac{n}{2}$ or $\lfloor \frac{n}{2} \rfloor$ or $\lceil \frac{n}{2} \rceil$.
Then $T(n) = 2T(\frac{n}{2}) + \Theta(n)$
if we solve this (Master Thm)
we get $T(n) = \Theta(n \lg n)$ ← BC

AVG-CASE: note that speed of partition is $\Theta(n)$ no matter what the order of the elements is.
The only thing that impacts speed is the resulting pivot index.
What if we picked the pivot value randomly (uniformly) each time.
We get (LOL):

$$T(n) = \frac{1}{n} \sum_{k=0}^{n-1} [T(k) + T(n-k-1) + \Theta(n)]$$ ← Expected Value Calculation!

Soln to this (not obvious, constructive induction) is $T(n) = \Theta(n \lg n)$ ← AC

note: this is how most implementations of Quick Sort work.

② In-Place? Yes!
③ Aux Space? $\Theta(1)$
④ Stable? No! Why not?!

LIMITATIONS OF COMPARISON-BASED SORTING ALGORITHMS!

① Intro: All of our sorting algs. use comparisons: $<, >, =$, etc.
If we create another (any) sorting alg using comparisons, how fast could it be?
Claim: in the worst-case, no better than $n \lg n$.

comparisons, how fast could it be?

Claim: in the worst-case, no better than $n \lg n$.

Rigorously: WC: $T(n) = \Omega(n \lg n)$.

② Decisions + Decision Trees

Q: To sort a list w/ n elts using comparisons, how many comparisons do we need?

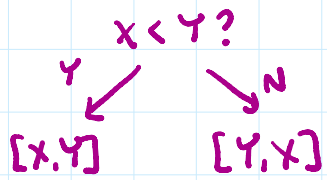
A: Let's look at two examples:

(A) List of length 2: we need 1 comparison.

(B) List of length 3: To sort any list w/ $n=3$ elts we will need at most 3 comparisons.

Before we go further, reflect on (A). Here is a decision tree:

Start w/ [X,Y] we wish to sort.



Now let's look at Bubble Sort!

```
for i = 0 to n-1
    for j = 0 to n-i-2
        if A[j] < A[j+1]
            nothing
        else
            swap A[j] and A[j+1]
        end
    end
end
```

How does this work on a list of length 3?

(1) Compares A[0] < A[1]. if false, swap

(2) " A[1] < A[2]. if false, swap

(3) " A[0] < A[1]. if false, swap.

Consider a list [X,Y,Z]. There are 6 possible ways this could be sorted.

Okay. Sps X<Y<Z, start w/ [X,Y,Z]

is X<Y? YES! no swap! now A=[X,Y,Z]

is Y<Z? YES! A=[X,Y,Z]

is X<Y? YES BUT NO REAL DECISION A=[X,Y,Z]

Sps X<Z<Y, start w/ [X,Y,Z]

is X<Y? YES A=[X,Y,Z]

is Y<Z? NO, SWAP A=[X,Z,Y]

is X<Z? YES A=[X,Z,Y]

Sps Z<X<Y, start w/ [X,Y,Z]

Sps $Z<X<Y$. Start wl $[X,Y,Z]$

is $X<Y$?	YES	$A=[X,Y,Z]$
is $Y<Z$?	no. swap	$A=[X,Z,Y]$
is $X<Z$?	no swap	$A=[Z,X,Y]$

Sps $Y<X<Z$. Start wl $[X,Y,Z]$

is $X<Y$?	no. swap	$A=[Y,X,Z]$
is $X<Z$?	YES	$A=[Y,X,Z]$
is $Y<X$?	yes, automatic	$A=[Y,X,Z]$

Sps $Y<Z<X$. Start wl $[X,Y,Z]$

is $X<Y$?	no. swap	$A=[Y,X,Z]$
is $X<Z$?	no. swap	$A=[Y,Z,X]$
is $Y<Z$?	yes	$A=[Y,Z,X]$

Sps $Z<Y<X$. Start wl $[X,Y,Z]$

is $X<Y$?	No. Swap	$A=[Y,X,Z]$
is $X<Z$?	No. Swap	$A=[Y,Z,X]$
is $Y<Z$?	NO. Swap	$A=[Z,Y,X]$