Java Programming Interview Crash Course

Michael Theodorides

November 16, 2016

Abstract

This is a quick crash course refresher for Java Programming and Algorithms Interviews.

1 Java Primer

1.1 Collections

Key Methods: clear()

1.2 List

Key Methods:

- sublist(int fromIndex, int toIndex) : Returns a view of the list from fromIndex (inclusive) to toIndex (exclusive)

1.3 Comparable Interface

```
public interface Comparable<T> {
   int compareTo(T other); //Returns Negative if other is greater,
      0 if equal, Positive if other is less
}
```

1.4 Comparator Interface

```
public interface Comparator<T> {
   int compare(T first, T second); //Returns Negative if second is
       greater, 0 if equal, Positive if second is less
}
```

1.5 Sorting

- Arrays.sort(a), Collections.sort(a) //a is comparable
- Arrays.sort(a, cmp), Collections.sort(a, cmp) //cmp is a comparator

1.6 Priority Queue (Heap)

- Comparable Constructor: new PriorityQueue¡¿();
- Comparator Constructor: new PriorityQueue¡¿(int initialSize, Comparator cmp);
- Main Operations: add(elem), peek(), poll() //removes min

Integer.compare(int a, int b);

2 Heaps (Priority Queues)

Heap Applications: Use a heap when (1.) all you care about is the largest or smallest elements, and you do not need to support fast lookup, delete, or search operations for arbitrary elements (2.) you need to compute the k largest or k smallest elements in a collection.

Heap Property: The key at each node is at least as great (higher priority) as the keys stored at its children.

Heap Operations:

- Insertion: O(log n)
- Lookup Max (Highest Priority): O(1)
- Delete Max (Highest Priority): O(log n)
- Arbitrary Search: O(n)

3 Sorting

Java Review:

- Sorting Comparable Collections and Arrays: Collections.sort(e), Arrays.sort(a)
- Comparable Interface
- Comparator Interface

Sorting Algorithm Criteria:

- Amount of Data Stored
- Available Memory (Does data fit into memory?)
- Partially Sorted Data
- In-Place Algorithm: Sorts data without any additional memory
- Stable Algorithm: Preserves the relative order of data elements that are otherwise identical for sorting purposes.

Comparision Algorithms: Sorting algorithms that only require that there is a way to determine whether one key is less than, equal to, or greater than another key.

- No comparison algorithm can have a more optimal worst-case running time than O(n log(n)).

Selection Sort: Algorithm: Start with the first element in the array and scan through the array to find the element with the smallest key to swap with the first element. Repeat the process with each subsequent element until the last element is reached.

- Running-Time: $O(n^2)$(Best, Average, Worst-Case)
- Advantage: Requires at most n-1 swaps (good when moving elements is more expensive than comparing them); In-Place; Maybe Stable

Insertion Sort: Algorithm: Build a sorted array one element at a time by comparing each new element to the already-sorted elements and inserting the new element into the correct location.

- Running-Time: $O(n^2)$(Average, Worst-Case), $O(n)$ (Best-Case)
- Advantage: Efficient way to add new elements to a presorted list; Stable, In-Place, Suitable for Small Sets

Quicksort: Algorithm: Select a pivot value from the data set and split the set into two subsets: a set that contains all values less than the pivot and a set that contains all values greater than or equal to the pivot. The pivot/split process is recursively applied to each subset until there are no more subsets to split.

- Running-Time: $O(n^2)$(Worst-Case), $O(nlog(n))$ (Best-Case, Average-Case)
- For randomly ordered data, the value of the pivot is unrelated to its location, therefore a pivot can be chosen from any location.
- Performance is dependent on the choice of pivot value.
- Maybe Stable, Maybe In-Place

Merge Sort: Algorithm: Split the data set into two or more subsets, sorting the subsets, and then merge them together into the final sorted set.

- Running-Time:$O(nlog(n))$ (Best-Case, Average-Case, Worst-Case)
- Good choice for data sets that are too large to fit into memory.

- O(n) Additional Memory required by Merge Step
- Stable, Not In-Place

Loop Join Intersection: Loop through the smaller list and binary search through the second list (if second list is sorted).

Concurrent Intervals Problem: Sort by Interval Start/End Times; Increment Counter on Start; Decrement Counter on End (Also: Peak Bandwidth Problem)