L14: Sorting
Alex Steiger
CompSci 201: Spring 2024
2/28/2024: LEAP DAY EVE

Announcements, Coming up

• Today, Wednesday 2/28
  • APT 5 (linked list problems) due

• Next Monday 3/4
  • Project P3: DNA (linked list project) due

• Next Wednesday 3/6
  • APT 6 (sorting problems) due

• Then...Spring Break!

Today’s outline

1. Announce Midsemester Survey
   1. Invaluable for staff, especially UTAs ⇒ for you!
   2. Look for Canvas announcement from Violet

2. Sorting in Java: Comparing objects with Comparable and Comparator

3. Efficient sorting algorithms
   1. Insertion sort
   2. Recursive Mergesort
Sorting in Java: Comparable, Comparator

Sorting w/ Java.util: Put elements of Array/List in non-decreasing order

• Arrays.sort / Collections.sort are void – they sort the array/list passed as an argument.
• Default order is non-decreasing (least to greatest).

```java
int[] elements = {5, 3, 9, 2, 4, 1};
Arrays.sort(elements);
System.out.println(Arrays.toString(elements));
```

• Prints [1, 2, 3, 4, 5, 9]

Java API Sort Algorithms

• Collections.sort (for a List)
• Arrays.sort (for an Array)

• Both O(N log(N)), nearly linear runtime complexity.
• Sorts in-place, mutates the input rather than return a new List/Array.
• Stable, does not reorder elements if not needed (e.g., if two elements are equal).
What can be compared and sorted in Java?

- Objects of a class that implements Comparable interface. Has a naturalOrder.
- Requires implementing a `compareTo()` method.

Should return an int:

- `< 0 if this comes before the parameter.
- `0 if this and the parameter are equal.
- `> 0 if this comes after the parameter.

Strings are Comparable

- What is the equivalent of `<` for Strings?
- Use the `compareTo` method for the natural lexicographic (dictionary/sorted) ordering.

```
jshell> "a".compareTo("b");
$30 == -1  // Negative for "less than"

jshell> "b".compareTo("a");
$32 == 1   // Positive for "less than"
```

Lexicographic, check first character, second if equal, third if still equal

Sorting Comparable objects by naturalOrder

```
String[] a = {"sloth", "house", "owl", "ant", "mice", "kelp"};
System.out.println(Arrays.toString(a));

String[] copy = Arrays.copyOf(a, a.length);
Arrays.sort(copy);
System.out.println(Arrays.toString(copy));
```

```
[ant, house, kelp, mice, owl, sloth]
```

- naturalOrder for Strings is lexicographic (alphabetical or dictionary order)
Comparable for other classes?

- Can implement `Comparable` interface when defining your own class.

```java
public class Blob implements Comparable<Blob> {
  String name;
  String color;
  int size;

  public int compareTo(Blob other) {
    return this.name.compareTo(other.name);
  }
}
```

- Must implement a `compareTo` method.

```
@override
public int compareToBlob(Blob other) {
  return this.name.compareTo(other.name);
}
```

Compares blobs by their names.

Sorting Comparable Objects

- Running code in a main method...

```
System.out.println(myBlobs);
Collections.sort(myBlobs);
System.out.println(myBlobs);
```

Original: ([bo, blue, 4], [al, red, 2], [cj, green, 1], [di, red, 4])

Sorted: ([al, red, 2], [bo, blue, 4], [cj, green, 1], [di, red, 4])

- Formal guarantee: Element `e1` will come before `e2` (after sorting) if `e1.compareTo(e2) < 0`.

Defining a Comparator

- What if...
  - The class doesn't implement Comparable?
  - Or you want to sort a different way?

  - Create a helper class that implements the `Comparator` interface.
    - One method: `compare` indicates how to compare two objects

  - Then pass a Comparator object to your call to sort.
Defining a Comparator<Blob>

1. `import java.util.Comparator;`

Separate class:
- implements Comparator<TypeToCompare>,
- and implements a single method `compare`

```
8 public class BlobComparator implements Comparator<Blob> {
9     @Override
10     public int compare(Blob a, Blob b) {
11         int sizeDiff = a.size - b.size;
12         if (sizeDiff == 0) {
13             return (-1) * a.size;
14         }
15         return a.compareTo(b);
16     }
17 }
```

Sorting with a Comparator

- Running code in a main method...

```
40     System.out.println(myBlobs);
```

Original: [(bo, blue, 4), (al, red, 2), (cj, green, 1), (di, red, 4)]
```
48     Collections.sort(myBlobs, new BlobComparator());
49     System.out.printf(format, "Sorted", myBlobs);
```

Sorted: [(bo, blue, 4), (di, red, 4), (al, red, 2), (cj, green, 1)]

- Element e1 will come before e2 (after sorting) if `compare(e1, e2) < 0`.

Private Inner Comparator

- Can define a Comparator class as a private inner class if only used inside the class.
- Useful for APTs, here is an example:
Private Inner Comparator

- Can define a Comparator class as a private inner class if only used inside the class.
- Useful for APTs, here is an example:

  **Given String[] values:**
  - Sort first in non-decreasing order of length.
  - then sort same-length in alphabetical order.

  ```java
  ['a', 'b', 'c', 'an', 'be', 'pi', 'test', 'quiz']
  ```

Template for Solving **LengthSort** with a Private Inner Comparator

```java
import java.util.Arrays;
import java.util.Comparator;

public class LengthSort {
    private class LengthSortImpl implements Comparator<String> {
        @Override
        public int compare(String a, String b) {
            // Need to modify this to solve the problem
            return a.compareTo(b);
        }
    }

    public String[] rearrange(String[] values) {
        Arrays.sort(values, new LengthSortImpl());
        return values;
    }
}
```

Comparable vs. Comparator

- **Comparable a use a.compareTo(b)**
  - What is method signature? One parameter
  - Method in class of which object a is an instance
  - a is this, b is a parameter
- **Comparator c use c.compare(a,b)**
  - Method has two parameters
  - Part of Comparator (Java API link)
- Both return an int:
  - < 0 (means a comes before b)
  - == 0 (means a equals b)
  - > 0 (means a comes after b)
Runtime Complexity of Sort and Comparator?

- Arrays.sort, Collections.sort, call either compareTo (default) or compare (if you give a Comparator)...

- O(N log(N)) compareTo/cmpares, on an Array/List of N elements.

- Exists theoretical proof that this many comparisons is necessary for any comparison-based sorting.

When is comparing once not constant time?

```
public class ListComp implements Comparator<List<Integer>> {
    @Override
    public int compare(List<Integer> list1, List<Integer> list2) {
        int minLength = Math.min(list1.size(), list2.size());
        for (int i=0; i<minLength; i++) {
            int diff = list1.get(i) - list2.get(i);
            if (diff != 0) {
                return diff;
            }
        }
        return 0;
    }
}
```

Overall runtime complexity to sort N ArrayLists, each with M elements, is O(MN log(N)) in the worst case with this Comparator.

java.util.Comparator: Convenient Shorthands

- Comparator.naturalOrder and reversed()

```
shell> Comparator<String> c = Comparator::naturalOrder
       c.compare("a", "b")  // $22 == -1

shell> c.reversed().compare("a", "b")  // $22 == 1
```

- Comparator.comparing

```
shell> Comparator<String> c = Comparator.comparing(String::length)
```

Syntax: `<Types>::method name` to sort something of the Type by the result of some getter method that returns something Comparable.
Comparator-generating shorthands

[sloth, house, owl, ant, mice, kelp]

• Why does "owl" come before "ant"?
  • Stable sort respects order of equal keys

Using .thenComparing shorthand

[ant, owl, kelp, mice, house, sloth]

• First compare by length
  • if same? Compare naturally

Comparator with "lambdas"

• Can also define a comparator with a "lambda expression."

Integer[] nums = {2, 0, 1};
Comparator<Integer> comp = (a, b) -> (b-a);
Arrays.sort(nums, comp);
nums is now {2, 1, 0}
What is printed by the following line of code?

```java
System.out.println("duke".compareTo("devils");
```

- true
- false
- an integer less than 0
- 0
- an integer greater than 0

After sorting, `ar` will be...

```java
String[] ar = {"bird", "dog", "cat", "snake");
Comparator<String> comp = Comparator.comparing(String::length);
Arrays.sort(ar, comp);
```

Ans: [dog, cat, bird, snake]

Suppose you have the following list of lists of integers:

```
[[2, 0, 1], [1, 0, 1], [1, 6]]
```

After sorting, the list would be ordered as...

```
[[1, 0, 1], [1, 6], [2, 0, 1]]
```
Suppose you have an `ArrayList myLists` of `N ArrayLists`, each of size at most `M`. The worst-case runtime complexity to compare any two elements of `myLists` would be...

```java
public class ListComp implements Comparator<List<Integer>> {
    @Override
    public int compare(List<Integer> list1, List<Integer> list2) {
        int minLength = Math.min(list1.size(), list2.size());
        for (int i = 0; i < minLength; i++) {
            int diff = list1.get(i) - list2.get(i);
            if (diff != 0) {
                return diff;
            }
        }
        return 0;
    }
}
```

Ans: `O(M)`

Given an Array of `N Strings`, each of length at most `M`, the worst-case runtime complexity to sort the Array with `java.util.Arrays.sort` is...

Ans: `O(M N log N)`

Efficient sorting algorithms

See example implementations here
Selection Sort with a Loop Invariant

- Loop invariant: On iteration $i$, the first $i$ elements are the smallest $i$ elements in sorted order.

- On iteration $i$...
  - Find the smallest element from index $i$ onward
    - (by loop invariant, must be the next smallest element)
    - Swap that with the element at index $i$

- Algorithm is called Selection Sort.

Selection Sort Code and Runtime

```java
3 public static void selectSort(int[] arr) {
4   for (int i=0; i<arr.length; i++) {
5     int minIndex = i;
6     for (int j=i+1; j<arr.length; j++) {
7       if (arr[j] < arr[minIndex]) {
8         minIndex = j;
9       }
10     }
11     int temp = arr[i];
12     arr[i] = arr[minIndex];
13     arr[minIndex] = temp;
14   }
}
```

Mergesort

- High level idea:
  - Base case: size 1
    - Return list
  - Recursive case:
    - Mergesort(first half)
    - Mergesort(second half)
    - ...
Mergesort

High level idea:
• Base case: size 1
  • Return list
• Recursive case:
  • Mergesort(first half)
  • Mergesort(second half)
  • Merge the sorted halves
  • Return sorted

Mergesort recursive wrapper

• A recursive wrapper method:
  • Is the top-level method a user would call,
  • Is not itself recursive, but makes the initial call to a recursive method,
  • Allows recursive helper method to have additional parameters.

```java
public static void mergeSort(int[] ar) {
    mergeHelper(ar, 0, ar.length);
}
```

Mergesort recursive method

• Should sort everything in ar starting at index l and up to (but not including) index r.

```java
public static void mergeHelper(int[] ar, int l, int r) {
    int diff = r - l;
    if (diff < 2) { return; }
    int mid = l + diff / 2;
    mergeHelper(ar, l, mid);
    mergeHelper(ar, mid, r);
    merge(ar, l, mid, r);
}
```
Merge method concept

- Given two sorted arrays, A and B, want to merge them into one with all values from both.
- Need to keep track of two indices, indexA and indexB.

```
1 3 4
IndexA
```

```
2 5 6
IndexB
```

Merge method

- Given two sorted arrays, A and B, want to merge them into one with all values from both.
- Need to keep track of two indices, indexA and indexB.

```
1 3 4
IndexA
```

```
2 5 6
IndexB
```
Merge method

- Given two sorted arrays, A and B, want to merge them into one with all values from both.
- Need to keep track of two indices, indexA and indexB.
Merge method

• Given two sorted arrays, A and B, want to merge them into one with all values from both.
• Need to keep track of two indices, indexA and indexB.

1 2 3 4 5 6
A
1 3 4
IndexA

2 5 6
B
2 5
IndexB
Merge method

• Given two sorted arrays, A and B, want to merge them into one with all values from both.
• Need to keep track of two indices, indexA and indexB.

```plaintext
Merger:

A: 1 2 3 4 5
B: 1 3 4 2 5 6
```

Index A
Index B

```plaintext
Index A
Index B
```

Merge method

• Given two sorted arrays, A and B, want to merge them into one with all values from both.
• Need to keep track of two indices, indexA and indexB.

```plaintext
Merger:

A: 1 2 3 4 5
B: 1 3 4 2 5 6
```

Index A
Index B

```plaintext
Index A
Index B
```

Merge method

• Given two sorted arrays, A and B, want to merge them into one with all values from both.
• Need to keep track of two indices, indexA and indexB.

```plaintext
Merger:

A: 1 2 3 4 5 6
B: 1 3 4 2 5 6
```

Index A
Index B

```plaintext
Index A
Index B
```
Merge method initialization

- Should merge \( \text{ar}[l...\text{mid}] \) and \( \text{ar}[\text{mid}...r] \)

```java
    public static void merge(int[] ar, int l, int mid, int r) {
        int[] sorted = new int[r-l];
        int sDex=0; int lDex=1; int rDex=mid;

        // Need a new array sorted to put the merged results in, will copy back over ar later.
        // Keeping track of 3 indices:
        //  sDex = where we are in the sorted array
        //  lDex = where we are in \( \text{ar}[l...\text{mid}] \)
        //  rDex = where we are in \( \text{ar}[\text{mid}...r] \)
```

Merge method loop

```java
    while (lDex < mid && rDex < r) {
        if (ar[lDex] < ar[rDex]) {
            sorted[sDex] = ar[lDex];
            lDex++;
        } else {
            sorted[sDex] = ar[rDex];
            rDex++;
        }
        sDex++;
    }
```

Finishing the merge method

- Will finish with \( \text{ar}[l...\text{mid}] \) or \( \text{ar}[\text{mid}...r] \) first, need to copy the rest of the other.
- Then need to copy sorted back onto \( \text{ar}[l...r] \)

```java
    if (lDex == mid) System.arraycopy(ar, rDex, sorted, sDex, r-rDex);
    else System.arraycopy(ar, lDex, sorted, sDex, mid-lDex);
    System.arraycopy(sorted, sorted[0], ar, l, r-l);
```

- Code uses the \textbf{System.arraycopy} method:
Is this any faster? Empirically...

<table>
<thead>
<tr>
<th>N (thousands)</th>
<th>Selection sort (ms)</th>
<th>Insertion sort (ms)</th>
<th>Merge sort (ms)</th>
<th>Java.util Arrays.sort (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10k</td>
<td>22</td>
<td>40</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>30k</td>
<td>168</td>
<td>334</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>90k</td>
<td>1481</td>
<td>967</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>270k</td>
<td>13175</td>
<td>8716</td>
<td>22</td>
<td>14</td>
</tr>
</tbody>
</table>

Why mergesort is $O(N \log(N))$, intuition

• Halves at each level, so just $O(\log(N))$ levels.

• If we can do all of the merges at each level in $O(N)$ time?

• Overall $O(N \log(N))$.

Recursion tree

Visualization from the Zybook
Recurrence Relations

Analyzing Recursive Runtime

Develop a recurrence relation of the form

\[ T(N) = a \cdot g(N) + f(N) \]

Where:

- \( T(N) \) - runtime of method with input size \( N \)
- \( a \) is the number of recursive calls
- \( g(N) \) - how much input size decreases on each recursive call
- \( f(N) \) - runtime of non-recursive code on input size \( N \)

Analyzing Runtime of Recursive Reverse

```java
public static ListNode reverse(ListNode list) {
    if (list == null || list.next == null) {
        return list;
    }
    ListNode reversedLast = list.next;
    ListNode reversedFirst = reverse(list.next);
    reversedLast.next = list;
    list.next = null;
    return reversedFirst;
}
```

\[ T(N) = T(N-1) + O(1) \]
Solving Recurrence Relations

\[ T(N) = T(N-1) + 1 \]
\[ = (T(N-2) + 1) + 1 \]
\[ = T(N-3) + 3 \]
\[ \vdots \]
\[ = T(1) + N \]
\[ = O(N) \]

Recurrence relations and expectations in 201

- In general, will **not** be asked to solve recurrence relations on exams (for later classes in theory).
- You **will** be asked to determine the recurrence relation of a given algorithm/code.

<table>
<thead>
<tr>
<th>Recurrence</th>
<th>Algorithm</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(n) = T(n/2) + O(1) )</td>
<td>binary search</td>
<td>( O(\log n) )</td>
</tr>
<tr>
<td>( T(n) = T(n-1) + O(1) )</td>
<td>sequential search</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>( T(n) = 2T(n/2) + O(1) )</td>
<td>tree traversal</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>( T(n) = 2T(n/2) + O(n) )</td>
<td>quicksort, mergesort</td>
<td>( O(n \log n) )</td>
</tr>
<tr>
<td>( T(n) = T(n-1) + O(n) )</td>
<td>selection or bubble sort</td>
<td>( O(n^2) )</td>
</tr>
</tbody>
</table>

Runtime complexity of mergesort?

Let \( N = r-l \), the number of elements to sort.

```java
public static void mergeHelper(int[] arr, int l, int r) {
    int diff = r-l;
    if (diff < 2) { return; }
    int mid = l + diff/2;
    mergeHelper(arr, l, mid);
    mergeHelper(arr, mid, r);
    merge(arr, l, mid, r);
}
```