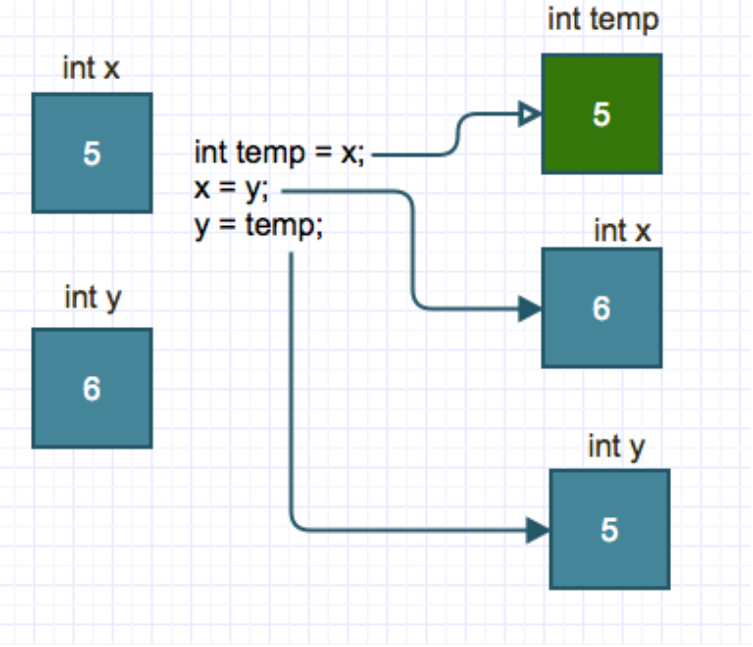
Swapping, partitioning, and sorting algorithms in Java

Posted on October 18, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Q1. Can you write an algorithm in Java to **swap two variables**?

A1.



Swapping variables algorithm

```
1  public class Swap {
2
3      public static void main(String[ ] args) {
4          int x = 5;
5          int y = 6;
6
7          //store 'x' in a temp variable
8          int temp = x;
9          x = y;
10         y = temp;
11
12         System.out.println("x=" + x + ",y=" + y)
13     }
14 }
15
16
```

Q2. Can you write an algorithm to **bubble sort** the following array { 30, 12, 18, 0, -5, 72, 424 }??

A2.

```
1  import java.util.Arrays;
2
3  public class BubbleSort {
4
5      public static void main(String[ ] args) {
6          Integer[ ] values = { 30, 12, 18, 0, -5,
7          int size = values.length;
8          System.out.println("Before:" + Arrays.de
9
10         for (int pass = 0; pass < size - 1; pass
```


16 Technical Key Areas

```
11              for (int i = 0; i < size - pass - 1;
12                  // swap if i > i+1
13                  if (values[i] > values[i + 1])
14                      swap(values, i, i + 1);
15              }
16          }
17
18          System.out.println("After:" + Arrays.dee
19      }
20
21      private static void swap(Integer[ ] array, i
22          int temp = array[i];
23          array[i] = array[j];
24          array[j] = temp;
25      }
26  }
```

Q3. Is there a more efficient sorting algorithm than a bubble sort algorithm?

A3. Although **bubble-sort** is one of the simplest sorting algorithms, it's also one of the slowest. It has the O(n2) time complexity. Faster algorithms include **quick-sort** and **heap-sort**. The A*rrays.sort( )* method uses the quick-sort algorithm, which on average has O(n * log n) but can go up to O(n2) in a worst case scenario, and this happens especially with already sorted sequences.

```
1  import java.util.Arrays;
2
3  public class QuickSort {
4
5      public static void main(String[ ] args) {
6          Integer[ ] values = { 30, 12, 18, 0, -5,
7          System.out.println("Before:" + Arrays.de
8          Arrays.sort(values);
9          System.out.println("After:" + Arrays.dee
10     }
11 }
```

Q4. How will you **partition** the following numbers 7 3 6 8 2 9 5 4 around the pivotal number of 5 so that the lower values are in front of it and higher values are behind it?

A4. **Approach 1:** With two pointers as shown below

7	3	6	8	2	9	5 (pivot)	4

Start two pointers, one at array index 0 and the other at array index 7, which is the last element in the array. Advance the left pointer forward until a value greater than the pivot (i.e. 5) is encountered. Advance the right pointer backwards until a value less than the pivot (i.e 5) is encountered. In this case, the current left pointer value 7 is greater than pivot (i.e. 5) and the right pointer value 4 is less than the pivot (i.e. 5), so swap them as shown below before advancing as shown below with the shading. The pointers have also advanced to the next position. Also note that the left pointer has skipped the value "3" as it is less than pivot value"5". The right pointer has skipped the values "5" and "9" as they are equal and greater than the pivot value of "5" respectively.

4	3	6	8	2	9	5 (pivot)	7

The left pointer value 6 is greater than pivot (i.e. 5) and the right pointer value 2 is less than the pivot (i.e. 5), so swap them as shown below before advancing.

4	3	2	8	6	9	5 (pivot)	7

When the pointers are pointing to the same value as shown above, swap it with the pivot value as shown below.

4	3	2	5 (pivot)	6	9	8	7

As you can see now, the smaller values (or numbers) are in front of the pivot value "5" and higher values are behind it. This array is now called **partitioned**. The partitioning is used recursively by the quick sorting algorithm to sort elements. The above partitioning algorithm can be applied as shown below.

Partitioning numbers with two pointers

```
1  public class Partition {
2
3      public static int partition(int[ ] values, i
4              int rightIndex, int pivotIndex) {
5
6          int i = leftIndex;
7          int j = rightIndex;
8
9          while (i < j) {
10             //moving from left to right
11             while (i <= j && values[i] <= values
12                 i++;
13             //moving from right to left
14             while (j >= i && values[j] >= values
15                 j--;
16
17             if (i < j)
18                 swap(values, i, j);
19         }
20
21         //swap the current index value with the
22         swap(values, i, pivotIndex);
23         return i;                       // return
24     }
25
26     public static void swap(int[ ] values, int i
27         int temp = values[i];
28         values[i] = values[j];
29         values[j] = temp;
```

```
30      }
31
32      public static void main(String[ ] args) {
33          int[ ] values = { 7, 3, 6, 8, 2, 9, 5, 4
34          //pivot value is 5, left index value is
35          partition(values, 0, values.length -1, v
36
37          //prints 4,3,2,5,6,9,8,7
38          for ( int i = 0; i < values.length; i++
39              System.out.print(values[i]);
40              if(i < values.length-1)
41                  System.out.print(",");
42          }
43      }
44  }
```

Approach 2: With a single pointer as shown below:

7	3	6	8	2	9	5 (pivot)	4

Send the pivot item to the back.

7	**3**	6	8	2	9	4	5 (pivot)
^							

Move from left to right, and swap with the pointer if the current value is less than the pivot value "5". 3 is less than the pivot value of 5, so swap it with the pointer value. After swapping, move the pointer forward one index.

3	7	6	8	**2**	9	4	5 (pivot)
	^						

Skipping values "6" and "8" as they are greater than the pivot value of "5". 2 is less than the pivot value, so swap it with the current pointer value. After swapping, move the pointer forward one index.

3	2	6	8	7	9	**4**	5 (pivot)
		^					

4 is less than the pivot value, so swap it with the current pointer value and move the pointer forward one index.

3	2	4	8	7	9	6	5 (pivot)
			^				

No more smaller values than the pivot value of "5" can be found, so swap the pivot value with the current pointer value.

3	2	4	5 (pivot)	7	9	6	8
			^				

Partitioning numbers with a single pointer

```
1  public class Partition2 {
2
3      public static int partition(int[ ] values, i
4                                              int rig
5
6          int pivotValue = values[pivotIndex];
7          // send pivot item to the back
8          swap(values, pivotIndex, rightIndex);
9
10         // keep track of where the front is
```

```
11          int index = leftIndex;
12          // check from the front to the back
13          for (int i = leftIndex; i < rightIndex;
14          {
15              // swap if the current value is less
16              if (values[i] < pivotValue) {
17                  swap(values, i, index);
18                  index++;
19              }
20          }
21
22          // put pivot item in the middle
23          swap(values, rightIndex, index);
24          return index;              // return the c
25      }
26
27      public static void swap(int[ ] values, int i
28          int temp = values[i];
29          values[i] = values[j];
30          values[j] = temp;
31      }
32
33      public static void main(String[ ] args) {
34          int[ ] values ={ 7, 3, 6, 8, 2, 9, 5, 4}
35          partition(values, 0, values.length - 1,
36
37          //prints 3,2,4,5,7,9,6,8
38          for (int i = 0; i < values.length; i++)
39              System.out.print(values[i]);
40              if(i < values.length-1)
41                 System.out.print(",");
42          }
43      }
44  }
```

Q5. How would yo go about applying the **QuickSort algorithm** to sort the values 7 3 6 8 2 9 5 4 without using the Arrays.sort(…)?

A5. The key steps involved are:

- **Partition** the array with respect to a random element.
- **Partition** the left part of the array.
- **Partition** the right part of the array.
- **Recursively perform the above steps** until the exit condition is reached.

Since this is a recursive algorithm, you need a terminating or exit condition that does not make a recursive call. The terminating condition is when the rightIndex is less than or equal to the leftIndex. If you don't know recursion, search this training for "**recursion**".

```
1  public class QuickSort {
```

```
2
3       //... partition and swap methods as shown be
4
5       public static int[ ] quicksort(int[ ] values
6
7           // This is a recursive algorithm and the
8           if (rightIndex > leftIndex) {
9               // partition the array with respect
10              int pivotIndex = (leftIndex + rightI
11              int newPivotIndex = partition(values
12
13              // partition the left part of the ar
14              quicksort(values, leftIndex, newPivo
15
16             // patition the right part of the arr
17              quicksort(values, newPivotIndex + 1,
18          }
19          return (int[ ]) values;
20      }
21
22      public static void main(String[ ] args) {
23          int[ ] values = { 7, 3, 6, 8, 2, 9, 5, 4
24          quicksort(values, 0, values.length - 1);
25
26          // prints 2,3,4,5,6,7,8,9
27          for (int i = 0; i < values.length; i++)
28              System.out.print(values[i]);
29              if (i < values.length - 1)
30                  System.out.print(",");
31          }
32      }
33  }
```

Q. Did you make any assumptions in the above code?
A. Yes, the supplied array is not already sorted, it is a large array, and the sorting is always in ascending order.

Q6. Can you write a function that checks if a given array is already sorted either in ascending or descending order?
A6.

```
1   public class AlgorithmUtils {
2
3       public static boolean sorted(int[ ] input) {
4
5           if (input == null || input.length == 0)
6               throw new IllegalArgumentException("
7           }
8
9           boolean ascending = false;
10          int first = input[0];
11          int second = input[1];
12          // ascending
13          if (first < second)
14              ascending = true;
15
16          for (int i = 0; i < input.length-1; i++)
17              if ((ascending && input[i] > input[i
18                      || (!ascending && input[i] <
```

```
19                  return false;
20          }
21
22          return true;
23      }
24
25      //...
26  }
```

Q. How will you ensure that the above code is functioning it correctly?
A. By writing unit tests that cover positive scenarios like an array sorted in ascending order and an array sorted in descending order and also a negative scenario like an unsorted array. Also, test for exceptional conditions like a null or empty array.

Popular Posts

♦ 11 Spring boot interview questions & answers

823 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

765 views

18 Java scenarios based interview Questions and Answers

399 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

388 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

295 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

285 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

239 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ 3 ways to get some experience on your Java CV

Searching algorithms in Java ›

Posted in Algorithms, member-paid

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category

© Disclaimer

The contents in this Java-Success are copy righted. The author has the right to correct or enhance the current content without any prior notice.

These are general advice only, and one needs to take his/her own circumstances into consideration. The author will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. No guarantees are made regarding the accuracy or usefulness of content, though I do make an effort to be accurate. Links to external sites do not imply endorsement of the linked-to sites.

↑