Searching and Sorting

- **Linear Search**
- **Binary Search**
- **Selection Sort**
- **Insertion Sort**
- **Bubble (or Exchange) Sort**
- **Exercises**

Linear Search

- 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.
- 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.
- The algorithm translates to the following Java method:

```
public static int linearSearch(Object[] array, Object key){
  for(int k = 0; k < array.length; k++)
     if(array[k].equals(key))
        return k;
  return -1;
}
```

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

Binary Search

- The binary search algorithm can only be applied to an array that is sorted; furthermore, the order of sorting must be known.
- A recursive binary search algorithm for an array sorted in ascending order is:

–if(there are more elements in the current sub-array){
–Compare the current middle sub-array element with key (the value searched for).
–There are three possibilities:
  - 1. key == array[middle]
  - the search is successful, return middle.
  - 2. key < array[middle]
  - binary search the current lower half of the array.
  - 3. key > array[middle]
  - binary search the current upper half of the array.
– }
– if no elements left in sub-array, thus the key is not in the array; return –1

Binary Search (cont'd)

The algorithm translates to the following Java method : Here the objects are sorted according to the comparator, another version can be written for objects sorted based on the natural ordering

```
public static int binarySearch(Object[] array, Object key,
    Comparator comparator){
   return binarySearch(array, key, 0, array.length - 1, comparator);
}
private static int binarySearch(Object[] array, Object key,
     int low, int high, Comparator comparator){
   if(low > high)
     return -1;
   else{
     int middle = (low + high)/2;
     int result = comparator.compare(key, array[middle]);
     if(result == 0) return middle;
     else if(result < 0)
        return binarySearch(array, key, low, middle - 1, comparator);
     else
        return binarySearch(array, key, middle + 1, high, comparator);
   }
}
```

Selection Sort

- Sorting is the process of arranging data in a data structure or a storage media such that it is in increasing or decreasing order for primitive types, or according to the natural ordering if the class of the sorted objects implements Comparable, or according to a comparator object that defines the criteria of sorting.
- We discuss five sorting algorithms on one-dimensional arrays starting with Selection Sort.
- The pseudo-code for Selection sort algorithm to sort an array in increasing is:

```
selectionSort(array){
    for(k = 0; k < array.length – 1; k++){
      select the minimum element among array[k]...array[array.length – 1];
      swap the selected minimum with x[k];
    }
}
```

- To sort an array in decreasing order, the maximum element is selected in each iteration (or pass) of the algorithm.

Selection Sort (cont'd)

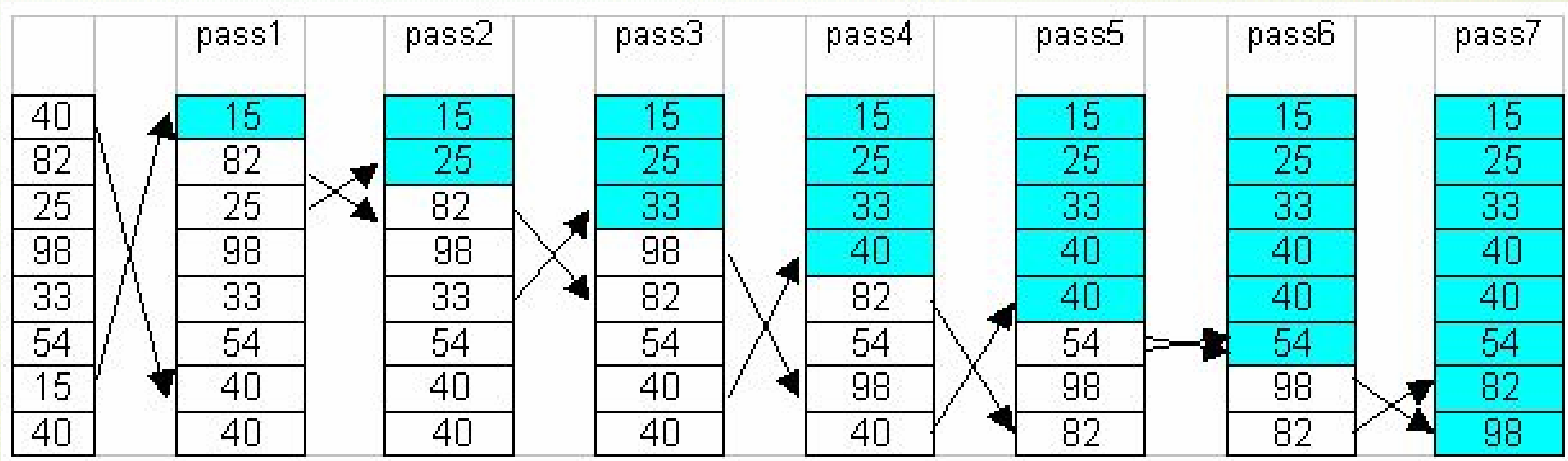
The algorithm translates to the following java method for Comparable objects.

Another version can be written using a Comparator object as done previously in searching, or dealing with primitive types

```
public static void  selectionSort(Comparable[] array )   {
    int minPos = 0;
    Object temp;
    for(int i = 0; i < array.length - 1; i++){
        minPos = i;
        for(int k = i + 1; k < array.length; k++){
             if(array[k].compareTo(array[minPos]) < 0)
                 minPos = k;
         }
         if(i!=minPos) {
             temp = array[minPos];
            array[minPos] = array[i];
            array[i] = temp;
    } }}
```

Selection Sort (cont'd)

- To sort an array with k elements, Selection sort requires k – 1 passes.
- Example:



Insertion Sort

- The array is partitioned into 2 parts, sorted (on the left) and unsorted (on the right). Initially the unsorted part has one element. In each pass, the first element of the unsorted part is inserted in the appropriate location in the sorted left block.
- The Insertion sort pseudo-code algorithm is:

```
insertionSort(array){
    for(i = 1; i < array.length; i++){   // array[i] is the element to be inserted
        temp = array[i];
        insert temp in its proper location in the sorted subarray array[0]...array[i -1]
}
```

- Inserting temp in its proper location involves two steps:
  - Finding the position k where temp is to be inserted.
  - Shifting each of the elements array[k]...array[i -1] to the right by one slot

```
//inserting to maintain ascending order
temp = array[i];
k = i;
while(k > 0 && array[k - 1] > temp){
   array[k] = array[k - 1];  // shift value at index k-1 to location with index k
   k--;
}
array[k] = temp; // insert array [i]  at the right location with index k
```

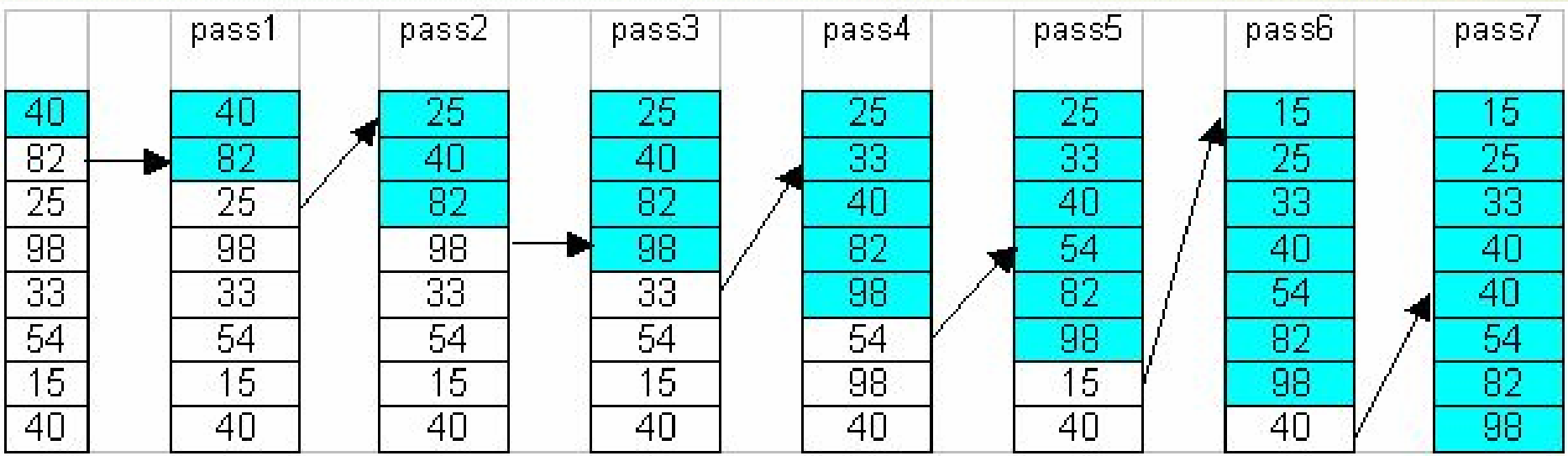
Insertion Sort (cont'd)

The following java method implements Insertion sort algorithm and sorts the array according to the comparator object.

```
public static void insertionSort(Object[] array,
   Comparator comp){
   int i,k;
   Object temp;
   for(i = 1; i < array.length  ; i++){
    temp = array[i];
    k = i;
    while((k > 0)&& comp.compare(array[k-1],temp) > 0){
       array[k] = array[k-1];
       k--;
    }
    array[k] = temp;
   }
}
```

Insertion Sort (cont'd)

- To sort an array with k elements, Insertion sort requires k – 1 passes.
- Example:



Bubble Sort

- The basic idea is to compare two **neighboring** objects, and to swap them if they are in the wrong order
- During each pass, the largest object is pushed to the end of the unsorted sub-array. This will continue until the unsorted sub-array has one element.

The Bubble (Exchange) sort pseudo-code algorithm is:

```
bubbleSort(array){
    numberOfPasses = 1;
    while(numberOfPasses < array.length){
        for(k = 1; k <= array.length – numberOfPasses; k++)
            swap array[k-1] and array[k] if they are out of order;
        numberOfPasses++;
}
```

- Note: If no swaps occur in a complete for loop, the array is sorted. A boolean variable may be used to terminate the while loop prematurely.

Bubble Sort (cont'd)

```
public static void bubbleSort(Object[] array,
  Comparator comp){
  int pass = 1;
  Object temp;
  boolean sorted;
  do{
     sorted = true;
     for(int m = 1; m <= array.length - pass; m++){
   if(comp.compare(array[m - 1], array[m]) > 0){
     temp = array[m-1]; // swap neighbors atm and m-1
     array[m-1] = array[m];
     array[m]   = temp;
     sorted = false;
   }} // end of for loop
      pass++;
  }while(! sorted);
  }
```

Bubble Sort (cont'd)

- To sort an array with k elements, Bubble sort requires k – 1 passes.
- Example:

	pass1	pass2	pass3	pass4	pass5	pass6	pass7
40	40	25	25	25	25	15	15
82	25	40	33	33	15	25	25
25	82	33	40	15	33	33	33
98	33	54	15	40	40	40	40
33	54	15	40	40	40	40	40
54	15	40	54	54	54	54	54
15	40	82	82	82	82	82	82
40	98	98	98	98	98	98	98

Exercises on Searching

1. What is a sequential search?
2. What is a binary search?
3. Write an iterative binarySearch method on an Object array.
4. Write a recursive binarySearch method on a double array.
5. Write a recursive linearSearch method on an Object array.
6. Write an iterative linearSearch method on a double array.
7. A binary search of an array requires that the elements be sorted in ascending order.
8. Suppose it is known that an array is sorted. When is linear search better than binary Search?
9. Mention the advantages, if any, of linear search over binary search.
10. Mention the advantages, if any, of binary search over linear search.
11. Mention the disadvantages, if any, of binary search over linear search.
12. Mention the disadvantages, if any, of linear search over binary search.
13. Design a Java program that will determine the average running times of binary search and linear search on sufficiently large integer arrays.
14. Each line of a text file contains the name of a person and his/her telephone number:

    firstName secondName telephoneNumber

    The names may not be unique and may also not be sorted. The telephone numbers are also not sorted. Write a Java telephone lookup program that handles lookups by name as well as by telephone number. Use binary search for both lookups.

Exercises on Searching (cont'd)

15. An integer array of size 100 stores contiguous integers. What is the minimum number of comparisons to determine if:
    (a) a value is in the array?
    (b) a value is not in the array?
16. The information of students taking two courses is maintained in two Object arrays, course1 and course2. By defining an appropriate Student class, write a Java program to determine the students who are:
    (a) taking both courses.
    (b) not taking both courses.
17. The element being searched for is not in an array of 100 elements. What is the maximum number of comparisons needed in a sequential search to determine that the element is not there if the elements are:
    (a) completely unsorted?
    (b) sorted in ascending order?
    (c) sorted in descending order?
18. The element being searched for is not in an array of 100 elements. What is the average number of comparisons needed in a sequential search to determine that the element is not there if the elements are:
    (a) completely unsorted?
    (b) sorted in ascending order?
    (c) sorted in descending order?

Exercises on Searching (cont'd)

19. Implement each of the following Java String search methods:

public boolean endsWith(String string)
public int indexOf(int char)
public int indexOf(int char, int fromIndex)
public int indexOf(String string)
public int indexOf(String s, int fromIndex)
public int lastIndexOf(int char)
public int lastIndexOf(int char, int fromIndex)
public boolean startsWith(String string)
public boolean startsWith(String s, int offset)
public int lastIndexOf(String s, int fromIndex)

20. Write a linear search method:

public static Object linearSearch(Object[] array, Object key)

that returns null if the search is not successful; otherwise it returns a reference to the first matching object.

21. Write a binary search method:

21 public static Object binarySearch(Object[] array, Object key)

that returns null if the search is not successful; otherwise it returns a reference to the a matching object. Assume that the elements of the array are Comparable.

Exercises on Searching (cont'd)

22. Consider the following array of sorted integers:

10, 15, 25, 30, 33, 34, 46, 55, 78, 84, 96, 99

Using binary search algorithm, search for 23. Show the sequence of array elements that are compared, and for each comparison, indicate the values of low and high.

Exercises on Sorting

1. Write a method:

   public static boolean isSorted(Object[] array, Comparator comparator)

   that returns true if array is sorted; otherwise it returns false.

2. Write a recursive bubble sort on a double array.

3. Write a recursive insertion sort on a double array.

4. Write a recursive selection sort on an Object array of Comparable objects.

5. Many operations can be performed faster on sorted than on unsorted data. For which of the following operations is this the case?

   (a) Finding an item with minimum value.

   (b) Computing an average of values.

   (c) Finding the middle value (the median).

   (d) Finding the value that appears most frequently in the data.

   (e) Finding the distinct elements of an array.

   (f) Finding a value closest to a given value.

   (g) Finding whether one word is an anagram (i.e., it contains the same letters) as another word.

   (example: plum and lump).

6. Rewrite some of the sorting methods we have studied such that each sorts an Object array of Comparable objects.

7. Show the contents of the following integer array:

   43, 7, 10, 23, 18, 4, 19, 5, 66, 14

   when the array is sorted in ascending order using:

   (a) bubble sort, (b) selection sort, (c) insertion sort.

Exercises on Sorting (cont'd)

8. Implement an insertion sort on an integer array that in each pass places both the minimum and maximum elements in their proper locations.

9. In our implementation of bubble sort, an array was scanned top-down to bubble down the largest element. What modifications are needed to make it work bottom-up to bubble up the smallest element?

10. A cocktail **shaker sort** is a modification of bubble sort in which the direction of bubbling changes in each iteration: In one iteration, the smallest element is bubbled up; in the next, the largest is bubbled down; in the next the second smallest is bubbled up; and so forth. Implement this algorithm.

11. Explain why insertion sort works well on partially sorted arrays.