Searching

Describe a process for determining if an element is contained in this array –

5	3	8	1	2	6	4	7

- *sequential* (or *linear*) *search* – start at the beginning, checking each element in turn
  - must go through the whole array because looking at one element doesn't tell you anything about the others

You are searching for 72 in the following array using *linear search* (or *sequential search*). Which values are checked, in order?

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	5	8	12	16	23	38	45	56	72	91	102	145	168	200

	Answer	Respondents	Percentage
×	2, 5, 8, 12, 16, 23, 38, 45, 56, 72, 91, 102, 145, 168, 200	0	0%
✓	2, 5, 8, 12, 16, 23, 38, 45, 56, 72	12	86%
×	2, 5, 8, 12, 16, 23, 38, 45, 56	1	7%
×	45, 102, 72	0	0%
×	56, 72, 102	1	7%
×	56, 102, 72	0	0%
×	102, 38, 56, 72	0	0%
×	12, 38, 72	0	0%
×	none of the above	0	0%

have to check the slot containing 72 to realize it has been found

sequential search checks every slot from the beginning

Sequential Search

5	3	8	1	2	6	4	7

- start at the beginning, checking each element in turn

Searching

Describe a process for determining if an element is contained in this array –

1	2	3	4	5	6	7	8

- *binary search* – start with the middle element, eliminate half of the elements as either too big or too small, and repeat with the remaining elements
  - in a sorted array, looking at one element lets you eliminate everything on the other side from further consideration
  - looking at the middle element maximizes the guaranteed number of elements eliminated – whether the target is smaller or larger, half the elements will be eliminated
- sequential search can also be used, but is less efficient

You are searching for 72 in the following array using *binary search*. Which values are checked, in order?

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	5	8	12	16	23	38	45	56	72	91	102	145	168	200

	Answer	Respondents	Percentage
×	2, 5, 8, 12, 16, 23, 38, 45, 56, 72, 91, 102, 145, 168, 200	0	0%
×	2, 5, 8, 12, 16, 23, 38, 45, 56, 72	0	0%
×	2, 5, 8, 12, 16, 23, 38, 45, 56	0	0%
✓	45, 102, 72	9	64%
×	56, 72, 102	0	0%
×	56, 102, 72	3	21%
×	102, 38, 56, 72	0	0%
×	12, 38, 72	1	7%
×	none of the above	1	7%

slots checked: *i*=7, *i*=11, *i*=9

i=7 is the middle slot of 15 (`low+high`)/2 = (0+14)/2 = 7

skips by 3s – neither sequential nor binary search

Binary Search

- start with the middle element, eliminate half of the elements as either too big or too small, and repeat with the remaining elements

Question

If an array isn't sorted, is it better to sort it first and then use binary search or to just use sequential search?

It depends!

- binary search is much faster than sequential search, but sorting takes longer than sequential search
  - if you are only going to search a few times, sequential search is faster
  - if you search a lot, the time saved by binary search makes up for the extra time to sort in the first place

Shuffling

Challenge –

- every possible resulting ordering should be equally likely

Strategy –

- moving forward from the end of the array, repeatedly swap each element with a random element from first part of the array up to and including the current element
  - allowing the possibility of not actually swapping the current element is essential

```
/**
 * Postcondition:  The items in A have been rearranged into a random order.
 */
static void shuffle(int[] A) {
   for (int lastPlace = A.length-1; lastPlace > 0; lastPlace--) {
         // Choose a random location from among 0,1,...,lastPlace.
      int randLoc = (int)(Math.random()*(lastPlace+1));
         // Swap items in locations randLoc and lastPlace.
      int temp = A[randLoc];
      A[randLoc] = A[lastPlace];
      A[lastPlace] = temp;
   }
}
```

known as the *Fisher-Yates shuffle* or Knuth shuffle

the implementation given is a more efficient implementation of the originally described algorithm

Sorting, Searching, Shuffling in Java

- the `Arrays` class provides static methods for manipulating arrays
  - `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Arrays.html` or google `java 17 Arrays`

`static void`	**`sort`**`(int[] a)`	Sorts the specified array into ascending numerical order.
`static void`	**`sort`**`(int[] a, int fromIndex, int toIndex)`	Sorts the specified range of the array into ascending order.
`static int`	**`binarySearch`**`(int[] a, int key)`	Searches the specified array of ints for the specified value using the binary search algorithm.
`static int`	**`binarySearch`**`(int[] a, int fromIndex, int toIndex, int key)`	Searches a range of the specified array of ints for the specified value using the binary search algorithm.

- there are similar methods for arrays of other primitive types (`double`, etc)
- we'll cover shuffling and arrays of objects when we cover the Java Collections classes