Searching
Searching an array of integers

• If an array is not sorted, there is no better algorithm than **linear search** for finding an element in it.

```java
static final int NONE = -1; // not a legal index
static int linearSearch(int target, int[] a) {
    for (int p = 0; p < a.length; p++) {
        if (target == a[p]) return p;
    }
    return NONE;
}
```
Searching an array of other Objects

• It’s like searching an array of Strings, provided the \texttt{equals} method has been defined appropriately
  – Remember, you need to override \texttt{public boolean equals(Object o)}

• Again, if the array is not ordered in any meaningful way, a linear search is required
About sorted arrays

• An array is **sorted in ascending order** if each element is no smaller than the preceding element

• An array is **sorted in descending order** if each element is no larger than the preceding element

• When we just say an array is “sorted,” by default we mean that it is sorted in ascending order

• An array of Object **cannot be in sorted order!**
  – There is no notion of “smaller” or “larger” for arbitrary objects
  – *We can* *define* an ordering for some of our objects
  – We’ll talk about **Comparable** objects later
Binary search

• *Linear search* has linear time complexity:
  – Time $n$ if the item is not found
  – Time $\leq n$ if the item is found
• If the array is sorted, we can write a faster search
• How do we look up a name in a phone book, or a word in a dictionary?
  – Look somewhere in the middle
  – Compare what’s there with the thing you’re looking for
  – Decide which half of the remaining entries to look at
  – Repeat until you find the correct place
  – This is the **binary search algorithm**
Example of binary search

• Search the following array \( a \) for 32

\[
\begin{array}{ccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\end{array}
\]
Binary search takes log $n$ time

- In binary search, we choose an index that cuts the remaining portion of the array in half.
- We repeat this until we either find the value we are looking for, or we reach a subarray of size 1.
- If we start with an array of size $n$, we can cut it in half $\log_2 n$ times.
- Hence, binary search has logarithmic ($\log n$) time complexity.
- For an array of size 1000, this is 100 times faster than linear search ($2^{10} \sim = 1000$).
Conclusion

• Linear search has linear time complexity
• Binary search has logarithmic time complexity
• For large arrays, binary search is far more efficient than linear search
  – However, binary search requires that the array be *sorted*
  – If the array *is* sorted, binary search is
    • 100 times faster for an array of size 1000
    • 50 000 times faster for an array of size 1 000 000

• *This* is the kind of speedup that we care about when we analyze algorithms