LECTURE 15: SORTING

ANNOUNCEMENTS:

READING:

DS&SD: pg. 92, 98-100; 5.3, 18.1-18.4, 18.5 is optional
DS&A: Chap. 15

OVERVIEW:

Motivation and applications for sorting
Brief mathematical theory (optional)
Example basic technique: select sort
More sophisticated: quick sort, merge sort
Choosing a sorting technique
Realistic choice: API's sort
Suggested exercises

1. Motivation

1.1 Help Searching

- see binary search

Study of Algorithms

1.2 Trend Analysis

- economic trends
- scientific trends
- grading trends

2. Mathematical Theory

2.1 See DS&A, pp. 491-492

2.2 The Gist:

- you need sortable values
- you need to collect some values together
- you need a comparison relationship

With these “standards” you can mathematically organize your information!

2.3 Stable sort:

- why learn? API uses this term a lot!
- problem: equal elements pose some problems
- stable sort maintains equal element’s relative positions

3. Sorting Classifications

Insertion
- insert sort
selection
- select sort
heap sort
exchange
- bubble sort
quick sort
merge
- merge sort
distribution
- bucket sort
radix sort
4. Select Sort

4.1 Motivation

- ease in remembering
- ease in understanding

4.2 Algorithm

- pick an element from end of unsorted portion of array
- compare that element with the rest of the elements
- if another element is the largest/smallest, swap with the current element
- repeat

4.3 Code

See SelectSortUp and BasicSorting

```java
// sort in ascending order:
public static void selectSortUp(Comparable[] a) {
    for (int i=0; i < a.length; i++) {
        int minPos = i;
        for (int rem = i+1; rem < a.length; rem++)
            if (a[rem].compareTo(a[minPos]) < 0) { 
                minPos = rem;
            }
        Comparable tmp = a[i];
        a[i] = a[minPos];
        a[minPos] = tmp;
    }
}
```

4.4 Analysis

See DS&SD, pp. 98-100

- \( T(n) = 2n^2 + 7n - 7 \)
- \( \Rightarrow T(n) \in O(n^2) \)

The Gist: count comparisons \( \text{compareTo}s \)

- assume \( n \) elements in array
- 1st time: \( n-1 \) comparisons for \( i=0 \) elem
- 2nd time: \( n-2 \) comparisons for \( i=1 \) elem
- ...
- last time: 1 comparison for \( i=n-1 \) elem

Derivation:

- Total: \( (n-1) + (n-2) + \ldots + 1 \)
- Math: \( 1 + 2 + \ldots + n = \frac{(n)(n+1)}{2} \)
- So, \( T_A(n) = \frac{(n-1)n}{2} \in O(n^2) \)

5. Quick Sort

5.1 Motivation

- want to improve time
- want to keep sorting in place (not create tmp space)

5.2 Algorithm

Divide and conquer!

- Given array \( x \) and pivot value \( p \)
- Partition array into subarrays Right and Left
  - \( R \) contains elements of \( x \geq p \)
  - \( L \) contains elements of \( x < p \)
- Concatenate elements into \([L \ p \ R]\)
- Sort \( R \) and \( L \) separately (recursive!)
5.3 Diagram

60 20 30 10 40 50 70 90 80  
20 10 30 40 60 50 70 90 80  
20 10 30 40 50 60 70 90 80  
10 20 30 40 50 60 70 80 90  

5.4 Top-Level Design

Need recursive method for sorting:

```java
// sort x[a..b]:
void quickSort(Comparable[] x, int a, int b)
QuickSort picks pivot, splits array, sorts each portion...
So, we need way to partition array:

- move items < pivot to one side (left)
- move items > pivot to the other side (right)
```

// reorganize array into [left, pivot, right]:
```java
int partition(Comparable[] x, int low, int high, Comparable pivot)
```

5.5 Partitioning

```
B Y B Y B Y B Y B Y B Y blue and yellow
| | L R L and R "fingers"
B Y B Y B Y B Y B Y B Y B Y
| | L R
```

We want to move blues and yellows to their own sides
(blue on left, yellow on right)

So, move the fingers on each side until hit wrong color

```
B Y B Y B Y B Y B Y B Y B Y
| | L R
```

We can swap colors and keep advancing indices

```
B Y B Y B Y B Y B Y B Y B Y B Y B Y B Y
| | L R
```

Eventually, L and R fingers will cross or match:

```
B B B B B Y Y Y Y Y Y Y
| | L R
```

We're done!

5.6 Code for Partition

```java
public static int partition(Comparable[] x,
int low, int high, Comparable pivot) {
    int i = low; int j = high;
    boolean flag = true;
    while (flag) {
        // advance left index to the right:
        while((x[i].compareTo(pivot)<0) &&
              (i < x.length)) i++;
        // advance right index to the left:
        while((pivot.compareTo(x[j])<0) &&
              (j >= 0)) j--;
        // swap:
        if (i < j) {
            Comparable temp = x[i];
            x[i] = x[j];
            x[j] = temp;
            i++;
            j--;
        } else flag=false;
    }
    // indices have "met" (i==j):
    return i;
}
```
5.7 Sorting Left and Right

Partitioning moves elements to left (<) or right (>=)

How to pick pivot?
- median value (expensive)
- use first value of array (kind of random)
- can use middle element or average of 1st, last, middle

5.8 Code for quickSort

```java
public static void quickSort(Comparable[] x, int a, int b) {
    if (a < b) {
        // pick "random" pivot and partition x:
        int p = partition(x, a+1, b, x[a]);
        // move pivot into its final resting place
        // swap x[p-1] and x[a]:
        Comparable temp = x[p-1];
        x[p-1] = x[a];
        x[a] = temp;
        // sort left and right partitions:
        quickSort(x, a, p-1);
        quickSort(x, p, b);
    }
}
```

5.9 Analysis

Worst case: wretched pivot (largest or smallest elem)
- $T(n) \in O(n^2)$ (see books for proof)

Best case: picked good pivot
- $T(n) \in O(n \log n)$ (see books for proof)

5.10 Improvements on Quick Sort

Pad left and right ends of array with sentinels:
- `Integer.MIN_VALUE`
- `Integer.MAX_VALUE`

Even more improvements! (see books)

6. Merge Sort

6.1 Motivation
- not worried about needing extra space for sorting
- want a dependable, "cheap" algorithm

6.2 Algorithm: Divide and conquer!

Divide array in halves (keep track of indices)
sort each part (call merge sort recursively)
merge sorted arrays a1 and a2:
- create array m: size(m) = size(a1)+size(a2)
- keep 3 indices: p1 for a1, p2 for a2, pm for m (p1, p2, m initialized to 0)
- compare a1[0] and a2[0] & move smaller (say a2) into 1st element of m, increment p2 and pm
- general: compare a1[p1] with a2[p2] and move smaller one into m[pm] and increment corresponding index
- if a1 or a2 depleted, copy remaining elems into m

6.3 Diagram
6.4 Code

```java
public static Comparable[] mergeSort(Comparable[] x, int low, int high) {
    // at least three elements:
    if (low < high - 1) {
        int mid = (low + high)/2;
        Comparable[] x1 = mergeSort(x, low, mid);
        Comparable[] x2 = mergeSort(x, mid + 1, high);
        return merge(x1, x2);
    }
    // 0, 1, or 2 elements:
    else {
        int length = high - low + 1;
        Comparable[] r = new Comparable[length];
        if (length == 1) r[0] = x[low];
        if (length == 2) {
            if (x[low].compareTo(x[high]) < 0) {
                r[0] = x[low];
                r[1] = x[high];
            } else {
                r[0] = x[high];
                r[1] = x[low];
            }
        }
        return r;
    }
}
```

```java
public static Comparable[] merge(Comparable[] a1, Comparable[] a2) {
    Comparable[] m = new Comparable[a1.length + a2.length];
    int p1 = 0;
    int p2 = 0;
    int pm = 0;
    while ((p1 < a1.length) && (p2 < a2.length)) {
        if (a1[p1].compareTo(a2[p2]) <= 0) {
            m[pm++] = a1[p1++];
        } else {
            m[pm++] = a2[p2++];
        }
    }
    // either a1 or a2 will empty at this point:
    // merge any leftovers from a1
    for (; p1 < a1.length; p1++)
        m[pm++] = a1[p1];
    // merge any leftovers from a2
    for (; p2 < a2.length; p2++)
        m[pm++] = a2[p2];
    return m;
}
```

6.5 Analysis of Merge Sort

Asymptotic complexity: $O(n \log n)$

refer to textbooks for proof

MS is asymptotically optimal algorithm for sorting

disadvantage: needs extra storage for merging

7. API (“Real World”)

See http://java.sun.com/j2se/1.4/docs/api/java/util/
    Arrays.html#sort(java.lang.Object[])

    public static void sort(Object[] a)
        • a must implement Comparable
        • guaranteed to be stable
        • modified merge sort!

See http://java.sun.com/j2se/1.4/docs/api/java/util/
    Collections.html#sort(java.util.List)

    public static void sort(List list)
        • a must implement Comparable
        • guaranteed to be stable
        • modified merge sort!

8. Suggested Exercises

• determine how to do bubble and insert sorts; determine their asymptotic complexity
• write select sort from left to right and right to left; sort up and sort down
• write non-recursive merge sort (see DS&SD: 5.3)
• improve quick sort with “median of 3” pivot