19CSE100 Problem Solving and Algorithmic Thinking

Searching

Searching is the process of looking for a particular thing in a collection

Searching is a fundamental problem

Variations of Search

Find first occurrence
Find all occurrences
Find the number of occurrences etc.

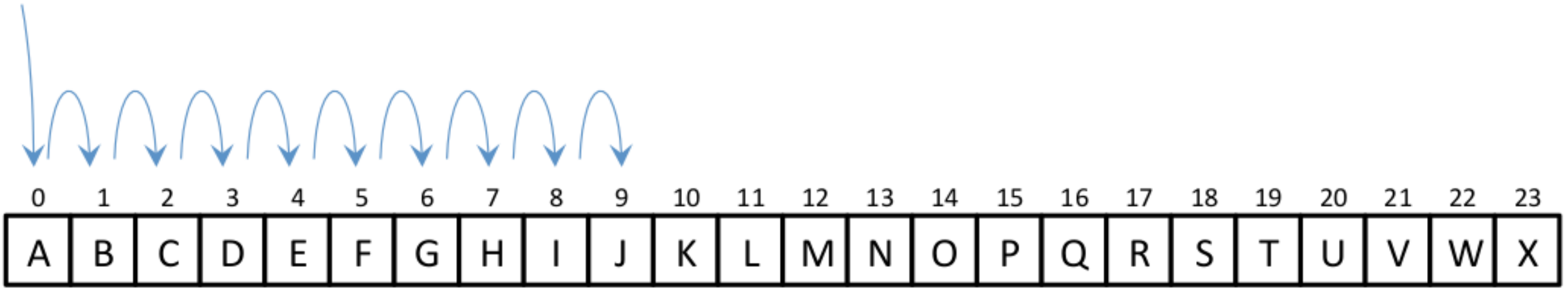
Given a sequence of elements and a target element, find whether the target occurs in the sequence

The collection that we are very familiar is arrays

Exhaustive (Linear) Search

- Systematically enumerate all possible values and compare to value being sought
- For an array, iterate from the beginning to the end, and test each item in the array

Find "J"



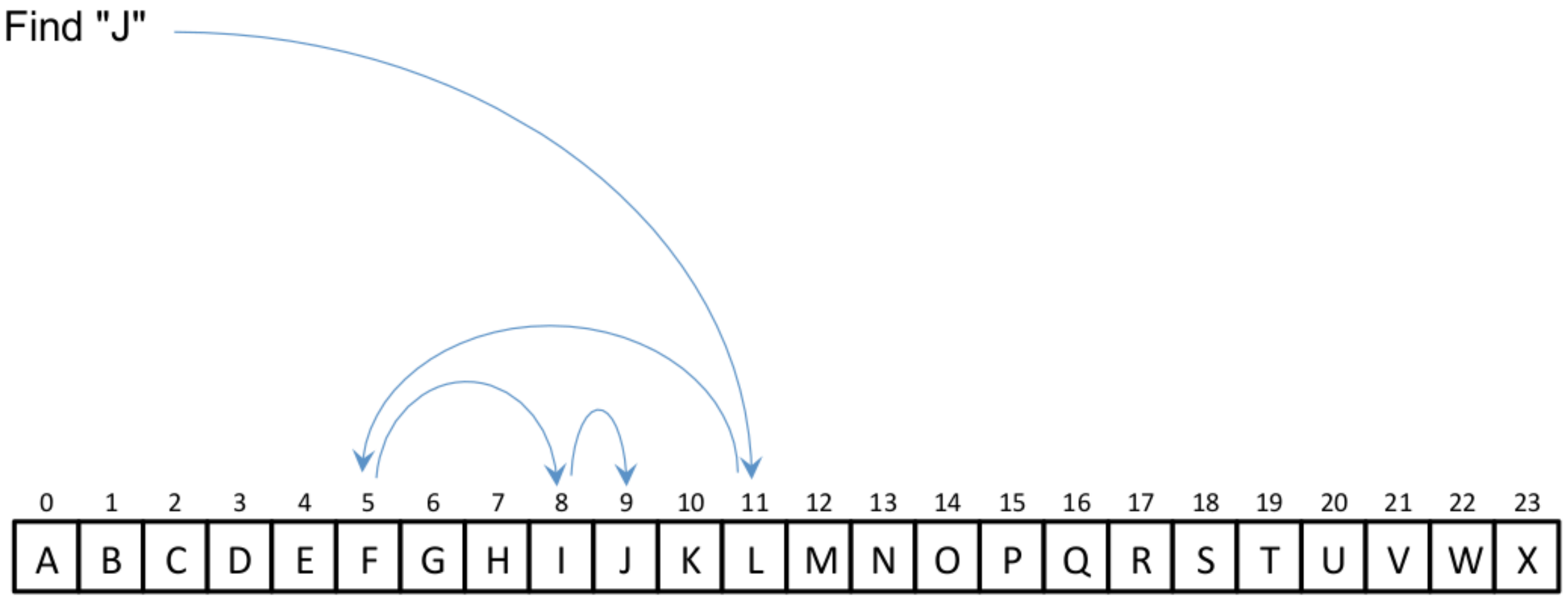
Linear Search Algorithm

```
Start with first array element (index 0)

while(element present in array){
   if value found at current index
      return index
   else
      try next element(increment index)
}

if value not found return -1
```

Linear Search Algorithm

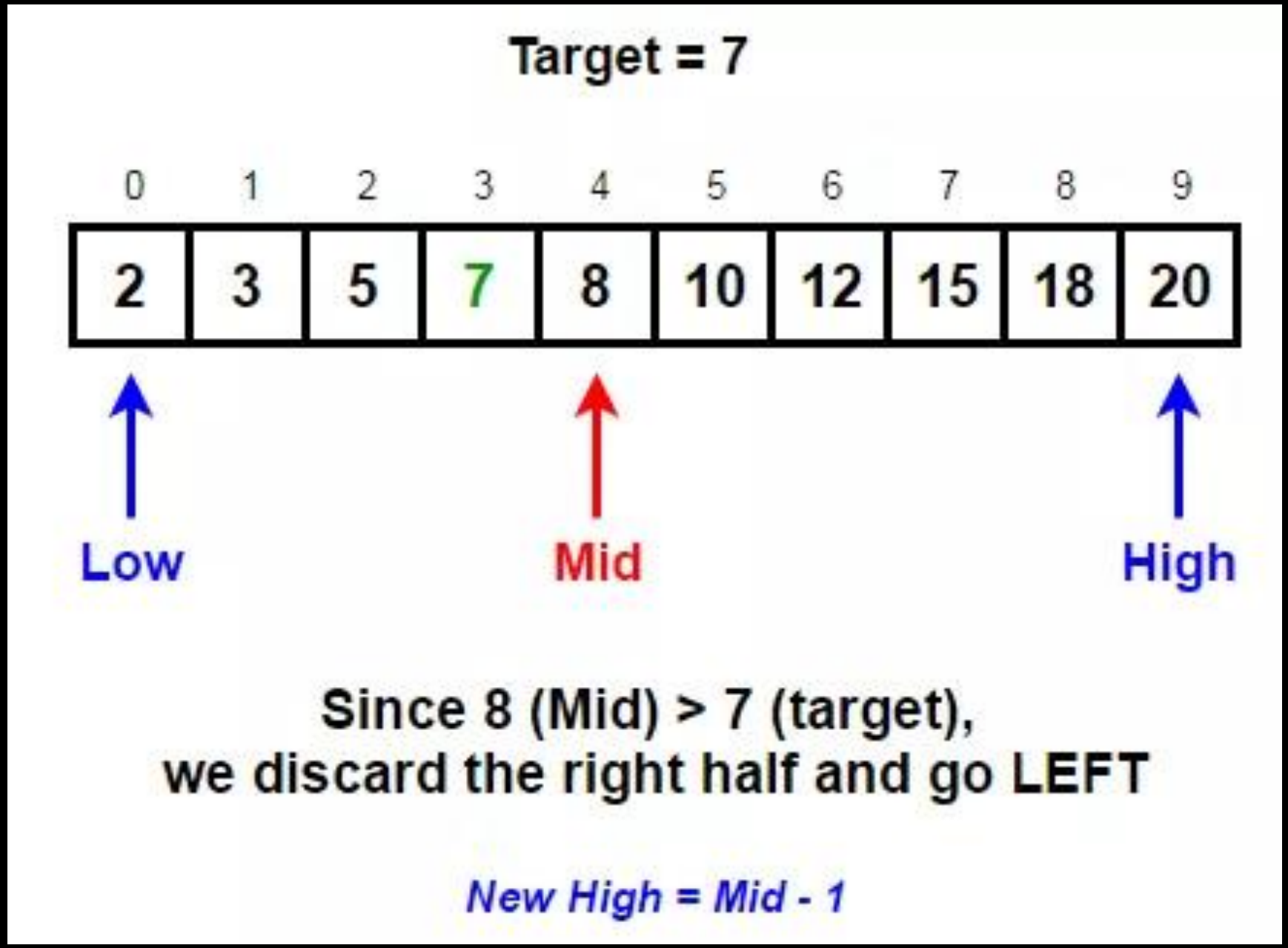
```
Start with first array element (index 0)
while(element present in array){
   if value at current index greater than
   value
      return -1 and break
   else if value found at current index
      return index
   else
      try next element(increment index)
}
if value not found return -1
```

Binary Search



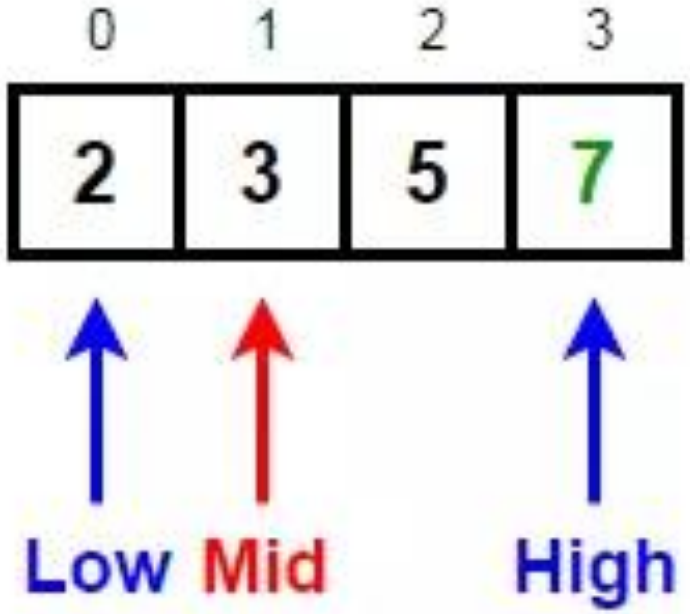
Binary Search Algorithm

```
init low and high variables to lowest and
highest index
while(low <= high){
   compare middle and value
   If (value = middle) return middle
   If (value < middle) set high = middle-1
   If (value > middle) set low  = middle+1
}
if (low > high) value not found
```



https://www.techiedelight.com/binary-search/

Target = 7

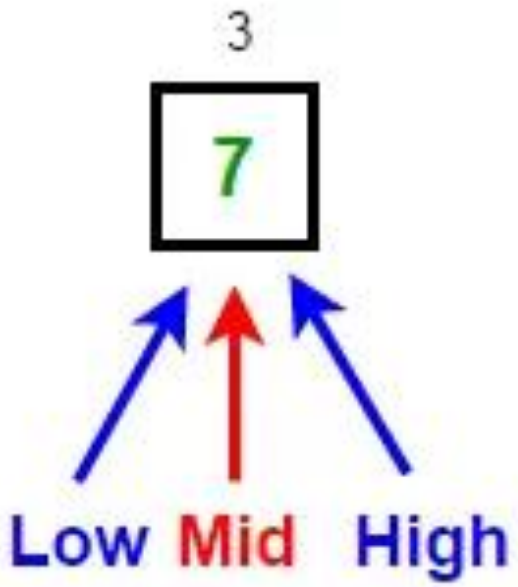


Since 3 (Mid) < 7 (target),
We discard the left half and go RIGHT

New low = mid + 1

https://www.techiedelight.com/binary-search/

Target = 7



Now our search space consists of only one element 7. Since 7 (Mid) = 7 (target), we return index of 7 i.e. 3 and terminate our search

Efficiency – Linear Search

- Given a deck of 52 playing cards, how many do you have to look at to find the Ace of Spades?

- Best case: 1
- Worst case: 52
- Average case: ?

Efficiency – Linear Search

Average Case

- To find the average case:
  1. Enumerate all possible cases
  2. Add the number of cards looked at for all of them
  3. Divide by the number of cases

Efficiency – Linear Search

1. Enumerate all possible cases

   1, 2, 3, 4 … 50, 51, 52

2. Add the number of cards looked at for all of them

   1 + 2 + … + 52 = (52 + 1) * (52 / 2) = 1378

3. Divide by the number of cases

   1378 / 52 = 26.5

Efficiency – Linear Search

- Given a deck of 52 playing cards, how many do you have to look at to find the Ace of Spades?

- Best case: 1
- Worst case: 52
- Average case: 26.5

- The number of comparisons is a better metric than the time it takes, because people (computers) may be quicker or slower to pick up each card and look at it

Efficiency – Binary Search

- Now assume the deck of cards is brand new and sorted by suit and number
  - E.g. 2-A♣, 2-A♦, 2-A♥, 2-A♠
  - Best case: 1
  - Worst case: 1
  - Average case: 1
- Organized data allows for more predictable, more efficient algorithms

Efficiency – Linear Search

- Given 128 index cards with random numbers on them, to find a particular number (if it is there) with linear search:
  - Best case: 1
  - Worst case: 128
  - Average case: 64.5

Efficiency – Binary Search

- With every iteration of binary search, you cut the size of the search in half
  - e.g. 128, 64, 32, 16, 8, 4, 2, 1
- 128 can be divided in half 7 times
  - In other terms, $128 = 2^7$
  - So, the worst case number of checks for binary search is:
    - $\log_2 n + 1$
  - With 2 comparisons per check (equal and less than)
    - $2(\log_2 128 + 1) = 16$ comparisons

Efficiency – Linear Search

- Linear search *scales* linearly with the number of items you search over (n)
  - Worst case number of comparisons = n
  - Average case comparisons = ½(n + 1)

Items	Worst case	Average case
10	10	5.5
100	100	50.5
1000	1000	500.5
10000	10000	5000.5

Efficiency – Binary Search

- Binary search *scales* logarithmically with the number of items you search over (n)
  - Worst case number of comparisons = $2(\log_2 n + 1)$

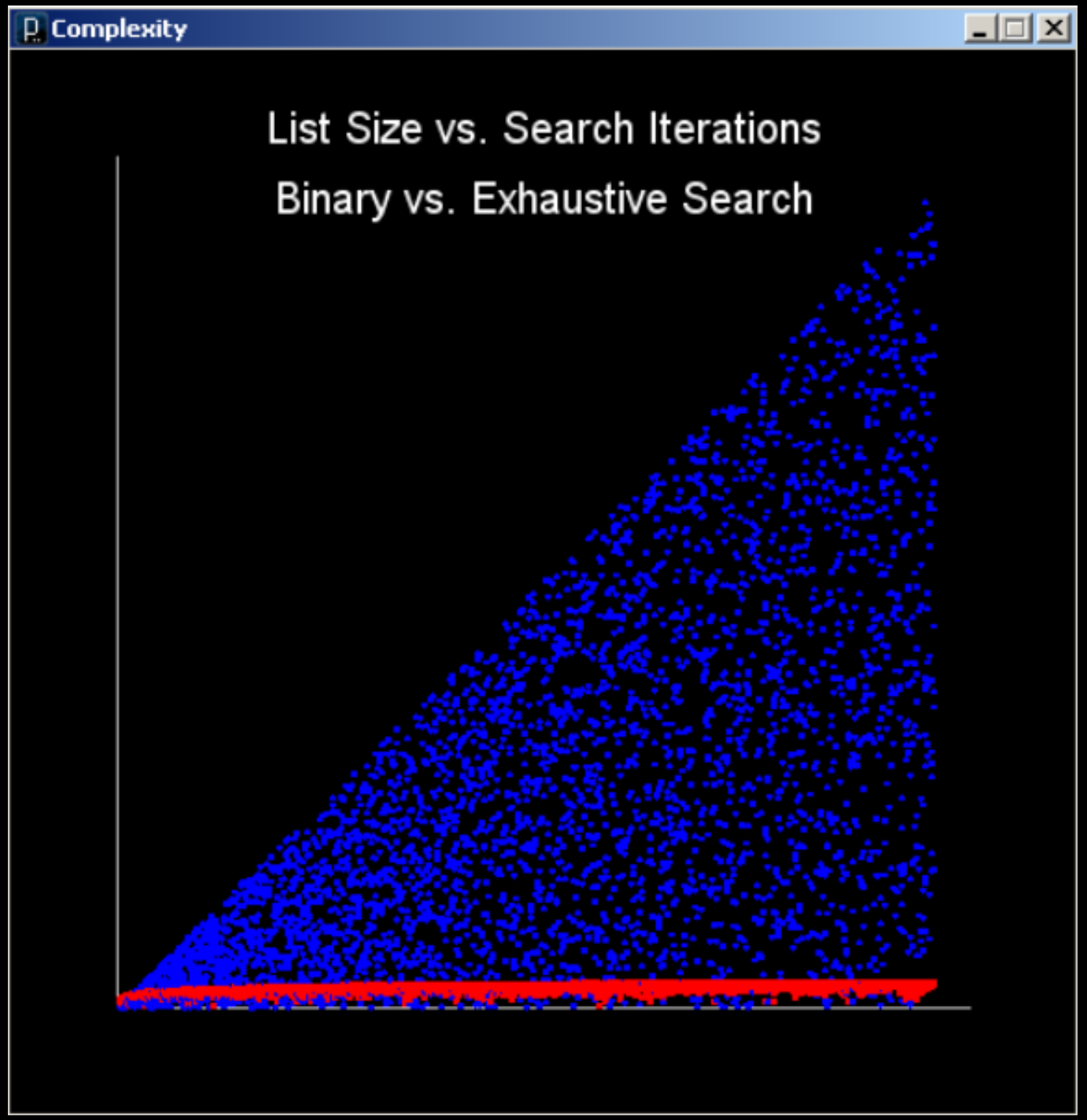
Items	Worst case linear	Worst case binary
10	10	9
100	100	15
1000	1000	22
10000	10000	29

- Binary search scales much better than linear search

An Experiment - Exhaustive vs. Binary Search

- For names (Strings) in arrays of increasing size…
  - Select 10 names at random from the list
  - Search for each name using Binary and Exhaustive Search
  - Count the number of iterations it takes to find each name
  - Plot number of iterations for each against list size

- Start with an array of 3830+ names (Strings)



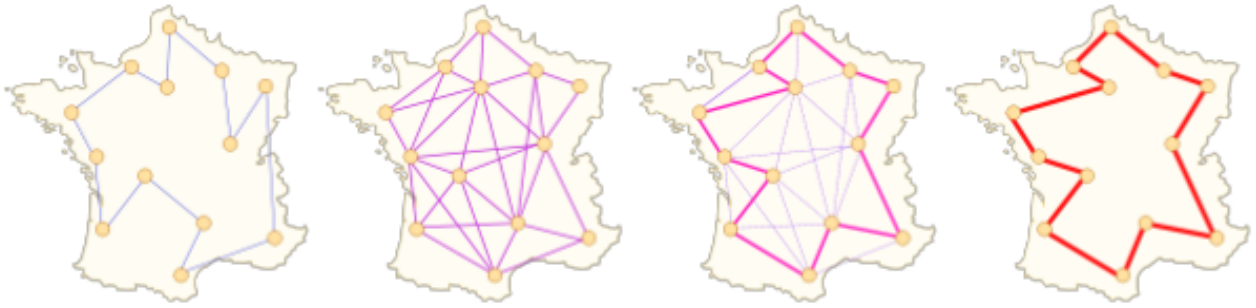
That's why effeciency is most sought after

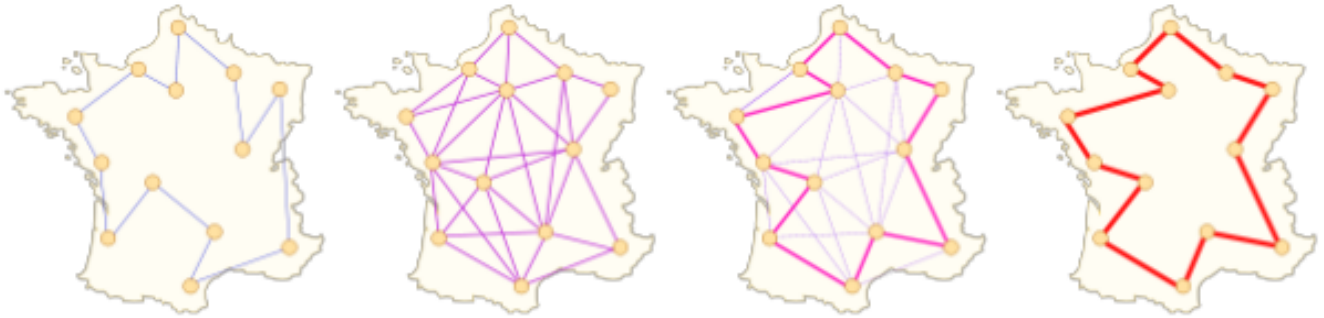
Why Do We Care?

- Computational complexity
  - Studying types of problems and how hard they are to solve
  - This is a key part of computer science beyond programming
- Classifying problems helps us:
  - Identify general strategies for solving problem types
  - Avoid wasting time developing programs that can’t work

The Traveling Salesman

- Consider a salesman who wants to visit a bunch of small towns
  - He has a list of distances between pairs of towns
  - How should he figure out what order to visit them in?
    - How long do you think it would take to figure out the best path for 25 towns?





1 2 3 4

- The brute-force solution to this problem is *O(n!)*, which is even worse than exponential
  - 25 towns = around a billion years on a 2GHz computer
  - Some more clever solutions are $O(n^2)$
- Reasonable solutions can only find an approximately best path