1.
big o
http://www.corejavainterviewquestions.com/idiots-guide-big-o/

2. collections
http://www.javaweb.cc/language/java/182388.shtml

3, arrays.equals
http://blog.csdn.net/forwayfarer/article/details/2147431

http://stackoverflow.com/questions/8777257/equals-vs-arrays-equals-in-java

Arrays inherit `equals()` from `Object` and hence compare only returns true if comparing an array against itself.

On the other hand, `Arrays.equals` compares the elements of the arrays.

This snippet elucidates the difference:

```
Object o1 = new Object(); Object o2 = new Object(); Object[] a1 = { o1, o2 };
Object[] a2 = { o1, o2 }; System.out.println(a1.equals(a2)); // prints false
System.out.println(Arrays.equals(a1, a2)); // prints true
```

See also `Arrays.equals()`. Another static method there may also be of interest: `Arrays.deepEquals()`.

The `Arrays.equals(array1, array2)` :

check if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal.

The `array1.equals(array2)` :

compare the object to another object and return true only if the reference of the two object are equal as in the `Object.equals()`

remove element from an array

https://www.cs.umd.edu/~clin/MoreJava/Container/arr-remove.html

1. **Removing An Element Using Loops**
2. **Removing Without Order-Preservation**

shuffle an array

http://www.programcreek.com/2015/03/rotate-array-in-java/

java arrayutils doc

https://commons.apache.org/proper/commons-lang/javadocs/api-2.6/org/apache/commons/lang/ArrayUtils.html#removeElement(boolean[], boolean)

4. arraylist

implement an arraylist

http://www.java2novice.com/java-interview-programs/arraylist-implementation/

> The answer is quite simple. The ArrayList used to implement the doubling-up policy. (In Java 6, there has been a change to be (oldCapacity*3)/2 +1). Here is the code snippet of how it is done in Java.

amortized analysis

https://en.wikipedia.org/wiki/Amortized_analysis

vector vs arraylist.

http://beginnersbook.com/2013/12/difference-between-arraylist-and-vector-in-java/

5. linkedlist

synchronized

http://stackoverflow.com/questions/9468187/collections-synchronizedlist-and-synchronized

6. stack

Min Stack

7. queue

java blocking queue

http://codereview.stackexchange.com/questions/7002/java-blocking-queue

bfs

1. https://leetcode.com/submissions/detail/32563204/
2. https://leetcode.com/problems/word-ladder/

8. 排序算法（1）
http://blog.csdn.net/hguisu/article/details/7776068
http://gengning938.blog.163.com/blog/static/128225381201141121326346/

稳定性：

通俗地讲就是能保证排序前2个相等的数其在序列的前后位置顺序和排序后它们两个的前后位置顺序相同。在简单形式化一下，如果Ai = Aj, Ai原来在位置前，排序后Ai还是要在Aj位置前。

非比较排序算法：

count sort

http://m.blog.csdn.net/blog/cqs_2012/18238621

radix sort
http://www.cnblogs.com/sun/archive/2008/06/26/1230095.html

bucket sort
http://blog.csdn.net/caspiansea/article/details/8606324

sort color leetcode

https://leetcode.com/problems/sort-colors/

```
public class Solution {
    public void sortColors(int[] A) {
        int []count = new int[3];
        for(int i = 0; i < A.length; i++) {
            count[A[i]] ++;
        }
        int p = 0;
        for(int i = 0; i < 3; i++) {
            for(int j = 0; j < count[i]; j++) {
                A[p++] = i;
            }
        }
    }
}
```

9. comparable vs comparator

http://www.programcreek.com/2011/12/examples-to-demonstrate-comparable-vs-comparator-in-java/

example :

1.**Merge Intervals**

2.https://leetcode.com/problems/merge-k-sorted-lists/

3. top k 问题

equals vs hashcode

http://www.programcreek.com/2011/07/java-equals-and-hashcode-contract/

http://javarevisited.blogspot.com/2013/08/10-equals-and-hashcode-interview.html

object class method

https://docs.oracle.com/javase/tutorial/java/IandI/objectclass.html

10. recursion

draw recursion tree

Lowest Common Ancestor of a Binary Tree

fibonacci一sequence

http://codercareer.blogspot.com/2011/10/no-15-fibonacci-sequences.html

11. hashing

1. linear probing
2. separate chaining

哈希函数

http://blog.csdn.net/eaglex/article/details/6310727

hash算法原理及常见函数

http://blog.csdn.net/pngynghay/article/details/22433715

12.hashtable

1.实现
http://blog.csdn.net/eaglex/article/details/6305997

2. 线程安全

http://blog.csdn.net/u011345136/article/details/46793981

http://topmanopensource.iteye.com/blog/1739110

http://javahungry.blogspot.com/2014/02/hashmap-vs-concurrenthashmap-java-collections-interview-question.html

http://javahungry.blogspot.com/2014/03/hashmap-vs-hashtable-difference-with-example-java-interview-questions.html

13. hashset vs hashmap

14. mergesort vs quicksort

collections.sort()

iterative mergesort that requires far fewer than n lg(n) comparisons

Arrays.sort()

The sorting algorithm is a Dual-Pivot Quicksort

15. binary tree
删除节点

http://blog.csdn.net/fightforyourdream/article/details/16843303

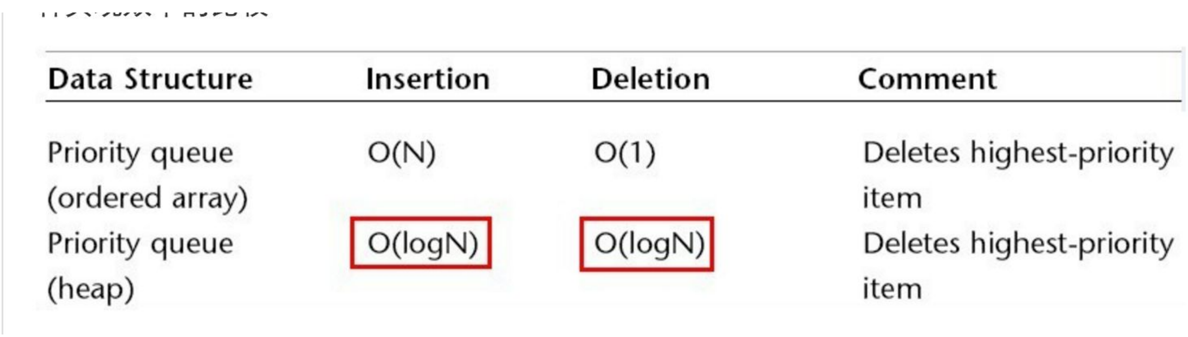
16. huffman code

http://www.cnblogs.com/mcgrady/p/3329825.html

17. Treeset

TreeSet is implemented using a tree structure(red-black tree in algorithm book).

18. Heap



Data Structure	Insertion	Deletion	Comment
Priority queue (ordered array)	O(N)	O(1)	Deletes highest-priority item
Priority queue (heap)	O(logN)	O(logN)	Deletes highest-priority item

http://blog.csdn.net/yangzhongblog/article/details/8607632

java priorityqueue

PriorityQueue是非线程安全的，所以Java提供了PriorityBlockingQueue（实现BlockingQueue接口）用于Java多线程环境。

http://www.cnblogs.com/gnivor/p/4841191.html

top K问题是这样的，给定一组任意顺序的数，假设有n个。如何尽快地找到它们的前K个最大的数？

首先，既然是找前K个最大的数，那么最直观的办法是，n个数全部都排序，然后挑出前K个最大数。但是这样显然做了一些不必要的事儿。

主要步骤如下：

step 1. 选出前K个数，挑出这K个数的最小的数。这个过程可以用最小堆完成。

step 2. 在剩下的n – K个数中，挑出任意一个数m，和最小堆的堆顶进行比较，如果比最小堆的堆顶大，那么说明此数可以入围前K的队伍，于是将最小堆的堆顶置为当前的数m。

step 3. 调整最小堆。时间复杂度为*Olg(K)*，由于K是constant(常数级别)，所以时间复杂度可以认为是常数级别。

step 4. 重复进行step 2 ~ step 3，直到剩下的n – K个数完成。进行了n –constant次，时间复杂度为*O(n lgK).*

heapsort

堆积排序(Heapsort)是指利用堆积树（堆）这种资料结构所设计的一种排序算法，可以利用数组的特点快速定位指定索引的元素。堆排序是不稳定的排序方法，辅助空间为O(1)，最坏时间复杂度为O(nlog2n)，堆排序的堆序的平均性能较接近于最坏性能。