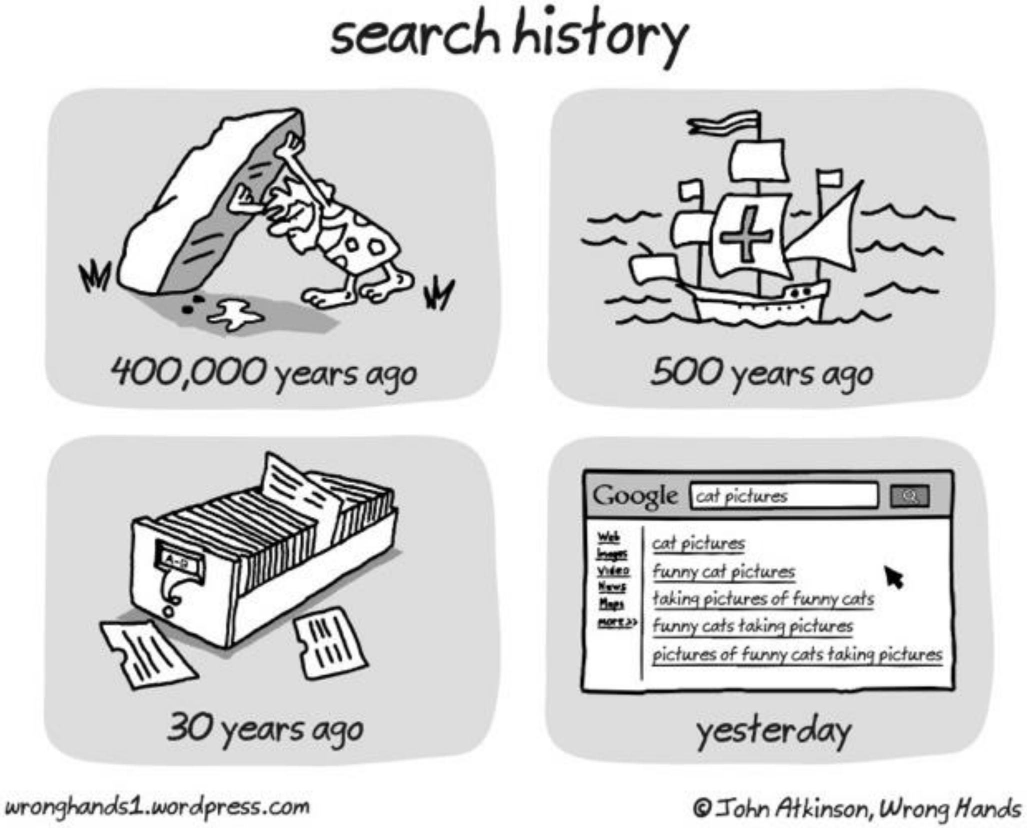
CSc 110, Spring 2018

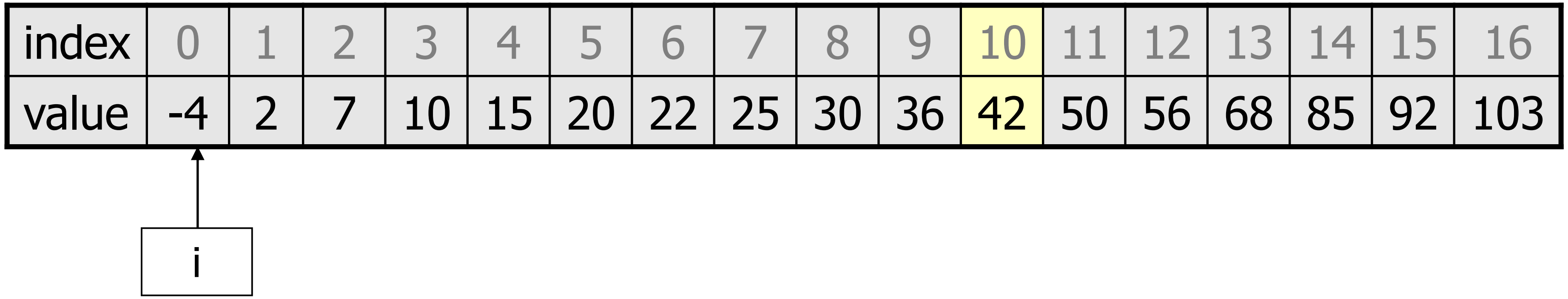
Lecture 38: searching

Adapted from slides by Marty Stepp and Stuart Reges


search history
400,000 years ago
500 years ago
30 years ago
A-B
Google
cat pictures
Web
Images
Video
News
Maps
more>>
cat pictures
funny cat pictures
taking pictures of funny cats
funny cats taking pictures
pictures of funny cats taking pictures
yesterday
wronghands1.wordpress.com
©John Atkinson, Wrong Hands

Sequential search

- **sequential search**: Locates a target value in a list by examining each element from start to finish. Used in `index`.
  - How many elements will it need to examine?
  - Example: Searching the list below for the value **42**:



index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103

Sequential search

- How many elements will be checked?

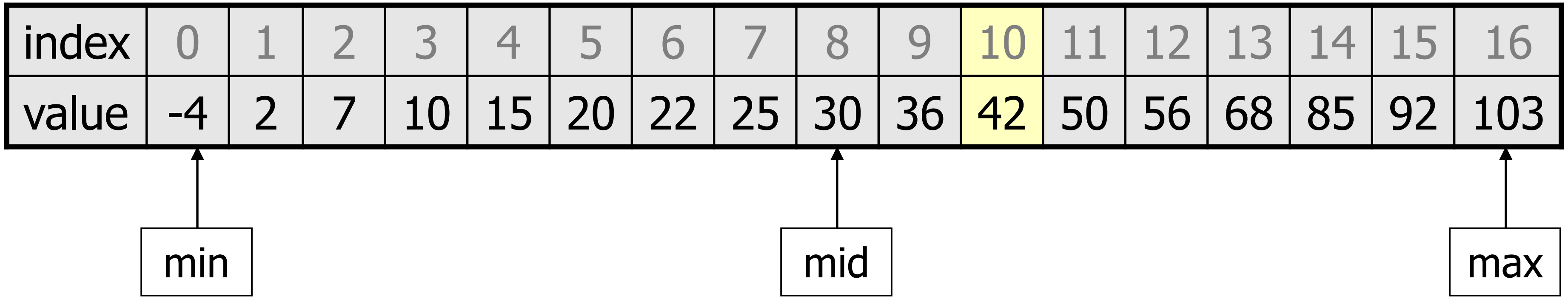
```
def index(value):
    for i in range(0, size):
        if my_list[i] == value:
            return i
    return -1   # not found
```

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103

- On average how many elements will be checked?

Binary search

- **binary search**: Locates a target value in a *sorted* list by successively eliminating half of the list from consideration.
  - How many elements will it need to examine?
  - Example: Searching the list below for the value **42**:



index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103

min

mid

max

Binary search runtime

- For an list of size N, it eliminates ½ until 1 element remains.

  N, N/2, N/4, N/8, ..., 4, 2, 1

  - How many divisions does it take?

- Think of it from the other direction:
  - How many times do I have to multiply by 2 to reach N?

    1, 2, 4, 8, ..., N/4, N/2, N
  - Call this number of multiplications "x".

    $2^x = N$

    $\mathbf{x = \log_2 N}$

- Binary search looks at a **logarithmic** number of elements

binary_search

Write the following two functions:

```
# searches an entire sorted list for a given value
# returns the index the value should be inserted at to maintain sorted order
# Precondition: list is sorted
binary_search(list, value)

# searches given portion of a sorted list for a given value
# examines min_index (inclusive) through max_index (exclusive)
# returns the index of the value or -(index it should be inserted at + 1)
# Precondition: list is sorted
binary_search(list, value, min_index, max_index)
```

Using `binary_search`

```
# index 0   1   2   3    4    5    6    7    8    9   10   11   12   13   14   15
a   =  [-4, 2,  7,  9,  15,  19,  25,  28,  30,  36,  42,  50,  56,  68,  85,  92]

index1 = binary_search(a, 42)
index2 = binary_search(a, 21)
index3 = binary_search(a, 17, 0, 16)
index2 = binary_search(a, 42, 0, 10)
```

- `binary_search` returns the index of the number

or

- (index where the value should be inserted + 1)

Binary search code

```
# Returns the index of an occurrence of target in a,
# or a negative number if the target is not found.
# Precondition: elements of a are in sorted order
def binary_search(a, target, start, stop):
    min = start
    max = stop - 1

    while min <= max:
        mid = (min + max) // 2
        if a[mid] < target:
            min = mid + 1
        elif a[mid] > target:
            max = mid - 1
        else:
            return mid   # target found

    return -(min + 1)    # target not found
```