Search methods

- Sequential ("linear") search
- Binary search

Sequential search

- Easiest and most intuitive search method
- Idea: search through each item in order until you find it, or until the end of the list has been reached.

Sequential search algorithm

For each item in the list,

   compare the current item to the search term
   if the item = search term, stop.  FOUND.

If end of list reached,

   stop.  NOT FOUND.
Sequential search

- Best case: first item is the searched-for item
- Worst case: last item is the searched-for item, OR the search term does not appear in the list.
- Average case: item is somewhere near the middle of the list

A diversion: evaluating the performance of a search algorithm

- We don't evaluate search algorithms by *timing* them
  - too many variations in hardware, software, etc.
- Instead, we *count* the number of *comparisons* and the number of *swaps* that each algorithm does
  - number of comparisons = how many times we compare an item in the list to the search item
  - number of swaps = number of times we move an item in the list
    - only makes sense for sorting algorithms

Sequential search's performance

- Best case:
  - number of comparisons = 1
- Worst case:
  - number of comparisons = N (the size of the list)
- Average case:
  - number of comparisons = N/2 (half the size of the list)

Binary search

- A more efficient search algorithm
- Assumes the list is already sorted
- Idea: Divide the list in “half” until you either find the item or determine that it cannot be in the list
  - we can always rule out half of the remaining possibilities
Binary search algorithm

first=first item in list; last=last item in list
repeat
    find middle of list
    if item at middle = searched-for item, stop. FOUND.
    else, if item at middle > searched-for item,
        set last = item just before the middle item.
    else,
        set first = item just after the middle item.
until item found or last and first “cross” (not in list)

Binary search performance

- Best case: middle item is the searched-for item
  - number of comparisons = 1
- Worst case: item is not in the list, OR item is first or last in list, OR item is on either side of the middle
  - number of comparisons = log N
- Average case: item is anywhere else
  - number of comparisons ~ log N