

16. Searching and Sorting

Java

Fall 2009

Instructor: Dr. Masoud Yaghini

Outline

- Searching Arrays
- Sorting Arrays
- Arrays Class
- References

Searching Arrays



Searching Arrays

- Searching is the process of looking for a specific element in an array
- There are many algorithms and data structures devoted to searching.
- In this section, two commonly used approaches are discussed:
 - **Linear search**
 - **Binary search**

Linear Search

- The linear search approach compares the key element, **key** sequentially with each element in the array.
- The method continues to do so until the key matches an element in the array or the array is exhausted without a match being found.
- If a match is made, the linear search returns the index of the element in the array that matches the key.
- If no match is found, the search returns -1.

Linear Search

- Java code:
 - [LinearSearch.java](#)
- Trace the method using the following statements:

```
int[] list = {1, 4, 4, 2, 5, -3, 6, 2};  
int i = linearSearch(list, 4);  
int j = linearSearch(list, -4);  
int k = linearSearch(list, -3);
```

Linear Search

- The result:

```
int[] list = {1, 4, 4, 2, 5, -3, 6, 2};  
int i = linearSearch(list, 4); // returns 1  
int j = linearSearch(list, -4); // returns -1  
int k = linearSearch(list, -3); // returns 5
```

- On average, the algorithm will have to compare half of the elements in an array before finding the key if it exists.
- Since the execution time of a linear search increases linearly as the number of array elements increases, linear search is inefficient for a large array.

Binary Search

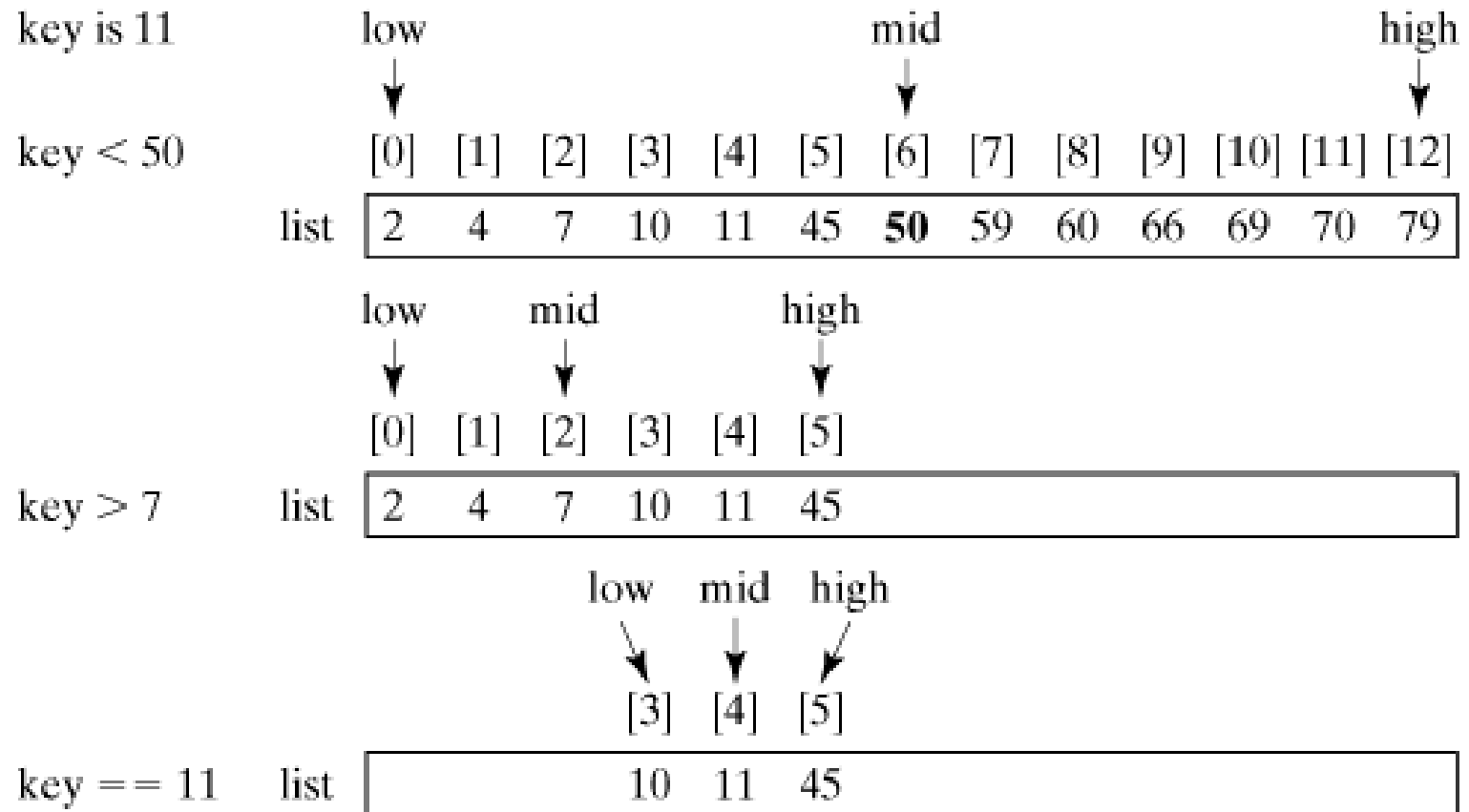
- For binary search to work, the elements in the array must already be ordered.
- Without loss of generality, assume that the array is in ascending order.

e.g., 2 4 7 10 11 45 50 59 60 66 69 70 79

- The binary search first compares the key with the element in the middle of the array.

Searching and Sorting

Binary Search Approach



Binary Search Approach

- The binary search returns the index of the search key if it is contained in the list.
- Otherwise, it returns $-(\text{insertion point} + 1)$.
- The insertion point is the point at which the key would be inserted into the list.
- For example,
 - the insertion point for key 5 is 2, so the binary search returns -3;
 - the insertion point for key 51 is 7, so the binary search returns -8.

Binary Search Approach

- The program:
 - [BinarySearch.java](#)
- Trace the program using the following statements:

```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66,  
             69, 70, 79};
```

```
int i = binarySearch(list, 2); // returns 0
```

```
int j = binarySearch(list, 11); // returns 4
```

```
int k = binarySearch(list, 12); // returns -6
```

Binary Search Approach

- Linear search is useful for finding an element in a small array or an unsorted array, but it is inefficient for large arrays.
- Binary search is more efficient, but requires that the array be pre-sorted.



Sorting Arrays



Sorting Arrays

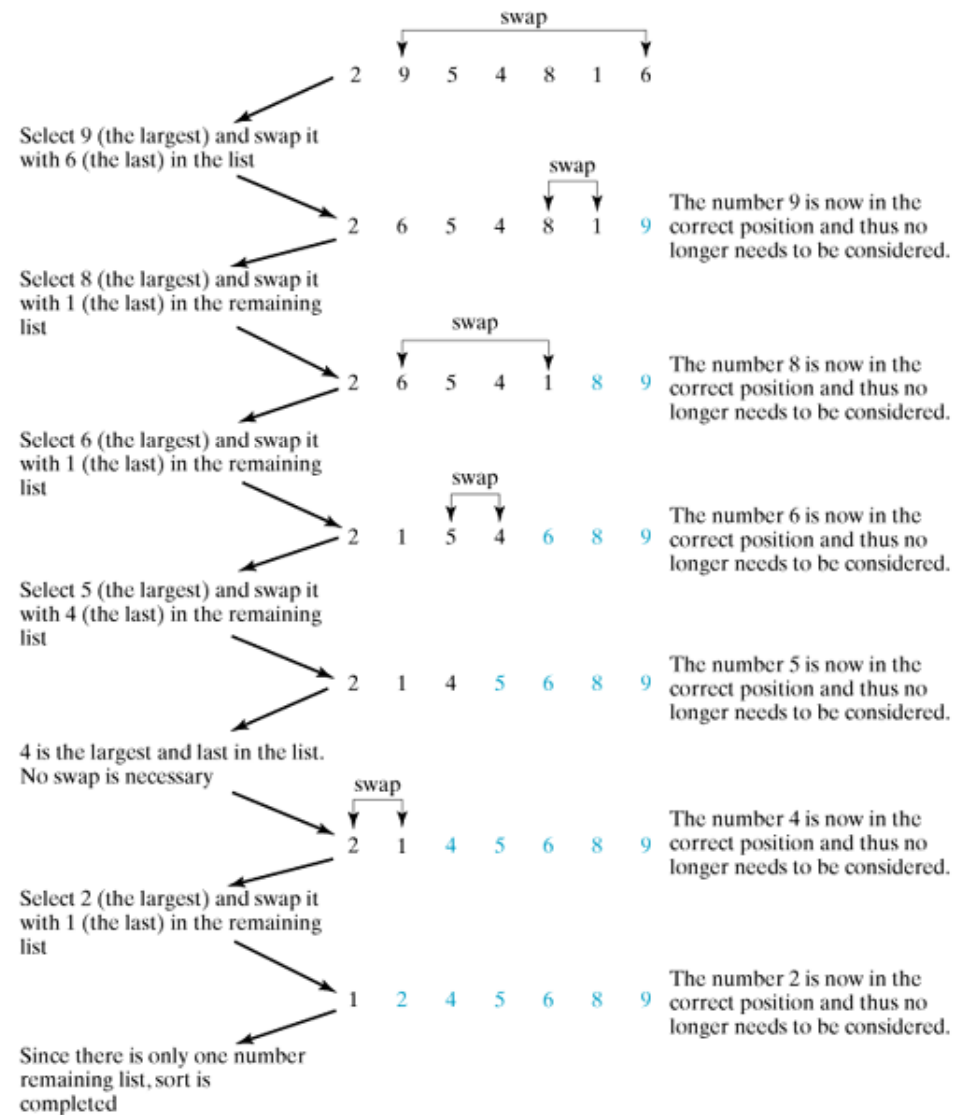
- Many different algorithms have been developed for sorting.
- This section introduces two simple sorting algorithms:
 - *Selection sort*
 - *Insertion sort.*

Selection Sort

- Selection sort finds the largest number in the list and places it last.
- It then finds the largest number remaining and places it next to last, and so on until the list contains only a single number.

Selection Sort

- The figure shows how to sort a list {2, 9, 5, 4, 8, 1, 6} using selection sort.
- The program:
 - [SelectionSort.java](#)



Insertion Sort

- The insertion-sort algorithm sorts a list of values by repeatedly inserting a new element into a sorted sublist until the whole list is sorted.
- The Figure shows how to sort a list {2, 9, 5, 4, 8, 1, 6} using insertion sort.

Searching and Sorting

Insertion Sort

Step 1: Initially, the sorted sublist contains the first element in the list. Insert 9 to the sublist.

2 9 5 4 8 1 6

Step 2: The sorted sublist is {2, 9}. Insert 5 to the sublist.

2 9 5 4 8 1 6

Step 3: The sorted sublist is {2, 5, 9}. Insert 4 to the sublist.

2 5 9 4 8 1 6

Step 4: The sorted sublist is {2, 4, 5, 9}. Insert 8 to the sublist.

2 4 5 9 8 1 6

Step 5: The sorted sublist is {2, 4, 5, 8, 9}. Insert 1 to the sublist.

2 4 5 8 9 1 6

Step 6: The sorted sublist is {1, 2, 4, 5, 8, 9}. Insert 6 to the sublist.

1 2 4 5 8 9 6

Step 7: The entire list is now sorted

1 2 4 5 6 8 9

Insertion Sort

- **How to Insert?**

list [0] [1] [2] [3] [4] [5] [6]
2 5 9 4

Step 1: Save 4 to a temporary variable `currentElement`

list [0] [1] [2] [3] [4] [5] [6]
2 5 9

Step 2: Move `list[2]` to `list[3]`

list [0] [1] [2] [3] [4] [5] [6]
2 5 9

Step 3: Move `list[1]` to `list[2]`

list [0] [1] [2] [3] [4] [5] [6]
2 4 5 9

Step 4: Assign `currentElement` to `list[1]`

- **The program:**
 - [InsertionSort.java](#)



Arrays Class



Arrays Class

- The `java.util.Arrays` class contains various static methods for sorting and searching arrays, comparing arrays, and filling array elements.
- These methods are overloaded for all primitive types.

The sort Method

- You can use the `sort` method to sort a whole array or a partial array.

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};  
java.util.Arrays.sort(numbers);  
// Sort the whole array
```

```
char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};  
java.util.Arrays.sort(chars, 1, 3);  
// Sort part of the array
```

The `binarySearch` Method

- You can use the `binarySearch` method to search for a key in an array.
- The array must be pre-sorted in increasing order. If the key is not in the array, the method returns $-(\text{insertion point} + 1)$.

The binarySearch Method

```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66,  
             69, 70, 79};
```

```
System.out.println("(1) Index is " +  
    java.util.Arrays.binarySearch(list, 11));
```

```
System.out.println("(2) Index is " +  
    java.util.Arrays.binarySearch(list, 12));
```

```
char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};
```

```
System.out.println("(3) Index is " +  
    java.util.Arrays.binarySearch(chars, 'a'));
```

```
System.out.println("(4) Index is " +  
    java.util.Arrays.binarySearch(chars, 't'));
```

- Output?

The binarySearch Method

- The output of the code is
 - (1) Index is 4
 - (2) Index is -6
 - (3) Index is 0
 - (4) Index is -4

The equals method

- You can use the `equals` method to check whether two arrays are equal. Two arrays are equal if they have the same contents.

```
int[] list1 = {2, 4, 7, 10};
```

```
int[] list2 = {2, 4, 7, 10};
```

```
int[] list3 = {4, 2, 7, 10};
```

```
System.out.println(java.util.Arrays.equals  
    (list1, list2)); // true
```

```
System.out.println(java.util.Arrays.equals  
    (list2, list3)); // false
```

The fill method

- You can use the `fill` method to fill in the whole array or part of the array.

```
int[] list1 = {2, 4, 7, 10};
```

```
int[] list2 = {2, 4, 7, 10};
```

```
// fill 5 to the whole array
```

```
java.util.Arrays.fill(list1, 5);
```

```
// fill 8 to a partial array
```

```
java.util.Arrays.fill(list2, 1, 3, 8);
```

Array Class

- You can find all methods from the following URL:

<http://java.sun.com/javase/6/docs/api/>



References



References

- Y. Daniel Liang, **Introduction to Java Programming**, Sixth Edition, Pearson Education, 2007. (Chapter 6)
- S. Zakhour and et. al., **The Java Tutorial: A Short Course on the Basics**, 4th Edition, Prentice Hall, 2006. (Chapter 3)



The End