ORACLE®

ACADEMY

Java Programming

2-6
Sorting and Searching

Overview

This lesson covers the following topics:

- Recognize the sort order of primitive types and objects
- Trace and write code to perform a simple Bubble Sort of integers
- Trace and write code to perform a Selection Sort of integers
- Trace and write code to perform a Binary Search of integers
- Compare and contrast search and sort algorithms
- Analyze the Big-O for various sort algorithms

Sorting Arrays

- There are many different algorithms for sorting arrays.
- Some are easier to code, and some have a faster computation time.
- When searching for elements in an array, it is beneficial to sort your array in lexicographical order to reduce time spent searching through the array.

An algorithm is a logical computational procedure that if correctly applied, ensures the solution to a problem.

Lexicographical order is an order based on the ASCII value of characters. The table of values can be found at http://www.asciitable.com/

Main Sorting Algorithms for Arrays

- There are many sorting algorithms for arrays.
- Some of the common algorithms are:
  - Selection sort
  - Bubble sort
  - Insertion sort
  - Merge sort
  - Quick sort

Selection Sort

- Selection sort is a simple sorting algorithm that works as follows:
  - Find the minimum value in the list.
  - Swap it with the value in the first position.
  - Repeat the steps above for the remainder of the list, starting at the second position and advancing each time.

Selection Sort

- This algorithm:
  - Is inefficient on large arrays because it has to iterate through the entire array each time to find the next smallest element.
  - Is preferred for smaller arrays because of its simplicity.

Steps to the Selection Sort Algorithm

- To sort the following array:

```
{5, 7, 2, 15, 3}
```

- Step 1: Find the smallest value, which is 2.
- Step 2: Swap the value with the first position:

```
{2, 7, 5, 15, 3}
```

- Repeat steps 1 and 2. 3 is the next smallest number.
- When swapped for the next position, the array is:

```
{2, 3, 5, 15, 7}
```

Steps to the Selection Sort Algorithm

- These steps are repeated until the array is sorted, or to be more specific, when array.length -1 is reached.
- At that point, if the second last element was bigger than the last, it would swap and the array would be sorted.
- Otherwise, the array is sorted.

```
{2, 3, 5, 7, 15}
```

- Watch this Selection Sort video:
  - https://www.youtube.com/watch?v=Ns4TPTC8whw

Selection Sort Code Example

- This example follows the rules of selection sort.

```
public class SelectionSortTester{
    public static void main(String[] args){
         int[] numbers = {40, 7, 59, 4, 1};
         int indexMin = 0; //the index of the smallest number
         for(int i = 0; i< numbers.length; i++){
              indexMin = i;
              for(int j = i + 1; j < numbers.length; j++){
                 if(numbers[j] < numbers[indexMin]){//if we find a smaller int, set it as the min
                    indexMin = j;
                  }
              } //we now have the index of the smallest int and can swap the values
              int temp = numbers[i];  //use temp to store the value
              numbers[i] = numbers[indexMin];
              numbers[indexMin] = temp;
         }
         for(int i = 0; i< numbers.length; i++){
              System.out.print(numbers[i] + " ");
         }
    }
}
```

Bubble Sort

- Bubble sort, also known as exchange sort, is another sorting algorithm that is simple but inefficient on large lists.
- The algorithm works as follows:
  - Compare 2 adjacent values (those at indexes 0 and 1).
  - If they are in the wrong order, swap them.
  - Continue this with the next two adjacent values (those at indexes 1 and 2) and on through the rest of the list.
  - Repeat steps 1 through 3 until array is sorted.

Bubble Sort Video

- Watch this Bubble Sort video:
  – http://www.youtube.com/watch?v=lyZQPjUT5B4

Bubble Sort Example

- To sort the following array:

```
{40, 7, 59, 4, 1}
```

- First, compare and swap the first two values:

```
{7, 40, 59, 4, 1}
```

- Then, compare and swap the next two values:

```
{7, 40, 59, 4, 1}
```

- Notice that the swap did not occur in the second comparison because they are already in correct order.
- Repeat these steps until the array is sorted.

```
{1, 4, 7, 40, 59}
```

Bubble Sort Array Example

- Step by step, the array would look as follows.

```
{40, 7, 59, 4, 1} //Starting array
{7, 40, 59, 4, 1} //7 and 40 swapped
{7, 40, 59, 4, 1} //40 and 59 did not swap
{7, 40, 4, 59, 1} //59 and 4 swapped
{7, 40, 4, 1, 59} //1 and 59 swapped
//First pass through array is complete
{7, 40, 4, 1, 59} //7 and 40 did not swap
{7, 4, 40, 1, 59} //40 and 4 swapped
{7, 4, 1, 40, 59} //40 and 1 swapped
{7, 4, 1, 40, 59} //40 and 59 did not swap
//Second pass is complete
{4, 7, 1, 40, 59} //7 and 4 swapped
{4, 1, 7, 40, 59} //7 and 1 swapped
{4, 1, 7, 40, 59} //7 and 40 did not swap
//Third pass is complete
{1, 4, 7, 40, 59} //1 and 4 swapped
{1, 4, 7, 40, 59} //4 and 7 did not swap
//Fourth pass is complete and the array is sorted
```

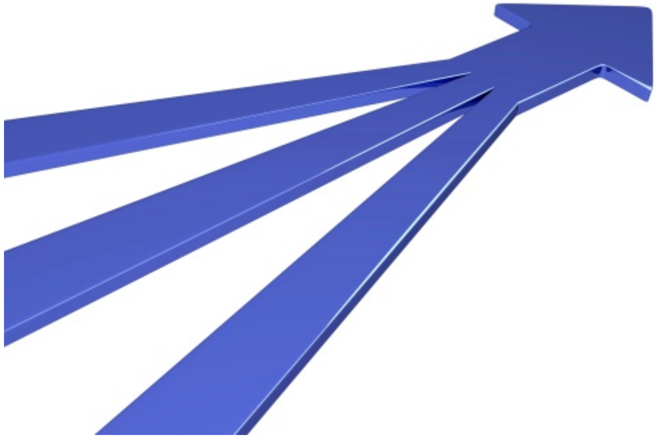
Bubble Sort Code Example

- Trace through this code on paper.
- Do you see all of the same steps as shown previously?

```
int[] numbers = {40, 7, 59, 4, 1};
for(int j =0 ; j < numbers.length; j++){
     for(int i = 0; i < numbers.length-1; i++){
          //if the numbers are not in order
         if(numbers[i] > numbers[i+1]){
               //swap the numbers
               int temp = numbers[i];
               numbers[i] = numbers[i+1];
               numbers[i+1] = temp;
         }
     }
}
```

Merge Sort

- Merge sort:
  – Is more complex than the previous two sorting algorithms.
  – Can be more efficient because it takes advantage of parallel processing.
  – Takes on a "divide and conquer" technique, which allows it to sort arrays with optimal speed.

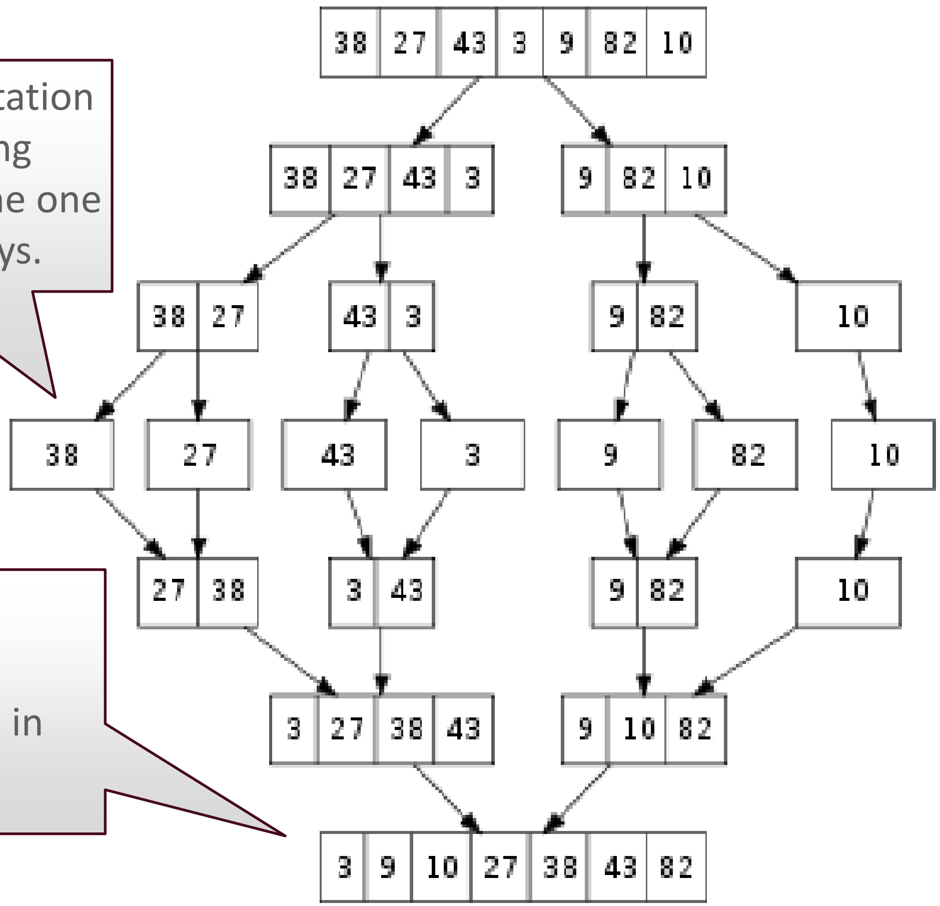


Merge Sort Algorithm

- The algorithm works as follows:
  - Divide the unsorted list into sub-lists, each containing one element (a list of one element is considered sorted).
  - Repeatedly merge sub-lists to produce new sub-lists until there is only one sub-list remaining.
  - This will be the sorted list.

Visual Representation of Merge Sort

This visual representation shows the array being broken down into the one element sorted arrays.



They are "merged" together again and again until they are in one sorted array.

Learn More About Merge Sort

- This lesson focuses on the theory of merge sort since it is so complex.
- Watch the following Merge Sort video:
  - https://www.youtube.com/watch?v=XaqR3G_NVoo

Searching Through Arrays

- Sorting an array makes searching faster and easier.
- There are two basic searching methods:
  - Sequential/Linear searches
  - Binary searches
- For example, in a sorted array of student names, you want to know if Juan is in your class.
- You could search the array, or find out exactly where his name is in alphabetical order with the other students.

Searching Through Arrays Sequentially

- A sequential search is an iteration through the array that stops at the index where the desired element is found.
- If the element is not found, the search stops after all elements of the list have been iterated through.
- This method works for unsorted or sorted arrays, but is not efficient on larger arrays.
- However, a sequential search is not more efficient if the list is sorted.
- Since we know that our array of student names is sorted, we could use a much more efficient searching method: A binary search.

Using Binary Search With Sorted Arrays

- Binary searches can only be performed on sorted data.
- To see how a binary search works:
  - Search for the target value 76 in the array.
  - Step 1: Identify the low, middle, and high elements in the array.
    - Low is 0, high is array.length = 11, middle is (high - low)/2 = 5
  - Step 2: Compare the target value with the middle value.
    - 76 is greater than 56.

Index	0	1	2	3	4	5	6	7	8	9	10
Value	2	10	16	34	37	56	57	76	81	83	85

Using Binary Search With Sorted Arrays

- To see how a binary search works:
  – Step 3: Since the target value is greater than the middle value, it is to the right of the middle. Set the low index to middle + 1, then calculate the new middle index.
    - Middle is ((5 +1) + 10)/2 = 8
  – Step 4: Compare the target value with the middle value.
    - 76 is less than 81.

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Value	2	10	16	34	37	56	57	76	81	83	85

Using Binary Search With Sorted Arrays

- To see how a binary search works:
  - Step 5: Since the target value is less than the middle value, it is to the left of the middle. Set the high index to middle - 1 then calculate the new middle index.
    - Middle is ((8 – 1) + 6) / 2 = 7
  - Step 6: Compare the target value with the middle value.
    - 76 is equal to 76.
  - Step 7: The target value has been found at index 7.

Index	0	1	2	3	4	5	6	7	8	9	10
Value	2	10	16	34	37	56	57	76	81	83	85

Decide What to Return in a Binary Search Method

- When writing a binary search method you will have to decide what to return.
- Index
- Boolean
- Value
  - Entire object
  - One field of the object
- What would you return if the value was not found?

Binary Search Example 1

- The example method will return true if the value is found in the array and false if it is not found.

```
public boolean binarySearch(int target, int[] data){
       int low = 0;
       int high = data.length - 1;

       while(high >= low){
          int middle = (low + high)/2; // Middle index

          if(data[middle] == target){
             return true;  // Target value was found
          }
          if(data[middle] < target){
             low = middle + 1;
          }
          if(data[middle] > target){
             high = middle – 1;
          }
        }
      return false; // The value was not found
}
```

Binary Search Example 2

- What would need to be changed to return the index of where the target value was found?

```
public boolean binarySearch(int target, int[] data){
       int low = 0;
       int high = data.length - 1;

       while(high >= low){
          int middle = (low + high)/2; // Middle index

          if(data[middle] == target){
             return true;  // Target value was found
          }
          if(data[middle] < target){
             low = middle + 1;
          }
          if(data[middle] > target){
             high = middle – 1;
          }
        }
      return false; // The value was not found
}
```

Comparison of Sorting Algorithms

- From https://en.wikipedia.org/wiki/Big_O_notation

Name	Best	Average	Worst	Memory	Stable	Method
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$	Depends	Partitioning
Merge sort	$n \log n$	$n \log n$	$n \log n$	Depends; worst case is n	Yes	Merging
In-place Merge sort	—	—	$n (\log n)^2$	1	Yes	Merging
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No	Selection
Insertion sort	n	n^2	n^2	1	Yes	Insertion
Introsort	$n \log n$	$n \log n$	$n \log n$	$\log n$	No	Partitioning & Selection
Selection sort	n^2	n^2	n^2	1	No	Selection
Timsort	n	$n \log n$	$n \log n$	n	Yes	Insertion & Merging
Shell sort	n	$n(\log n)^2$ or $n^{3/2}$	Depends on gap sequence; best known is $n(\log n)^2$	1	No	Insertion
Bubble sort	n	n^2	n^2	1	Yes	Exchanging
Binary tree sort	n	$n \log n$	$n \log n$	n	Yes	Insertion
Cycle sort	—	n^2	n^2	1	No	Insertion
Library sort	—	$n \log n$	n^2	n	Yes	Insertion
Patience sorting	—	—	$n \log n$	n	No	Insertion & Selection
Smoothsort	n	$n \log n$	$n \log n$	1	No	Selection
Strand sort	n	n^2	n^2	n	Yes	Selection
Tournament sort	—	$n \log n$	$n \log n$	n[5]		Selection
Cocktail sort	n	n^2	n^2	1	Yes	Exchanging
Comb sort	n	$n \log n$	n^2	1	No	Exchanging
Gnome sort	n	n^2	n^2	1	Yes	Exchanging
Bogosort	n	$n \cdot n!$	$n \cdot n! \to \infty$	1	No	Luck

Big-O Notation

- Big-O Notation is used in Computer Science to describe the performance of Sorts and Searches on arrays.
- The Big-O is a proportion of speed or memory usage to the size of the array.
- In the previous slide, Big-O examples are:
  – n
  – n log n
  – n2
- Compare the values of these sorts.
  – Which sort(s) is/are quickest when n is 100?
  – Which sort(s) is/are quickest when n is 1 Billion?

Terminology

Key terms used in this lesson included:

- Array
- Binary search
- Bubble Sort
- Lexicographical order
- Merge Sort
- Selection Sort
- Sequential search

Summary

In this lesson, you should have learned how to:

- Recognize the sort order of primitive types and objects
- Trace and write code to perform a simple Bubble Sort of integers
- Trace and write code to perform a Selection Sort of integers
- Trace and write code to perform a Binary Search of integers
- Compare and contrast search and sort algorithms
- Analyze the Big-O for various sort algorithms

ORACLE® ACADEMY