# CS416

## Introduction to Computer Science II

Spring 2018

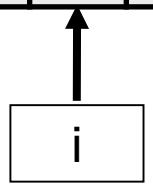
# 6c Binary search and complexity

Adapted from course-note CSE 143, University of Washington

# Sequential search

- **Sequential search**: Locates a target value in an array / list by examining each element from start to finish. Used in `indexOf and contains` methods.

| 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 |

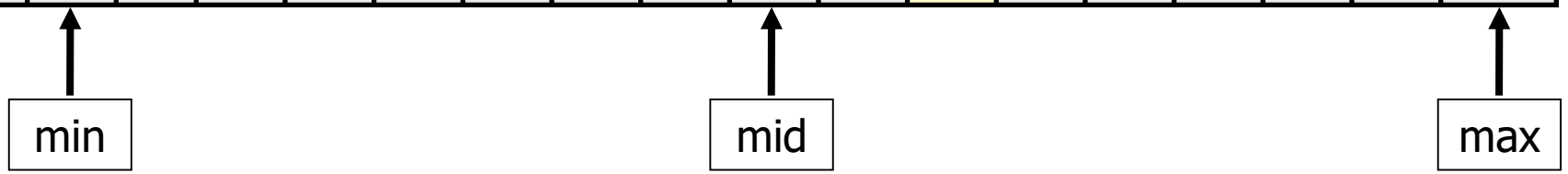



  - How many elements will it need to examine? E.g. Searching the array above for the value **42**.
    - 11 in this case. On average, it would be *n/2* where *n* is size of the array -- **if** number is there
  - Notice that this particular array is sorted.
  - Can we take advantage of this?

# Binary Search

- **Binary Search**: Locates a target value in a sorted array / list by successively eliminating half of the array from consideration.
  - How many elements will it need to examine?
  - Example: Searching the array 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 |




3 comparisons

# Arrays.binarySearch

- The binarySearch method in the `Arrays` class searches an array very efficiently if the array is sorted.
- You can search the entire array, or just a range of indexes

```
// searches an entire sorted array for a given value
// returns its index if found; a negative number if not found
// Precondition: array is sorted
Arrays.binarySearch(array, value)
// searches given portion of a sorted array for a given value
// examines minIndex (inclusive) through maxIndex (exclusive)
// returns its index if found; a negative number if not found
// Precondition: array is sorted
Arrays.binarySearch(array, minIndex, maxIndex, value)
```

# Using **`binarySearch`**

```
// index  0  1  2  3   4   5   6   7   8   9  10  11  12  13  14  15
int[] a={-4, 2, 7, 9, 15, 19, 25, 28, 30, 36, 42, 50, 56, 68, 85, 92};
int index  = Arrays.binarySearch(a, 0, 16, 42);   // index1 is 10
int index2 = Arrays.binarySearch(a, 0, 16, 21);   // index2 is -7
```

- `binarySearch` returns the index where the value is found
- if the value is not found, `binarySearch` returns:

  `-(insertionPoint + 1)`

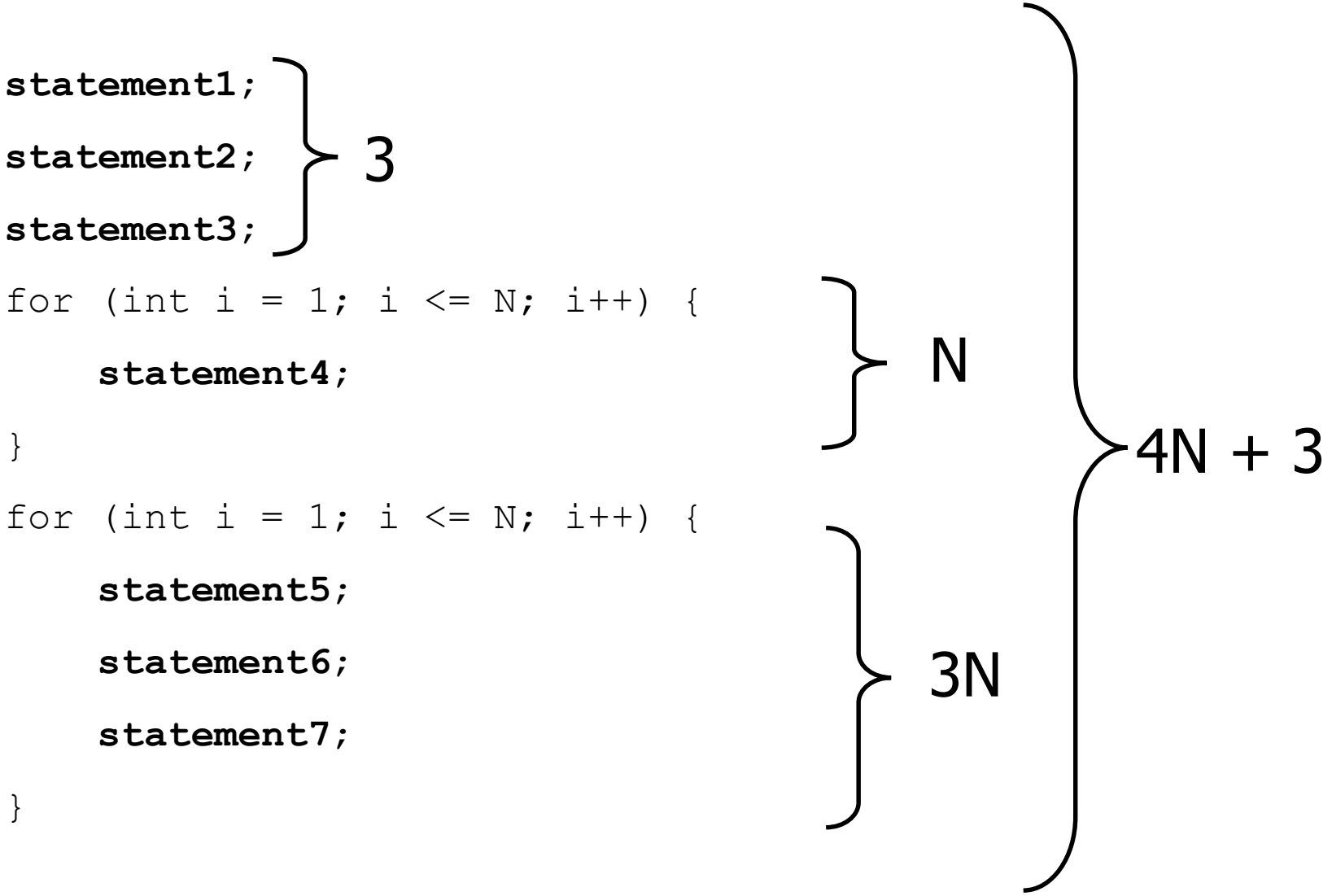
  - where insertionPoint is the index where the element would have been in the array in sorted order.
  - To insert the value into the array, negate `(insertionPoint + 1 )`

```
int indexToInsert21 = -(index2 + 1); // 6
```

# Runtime Efficiency

- How much better is binary search than sequential search?
- **Efficiency**: A measure of the use of computing resources by code.
  - can be relative to speed (time), memory (space), etc.
  - most commonly refers to run time
- Basic assumption:
  - Any single Java statement takes the same amount of time to run.
  - A method call's runtime is measured by the total of the statements inside the method's body.
  - A loop's runtime, if the loop repeats *N* times, is *N* times the runtime of the statements in its body.

# Efficiency examples



# Efficiency examples 2

```
for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= N; j++) {
        statement1;
    }
}
for (int i = 1; i <= N; i++) {
    statement2;
    statement3;
    statement4;
    statement5;
}
```

$N^2$

4N

$N^2 + 4N$

- How many statements will execute if N = 10? If N = 1000?

# Algorithm growth rates

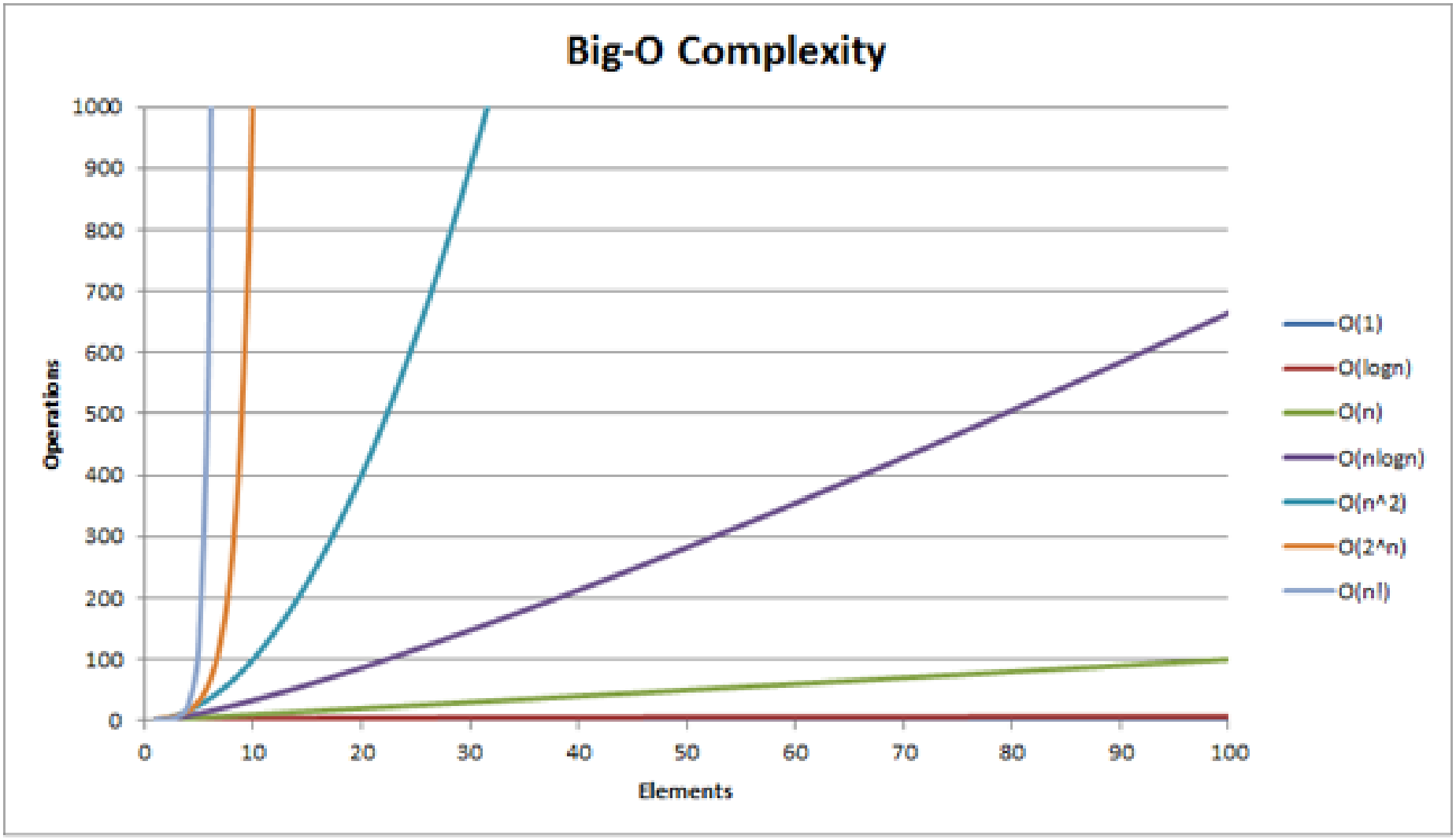
- We measure runtime in proportion to the input data size, N.
  - **growth rate**: Change in runtime as N changes.
- Say an algorithm runs **$0.4N^3 + 25N^2 + 8N + 17$** statements.
  - Consider the runtime when N is extremely large .
  - We ignore constants like 25 because they are tiny next to N.
  - The highest-order term ($N^3$) dominates the overall runtime.
  - We say that this algorithm runs "on the order of" $N^3$.
  - or **$O(N^3)$** for short ("**Big-Oh** of N cubed")

# Complexity classes

- **complexity class**: A category of algorithm efficiency based on the algorithm's relationship to the input size N.

| Class | Big-Oh | If you double N, ... | Example |
|---|---|---|---|
| constant | $O(1)$ | unchanged | 10ms |
| logarithmic | $O(\log_2 N)$ | increases slightly | 175ms |
| linear | $O(N)$ | doubles | 3.2 sec |
| log-linear | $O(N \log_2 N)$ | slightly more than doubles | 6 sec |
| quadratic | $O(N^2)$ | quadruples | 1 min 42 sec |
| cubic | $O(N^3)$ | multiplies by 8 | 55 min |
| ... | ... | ... | ... |
| exponential | $O(2^N)$ | multiplies drastically | $5 * 10^{61}$ years |

# Complexity classes



# Sequential search

- What is its complexity class?

```
public int indexOf(int value) {
    for (int i = 0; i < size; i++) {
        if (elementData[i] == value) {
            return i;
        }
    }
    return -1;   // not found
}
```

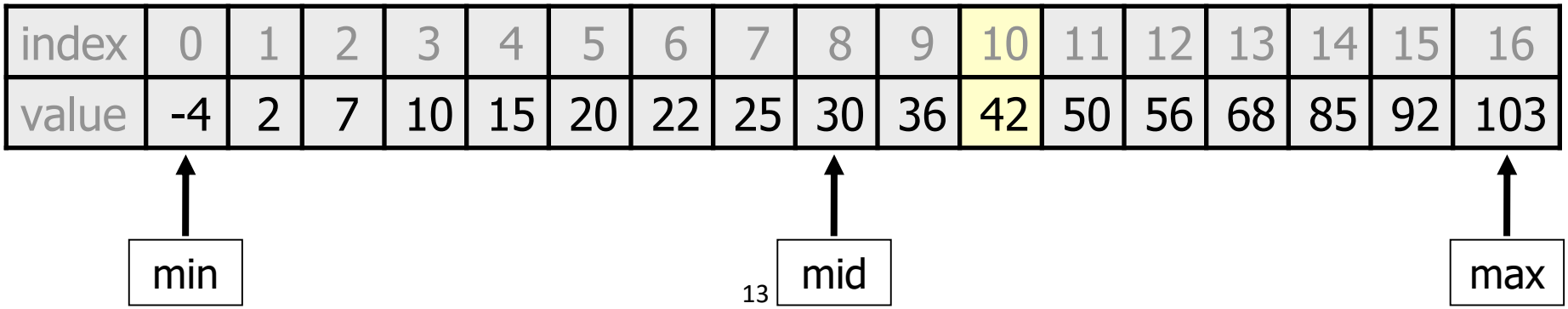
N

| 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, N/2 elements are visited
  - 1/2 is a constant that can be ignored

# Binary search

- **Binary search** successively eliminates half of the elements.
- Algorithm: Examine the middle element of the array.
  - If it is too big, eliminate the right half of the array and repeat.
  - If it is too small, eliminate the left half of the array and repeat.
  - Else it is the value we're searching for, so stop.
- Which indexes does the algorithm examine to find value **42**?
- What is the runtime complexity class of binary search?



# Binary search runtime

- For an array 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 is in the **logarithmic** complexity class.