lecture 18

(deterministic) quicksort and selection

- review of quicksort
- selection problem
- median of median of 5 method

Resources: Roughgarden : Algorithms I units V to VIII
https://class.coursera.org/algo-004/lecture/38

```
mergesort (list) {
   if (list.size == 1)
      return list
   else {
      partition list into two approximately
        equal size lists l1, l2
      l1 <- mergesort (l1)
      l2 <- mergesort (l2)

      return merge (l1, l2)
   }
}
```

Recall from COMP 250

```
quicksort (list) {
  if (list.size() <= 1)
        return list
  else { p <- list.removePivot()
        l1 <- list of elements LessThan(p)
        l2 <- list of elements NotLessThan (p)
        l1 <- quicksort (l1)
        l2 <- quicksort (l2)
        return concatenate (l1, p, l2)
     }
}
```

Mergesort does it main work (merge) after the recursive calls.

Quicksort does its main work (partition) before the recursive calls.

In both cases, this work is $O(n)$.

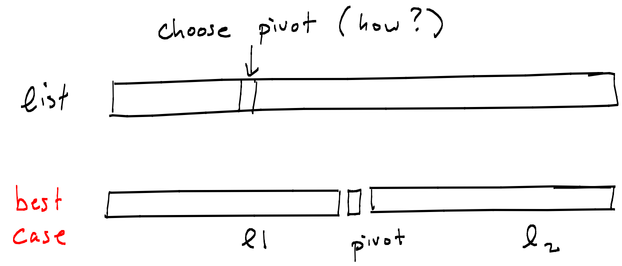
Mergesort takes $O(n \log n)$ time. The typical implementation requires $O(n)$ extra space, however, which slows down the algorithm.

(Those who've done ESCE 221 or COMP 273 know about memory hierarchies and will understand why using extra space slows down algorithms)

Quicksort takes $O(n^2)$ time in the worst case of a poorly chosen pivot, and $O(n \log n)$ in the best case that the pivot partitions the set into two sets of size $\approx \frac{n}{2}$.

Quicksort can be done "in place" which makes it faster than mergesort in practice.

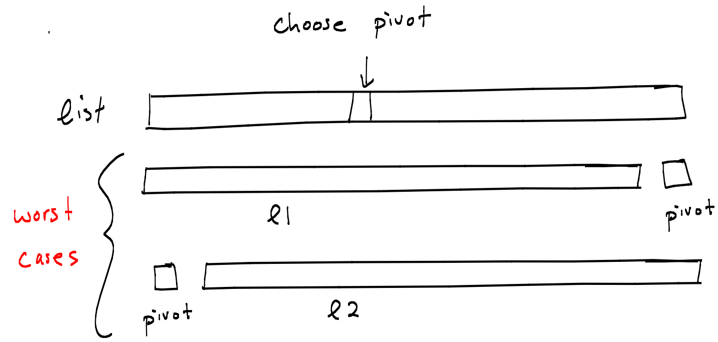
Quicksort



best case splits problem exactly in half.

$$t(n) = 2\,t\left(\tfrac{n}{2}\right) + cn$$

so $t(n)$ is $O(n \log n)$



worst case : larger subproblem has size $n-1$.

$$t(n) = t(n-1) + t(0) + n$$
$$= 1 + 2 + 3 + \dots \; n-1 + n + n\,t(0)$$

which is $O(n^2)$

How to choose the pivot in quicksort?

The median of a set of numbers is the element such that half the elements are less than and half are greater than that element.

The median would be the best pivot for quicksort.

In practice it is usually "good enough" to consider the first, middle, and last elements in the list and use the median of these 3 as the pivot. You can compute this median in $O(1)$.

It only rarely produces a poor partition.

It gives the median if the whole list is already sorted.

One way to select the median is to sort the list and then choose the middle element. But this is silly because we want to choose the median in order to speed up quicksort!

Does there exist an algorithm that selects the median in $O(n)$?

Yes, as we'll now see.

Selection problem (more general)

Given a set of n comparable elements, (an ordering exists, but it is not given) find the ith element in this ordering.

In particular, to select the median, we use:

$$i = \begin{cases} \frac{n}{2}, & \text{if } n \text{ is even} \\ \frac{n-1}{2}, & \text{if } n \text{ is odd} \end{cases}$$

Example $(n = 11)$

a[ ]

0	1	2	3	4	5	6	7	8	9	10
9	3	16	5	2	6	81	24	1	19	16

select (a, 0) = 1

select (a, 3) = 5

select (a, 5) = 9

select (a, 10) = 81

Obvious solution (but too slow):

- sort the elements a[ ]
- do an array lookup, a[i]

However, this takes $O(n \log n)$.

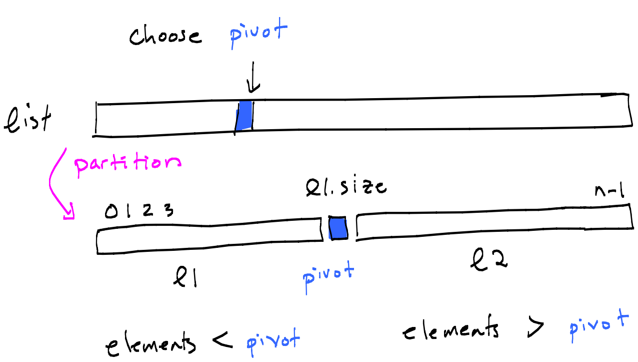
We want an $O(n)$ algorithm.

Selection problem (today's approach)

1) Consider a divide and conquer algorithm for select that has the same flavour as quicksort, and the same worst case behavior i.e. $O(n^2)$.

2) Improve the above algorithm by using the median-of-median-of-5 method ⟹ an $O(n)$ select algorithm.

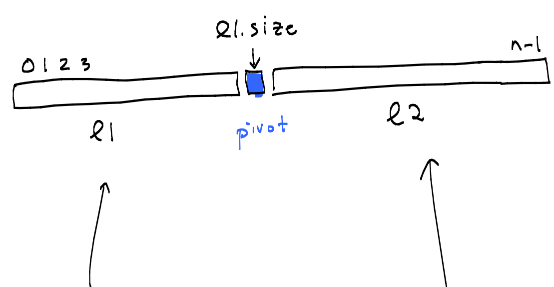
It was pointed out to me after the lecture that there is a problem with allowing elements to be equal to the pivot. See exercises. So I have changed the slides to avoid this problem.



i == l1.size ⟹ we have found it

" < " ⟹ element i is in l1

" > " ⟹ " " l2

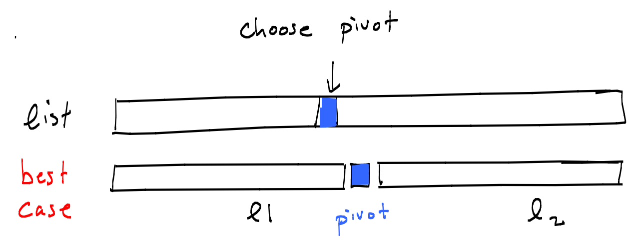


if i < l1.size
then select (l1, i)

if i > l1.size
then select(l2, i − (l1.size + 1))

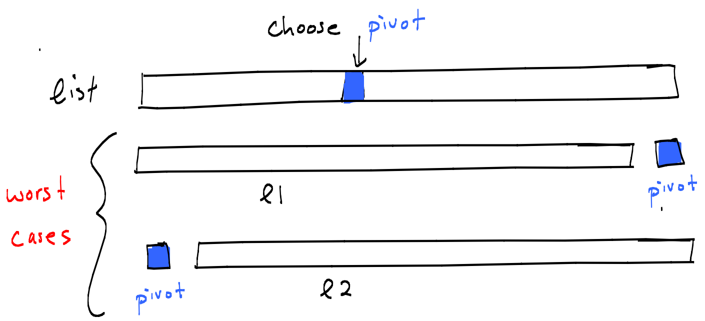
```
select ( list, i ) {       // 0 ≤ i < list.size
   partition the list around pivot
   l1 is the list of elements less than pivot
   l2 is   "    "    "    "   greater than pivot

   if ( i == l1.size )
       return pivot
   else if ( i < l1.size)
       return select ( l1, i )
   else
       return select ( l2, i - (l1.size + 1))
}
```



best case : split problem exactly in half.

$$t(n) = t\left(\frac{n}{2}\right) + cn$$
$$= c\left(1 \cdots + \frac{n}{8} + \frac{n}{4} + \frac{n}{2} + n\right)$$
$$= 2cn$$



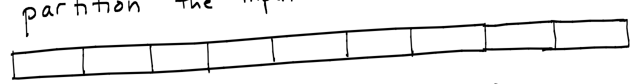
worst case : larger subproblem has size $n-1$ and recursive call on that larger list.

$$t(n) = t(n-1) + n$$
$$= n + (n-1) + (n-2) + \ldots + 2 + 1$$
$$= n(n+1)/2 \implies O(n^2)$$

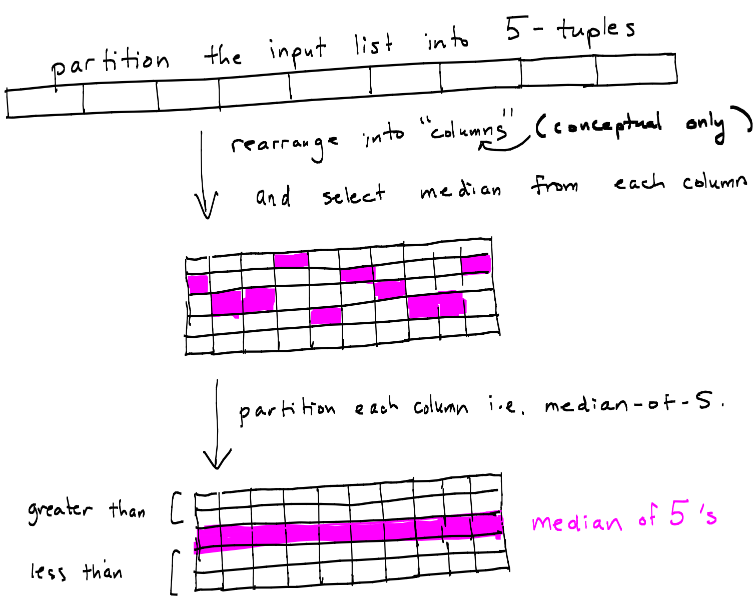
Q: How to choose pivot so that you are guaranteed to stay away from worst case ?

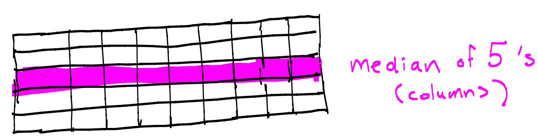
A: the "median of median of 5's" method.

- partition the input list into 5-tuples



- use a $O(1)$ algorithm for finding the median within each 5-tuple and make a list of these median-of-5's
- select the median of this list of median-of-5's (recursive) as the pivot

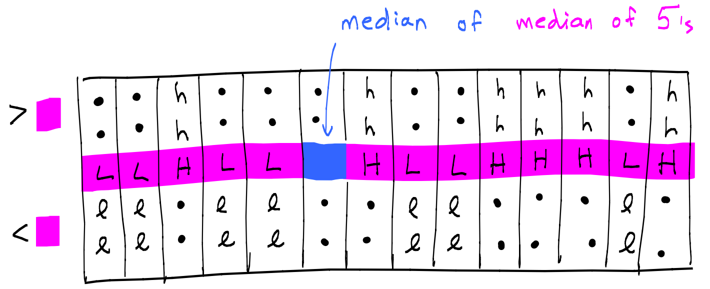




Next, ...

Select the median of the median of 5's.

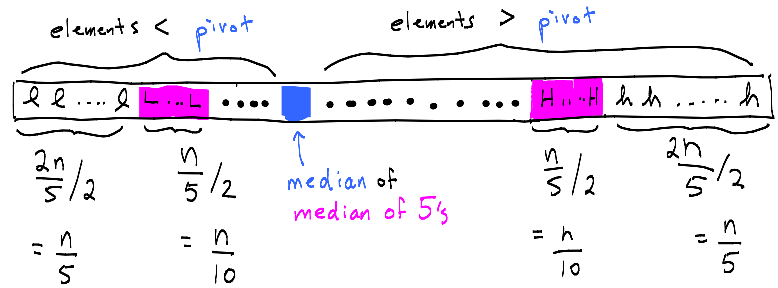
We will use it as the pivot to select the ith element from the list of n.



$\ell < L <$ median of median of 5's $< H < h$

How many of each are there?

Reorder the original list (n elements) and group the elements as follows.



Thus, the median of median of 5's is greater than $\frac{3}{10}n$ of the elements and less than $\frac{3}{10}$ of the elements.

If select(n, i) uses the median of median of 5's as the pivot, then the next recursive call to select() will be on a list of size <u>at most</u> $\frac{7}{10}n$.

Thus, there will be at most $\log_{\frac{10}{7}} n$ recursive calls to select.

[ASIDE: recall master method ideas, $b = \frac{10}{7}$]

```
select (list, i) {   // 0 ≤ i < list.size
   if list.size ≤ 5, find ith element by brute force
      and return it    // base case.
   else {
     partition list into 5-tuples
     listMedianOf5s ← find medians of 5-tuples
     pivot ← select( listMedianOf5s, listMedianOf5s.size/2)

     partition the list around pivot          }
     if (i == l1.size)                        }
        return pivot                          }  Same
     else if (i < l1.size)                    }  as
        return select(l1, i)                  }  before
     else                                     }
        return select(l2, i-(l1.size + 1))    }
}  }
```

select makes <u>two</u> recursive calls

1) select the median of median of 5's and use it as the pivot
   - this avoids the worst case of a poorly chosen pivot

2) select the ith element in the list (by looking in l1 or l2)

```
t(n)        select (list, i) {   // 0 ≤ i < list.size
O(1)           if list.size ≤ 5, find ith element by brute force
                  and return it    // base case.
               else {
O(n)             partition list into 5-tuples
O(n)             listMedians5 ← find medians of 5-tuples
t(n/5)           pivot ← select( listMedianof5, listMedianof5.size / 2)
O(n)             partition the list around pivot
O(1)             if ( i == l1.size )
                    return pivot
< t(7/10 n)      else if ( i < l1.size)
                    return select( l1, i )
< t(7/10 n)      else
                    return select( l2, i-(l1.size + 1))
            }  }
```

$$\Rightarrow t(n) < t\left(\frac{n}{5}\right) + t\left(\frac{7}{10}n\right) + cn$$

$$t(n) < t\left(\frac{n}{5}\right) + t\left(\frac{7}{10}n\right) + cn$$

We <u>cannot</u> apply Master Theorem.

However, note $\frac{n}{5} + \frac{7}{10}n < n$

This is all we will need to show that $t(n)$ is $O(n)$.

<u>Claim:</u>
if $t(n) < t\left(\frac{n}{5}\right) + t\left(\frac{7}{10}n\right) + cn$ for all n,
then $t(n) \le \beta n$ for some $\beta > 0$.

<u>Proof</u>: by induction.

<u>base case (easy):</u> $t(1) \le \beta$ for some β.
(next slide, we'll take $\beta = 10c$)

<u>induction hypothesis:</u>
Suppose $t(k) \le \beta k$ for all $k < n$

<u>induction step</u>: show $t(n) \le \beta n$

$$
\begin{aligned}
t(n) &< t\left(\frac{n}{5}\right) + t\left(\frac{7}{10}n\right) + cn \\
&\le \beta \frac{n}{5} + \beta \cdot \frac{7}{10} n + cn \\
&\le \beta\left(\frac{2n}{10} + \frac{7n}{10}\right) + cn \\
&\le \beta\left(\frac{9}{10}n\right) + cn \\
&= 10c\left(\frac{9}{10}n\right) + cn, \text{ where } \beta = 10 \cdot c \quad \text{(because it works)} \\
&= 10\,c\,n \\
&= \beta n
\end{aligned}
$$

Thus, $t(n)$ is $O(n)$.

<u>Returning to Quicksort</u>

We can select the median in $O(n)$.
Thus, quicksort <u>can</u> solved using
$$t(n) = 2t\left(\frac{n}{2}\right) + cn$$
i.e. $O(n \log n)$ in the worst case.

Most implementations of quicksort do <u>not</u> use median of median of 5's, however. Why not?

- You can select a median in $O(n)$ but the constant β will be large. In practice, it is not worth it. i.e. it doesn't run faster.

- If you choose the pivot "<u>randomly</u>", then the chances of getting bad pivots over and over is small.
  We will see this next week!