Recall from COMP 270

```plaintext
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) }
```

```plaintext
mergesort (list) if (list.size == 0) return list else { partition list into two approximately equal size lists l1, l2 l1 = mergesort (l1) l2 = mergesort (l2) return merge (l1, l2) }
```

Mergesort does most of its 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 22 or COMP 273 know about memory hierarchies and will understand why using extra space slows down algorithms)

Quicksort takes $O(n \log n)$ 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.

Resources: Roughgarden: Algorithm I units VII to VIII

https://class.coursera.org/algo-004/lecture/38
**Quicksort**

- **Choose pivot**
  - Choose pivot
  - List

- **Best case**
  - List
  - Pivot
  - \( L_1 \)
  - \( L_2 \)

  Best case splits problem exactly in half.
  
  \[ t(n) = 2t\left(\frac{n}{2}\right) + cn \]

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

- **Worst case**
  - List
  - Pivot
  - \( L_1 \)
  - \( L_2 \)

  Worst case: larger subproblem has size \( n - 1 \).
  
  \[ t(n) = t(n - 1) + t(0) + n \]

  \[ = 1 + 2 + 3 + \ldots + n - 1 + n + n t(0) \]

  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.

---

**Selection problem (more general)**

Given a set of \( n \) comparable elements, (an ordering exists, but it is not given) find the \( i \)th 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 \( n = 11 \)

\[
\begin{array}{cccccccccccc}
\text{a[0] } & 9 & 3 & 1 & 6 & 5 & 2 & 6 & 8 & 1 & 9 & 16 \\
\text{a[1] } & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{array}
\]

\[
\begin{align*}
\text{select } (a, 0) &= 1 \\
\text{select } (a, 3) &= 5 \\
\text{select } (a, 5) &= 9 \\
\text{select } (a, 10) &= 81
\end{align*}
\]

Obvious solution (but too slow):
- sort the elements \( a[i] \)
- 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 \texttt{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-five method \(\Rightarrow\) an \( O(n) \) \texttt{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. \textbf{See exercises}.

So I have changed the slides to avoid this problem.

\[
\begin{align*}
\text{list} \\
\text{partition} \\
0 & 1 & 2 & 3 & \text{pivot} & \text{R1.size} & n-1 \\
\text{L1} & \text{pivot} & \text{L2} \\
\text{elements } < \text{pivot} & \text{elements } > \text{pivot} \\
\end{align*}
\]

\[
\begin{align*}
\text{if } i < \text{R1.size} & \text{ then } \text{select} (\text{R1}, i) \\
\text{if } i > \text{R1.size} & \text{ then } \text{select} (\text{R2}, i-(\text{R1.size}+1))
\end{align*}
\]
select (list, i) \{ 
  if (i == pivot) return pivot 
  else if (i < list.size) return select(l1, i) 
  else return select(l2, i-(l1.size+1)) 
\}

best case: split problem exactly in half. 

\[ t(n) = t(\frac{n}{2}) + cn \]

\[ = c \left( \frac{n^2}{8} + \frac{n}{4} + \frac{n}{2} + n \right) \]

\[ = 2cn \]

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 i'th element from the list of n.
Reorder the original list (n elements) and group the elements as follows.

\[ \text{elements} < \text{pivot} \quad \quad \text{elements} > \text{pivot} \]

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

select \( (\text{list}, i) \) \( \{ \}
\) \( \text{0} \leq i < \text{list.size} \)
\( \text{if list.size} \leq 5 \), find i th element by brute force and return it \( \text{// base case} \).
\( \text{else} \{ \)
\( \text{\quad partition list into 5-tuples} \)
\( \text{\quad listMediansOf5s} \leftarrow \text{find medians of 5-tuples} \)
\( \text{\quad pivot} \leftarrow \text{select} \{ \text{listMediansOf5s, listMediansOf5s.size}/2 \} \)
\( \text{\quad partition the list around pivot} \)
\( \text{\quad if (i = q1.size)} \)
\( \text{\quad \quad return pivot} \)
\( \text{\quad else if (i < q1.size)} \)
\( \text{\quad \quad return select(q1, i)} \)
\( \text{\quad else return select(q2, i-(q1.size+1))} \)
\( \text{\} \}
\( \text{\} \}
\( \text{\} \}
\( \text{\} } \)

select makes two 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 i th element in the list
   (by looking in Q1 or Q2)

\( t(n) \) \( \{ \)
\( \text{select (list, i)} \)
\( \{ \)
\( \text{0} \leq i < \text{list.size} \)
\( \text{if list.size} \leq 5 \), find i th element by brute force and return it \( \text{// base case} \).
\( \text{else} \{ \)
\( \text{\quad partition list into 5-tuples} \)
\( \text{\quad listMediansOf5} \leftarrow \text{find medians of 5-tuples} \)
\( \text{\quad pivot} \leftarrow \text{select} \{ \text{listMediansOf5, listMediansOf5.size}/2 \} \)
\( \text{\quad partition the list around pivot} \)
\( \text{\quad if (i = q1.size)} \)
\( \text{\quad \quad return pivot} \)
\( \text{\quad else if (i < q1.size)} \)
\( \text{\quad \quad return select(q1, i)} \)
\( \text{\quad else return select(q2, i-(q1.size+1))} \)
\( \text{\} \}
\( \text{\} } \)
\( \Rightarrow t(n) < t(\frac{n}{5}) + t(\frac{3n}{10}) + cn \)
\[ t(n) < t\left( \frac{n}{5} \right) + t\left( \frac{3n}{10} \right) + cn \]

We cannot apply Master Theorem.

However, note \( \frac{n}{5} + \frac{3n}{10} < n \)

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

\[ t(n) < t\left( \frac{n}{5} \right) + t\left( \frac{3n}{10} \right) + cn \]
\[ \leq \beta \frac{n}{5} + \beta \cdot \frac{7}{10} n + cn \]
\[ \leq \beta \left( \frac{2n}{10} + \frac{3n}{10} \right) + cn \]
\[ \leq \beta \left( \frac{9n}{10} \right) + cn \]
\[ = 10c \left( \frac{9n}{10} \right) + cn, \text{ where } \beta = 10c \]
\[ = 10c n \]
\[ = \beta n \]

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

Claim:

if \( t(n) < t\left( \frac{n}{5} \right) + t\left( \frac{3n}{10} \right) + cn \) for all \( n \),
then \( t(n) \leq \beta n \) for some \( \beta > 0 \).

Proof: by induction.

base case (easy): \( t(1) \leq \beta \) for some \( \beta \).

(next slide, we'll take \( \beta = 10c \))

induction hypothesis:

Suppose \( t(k) \leq \beta k \) for all \( k < n \)

induction step: show \( t(n) \leq \beta n \)

Returning to Quicksort

We can select the median in \( O(n) \).
Thus, quicksort can be 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 not use median of median of 5's,
however. Why not?

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

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