SEARCHING
AND
SORTING

Searching

- A basic operation in computer science
- Problem formulation
  - Given:
    - Collection (array) of values
    - A property on values
  - Find: a value having the property in the array
- Examples
    - Find a particular element
    - Find the least element

- We will study different algorithms for solving this problem

The Search Game

- Pick an element in the range 0 … 1,023
- How long does it take to guess?

- This is a search problem
- The collection: numbers 0 … 1,023
- The “property”: number chosen by the other party

Searching

- Searching is the process of determining whether or not a given value exists in a data structure or a storage media.

- We discuss two searching methods on one-dimensional arrays: linear search and binary search.

Linear Search

- The linear (or sequential) search algorithm on an array is:
  - Sequentially scan the array, comparing each array item with the searched value.
  - If a match is found; return the index of the matched element; otherwise return –1.

- Note: linear search can be applied to both sorted and unsorted arrays.

The Search Game: Linear Search

- Start at 0
- Guess until you find the element!

- Answer to guess is “yes / no”
  - Next guess is one more than previous guess
    - How long will it take?

- This kind of algorithm is called linear search

Implementing Linear Search

```
package javaapplication10;
public class Linearsearch {
public static void main(String[] args) {
        int array []={7,5,3,1,6,9};
 int key=11;
 {
    for(int k = 0; k < array.length; k++){
      if(array[k]==(key))
       break;
}
                        if(k == array.length)
System.out.println("Can't find " + key);
else
System.out.println("Found " + key);

   }}
```

Binary Search

- Binary search uses a recursive method to search an array to find a specified value
- The array must be a sorted array:

  `a[0]≤a[1]≤a[2]≤. . . ≤ a[finalIndex]`
- If the value is found, its index is returned
- If the value is not found, -1 is returned
- Note: Each execution of the recursive method reduces the search space by about a half

Binary Search

- An algorithm to solve this task looks at the middle of the array or array segment first
- If the value looked for is smaller than the value in the middle of the array
  - Then the second half of the array or array segment can be ignored
  - This strategy is then applied to the first half of the array or array segment

Binary Search

- If the value looked for is larger than the value in the middle of the array or array segment
  - Then the first half of the array or array segment can be ignored
  - This strategy is then applied to the second half of the array or array segment
- If the value looked for is at the middle of the array or array segment, then it has been found
- If the entire array (or array segment) has been searched in this way without finding the value, then it is not in the array

The Search Game Revised: Binary Search

- Remember lower, upper bound
- Guess middle element
  - Answer is “yes / higher / lower”
  - If “higher”, adjust lower bound
  - If “lower”, adjust higher bound
    - How long does it take?
- This kind of algorithm is called binary search

Display 11.6 Recursive Method for Binary Search

```
1  public class BinarySearch
2  {
3      /**
4       Searches the array a for key. If key is not in the array segment, then -1 is
5       returned. Otherwise returns an index in the segment such that key == a[index].
6       Precondition: a[first] <= a[first + 1]<= ... <= a[last]
7      */
8      public static int search(int[] a, int first, int last, int key)
9      {
10         int result = 0; //to keep the compiler happy.

11         if (first > last)
12             result = -1;
13         else
14         {
15             int mid = (first + last)/2;

16             if (key == a[mid])
17                 result = mid;
18             else if (key < a[mid])
19                 result = search(a, first, mid - 1, key);
20             else if (key > a[mid])
21                 result = search(a, mid + 1, last, key);
22         }
23         return result;
24     }
25 }
```

Iterative Version of Binary Search (Part 1 of 2)

Display 11.9 Iterative Version of Binary Search

```
1  /**
2   Searches the array a for key. If key is not in the array segment, then -1 is
3   returned. Otherwise returns an index in the segment such that key == a[index].
4   Precondition: a[lowEnd] <= a[lowEnd + 1]<= ... <= a[highEnd]
5  */
6  public static int search(int[] a, int lowEnd, int highEnd, int key)
7  {
8      int first = lowEnd;
9      int last = highEnd;
10     int mid;

11     boolean found = false; //so far
12     int result = 0; //to keep compiler happy

13     while ( (first <= last) && !(found) )
14     {
15         mid = (first + last)/2;
```

Iterative Version of Binary Search (Part 2 of 2)

Display 11.9 Iterative Version of Binary Search (continued)

```
16            if (key == a[mid])
17            {
18                found = true;
19                result = mid;
20            }
21            else if (key < a[mid])
22            {
23                last = mid - 1;
24            }
25            else if (key > a[mid])
26            {
27                first = mid + 1;
28            }
29        }

30        if (first > last)
31            result = -1;

32        return result;
33    }
```

Which Search Is Better?

- For linear search: no need for sorted array
- For binary search: *faster … or is it?*
- How many guesses needed in search game (worst case) for
  - Linear search?
  - Binary search

Answers

- For linear search: 1,024
  - Number chosen may be 1,023
  - This would require each number to be guessed
- For binary search: 10
  - Each guess rules out half of the remaining possibilities
  - Total number of possibilities: 1,024
  - This can be cut in half at most 10 times

How many guesses needed in worst case for:

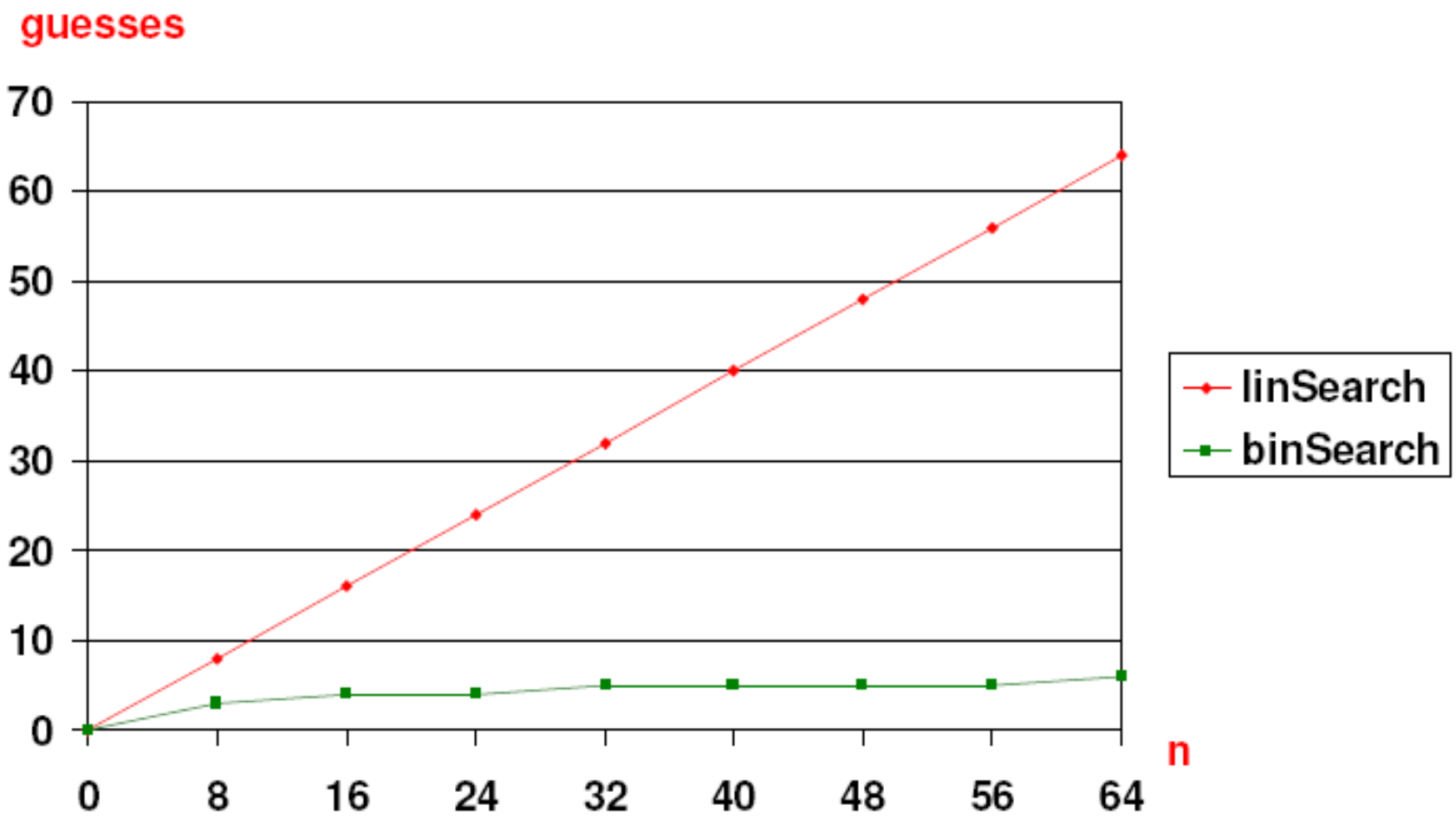
- Linear search
- Binary search

Answers

- Linear search: n
- Binary search: log2n

log2n = number of times n can be cut in half

Number of Guesses as n Grows



Sorting

- Sorting = ordering.
- Sorted = ordered based on a particular way.
- Generally, collections of data are presented in a sorted manner.

- Why?
  - Imagine finding the phone number of your friend in your mobile phone, but the phone book is not sorted.

Examples

- Examples of Sorting:
  - Words in a dictionary are sorted (and case distinctions are ignored).
  - Files in a directory are often listed in sorted order.
  - The index of a book is sorted (and case distinctions are ignored).
  - Many banks provide statements that list checks in increasing order (by check number).
  - In a newspaper, the calendar of events in a schedule is generally sorted by date.
  - Musical compact disks in a record store are generally sorted by recording artist.

Sorting - Definitions

- Input: You are given an array A of data records, each with a key (which could be an integer, character, string, etc.).
  - There is an *ordering* on the set of possible keys
  - You can compare any two keys using <, >, ==

- For simplicity, we will assume that A[i] contains only one element – the key

- Sorting Problem: Given an array A, output A such that:
  - For any i and j, if i < j then A[i] <= A[j]

- *Internal sorting*: all data in main memory
- *External sorting:* data on disk

Why Sort?

- Sorting algorithms are among the most frequently used algorithms in computer science
  - Crucial for efficient retrieval and processing of large volumes of data, e.g., Database systems

- Allows binary search of an N-element array in O(log N) time

- Allows O(1) time access to k^{th} largest element in the array for any *k*

- Allows easy detection of any duplicates

Bubble Sort: Idea

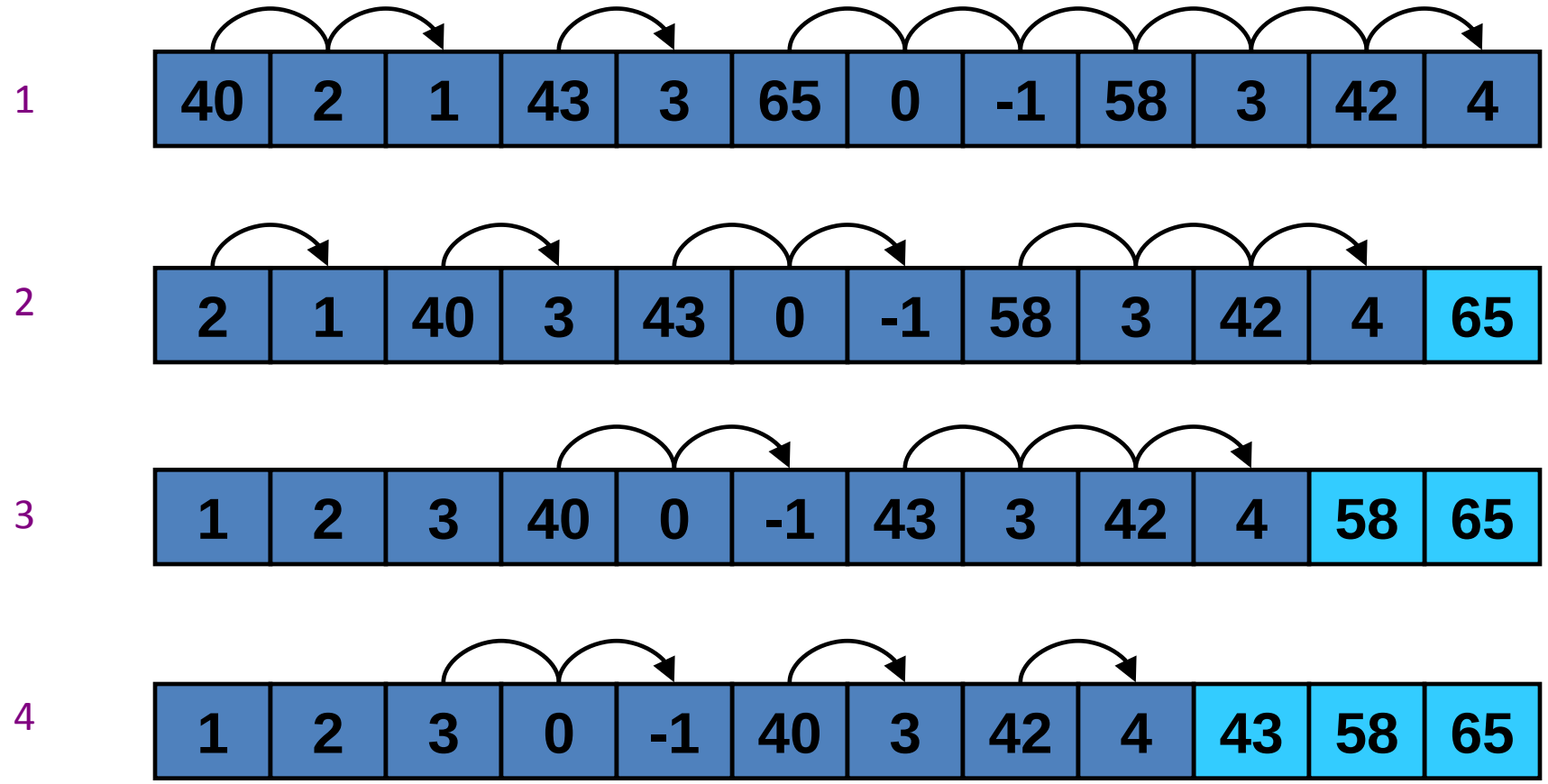
- Idea: “Bubble” larger elements to end of array by comparing elements i and i+1, and swapping if A[i] > A[i+1]

  – Repeat from first to end of unsorted part

Bubble Sort: Implementation

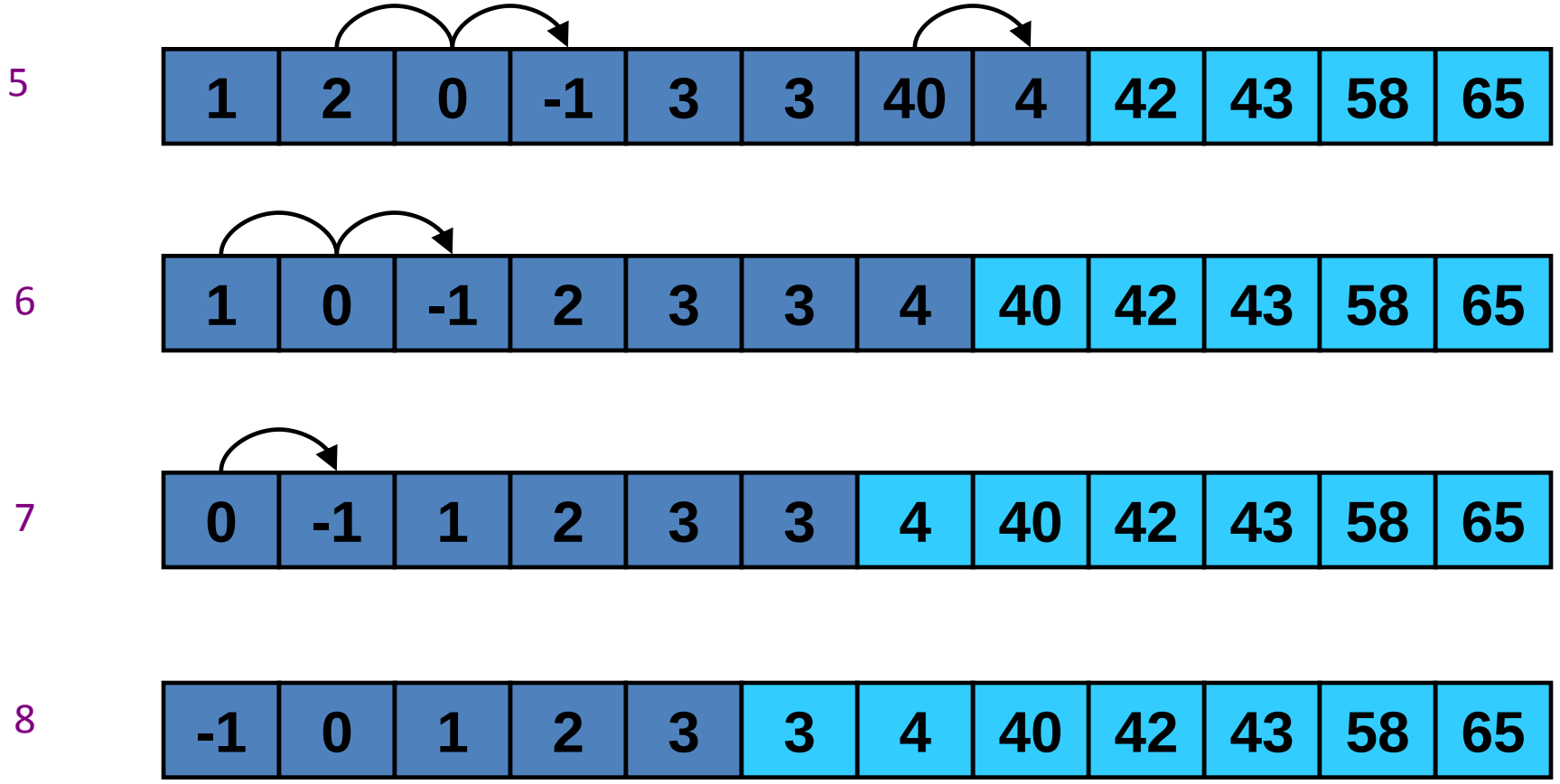
```
void sort(int a[]){

 for (int i = a.length; i>=0; i--) {

        for (int j = 0; j<i; j++) {

            if (a[j] > a[j+1]) {
                int T = a[j];
                a[j] = a[j+1];
                a[j+1] = T;
                swapped = true;

}

}
```

Bubble Sort: Example



- Notice that at least one element will be in the correct position each iteration.

Bubble Sort: Example



Bubble Sort: Analysis

- Running time
  - Worst case: $O(N^2)$
  - Best case: $O(N^2)$ --

Stable Sorting

- A property of sorts
- If a sort guarantees the relative order of equal items stays the same then it is a *stable sort*

- That is, a **sorting** algorithm is ***stable*** if whenever there are two records *R* and *S* with the same key and with *R* appearing before *S* in the original list, *R* will appear before *S* in the sorted list

- $[7_1, 6, 7_2, 5, 1, 2, 7_3, -5]$
  - subscripts added for clarity
- $[-5, 1, 2, 5, 6, 7_1, 7_2, 7_3]$
  - result of stable sort