CS 61B
Spring 2017

Sorting Algorithms

Discussion 13: April 19, 2017

1 Warmup

Algorithm	Best-case	Worst-case	Stable
Selection Sort	$\Theta(N^2)$	$\Theta(N^2)$	Depends
Insertion Sort	$\Theta(N)$	$\Theta(N^2)$	Yes
Merge Sort	$\Theta(N \log N)$	$\Theta(N \log N)$	Yes
Quicksort	$\Theta(N)$	$\Theta(N^2)$	Depends
Heapsort	$\Theta(N \log N)$	$\Theta(N \log N)$	Hard

1.1 Give a best and worst case input for insertion sort.

Best case is a completely sorted array with 0 inversions while the worst case is a reverse-sorted array with $\Theta(N^2)$ inversions. Recall that the runtime for insertion sort is given by $\Theta(N + K)$ where K is the number of inversions.

1.2 Do you expect selection or insertion sort to run more quickly on a reverse list?

Selection sort is useful here since performs only $\Theta(N)$ swaps as opposed to $\Theta(N^2)$ swaps in insertion sort's case.

1.3 Give a worst case input for quicksort. Assume that we're always picking the left-most item as our pivot.

Any sequence where the leftmost element of the list is always either the minimum element or the maximum element.

1.4 Why does Java's built-in `Array.sort` method use quicksort for `int`, `long`, `char`, or other primitive arrays, but merge sort for all `Object` arrays?

Fast, in-place solutions for quicksort are unstable, meaning that, for any two equivalent keys, their final order in the output is not guaranteed to be the same.

2 Inversions

We have a list of N elements that should be sorted, but to our surprise we recently discovered that there are at most k pairs out of order, or k **inversions**, in the list. As a small example, the list `{ 0, 1, 2, 6, 4, 5, 3 }` contains 5 inversions: $(6,4)$, $(6,5)$, $(6,3)$, $(4,3)$, $(5,3)$.

3 Decisions

3.1 Given a list of objects that is totally sorted except for $K < \log N$ randomly chosen elements that are out of place, choose the best sorting algorithm to solve the problem and give a tight asymptotic runtime bound.

Use insertion sort. Since each element is at most N spaces away from its final position, we will need to spend $O(N)$ time moving the item. The overall runtime is in $O(N + NK) \in O(NK)$.

3.2 What's the best sorting algorithm to apply on an array of N randomly distributed objects? Give a tight asymptotic runtime bound as well.

Merge sort is the best option since its design makes the algorithm stable, preserving the relative order of equal items in the list. Quicksort and heap sort are also acceptable, but because their fast or usual implementations are unstable, we prefer not to use them on Java reference types.

4 What's that sort?

4.1 Each of the following sequences represent an array being sorted at some intermediate step. Match each sample with one of the following sorting algorithms: **insertion sort, selection sort, heap sort, merge sort, quicksort**. The original array is below.

```
5103 9914 0608 3715 6035 2261 9797 7188 1163 4411
```

(a)
```
5103 9914 0608 3715 2261 6035 7188 9797 1163 4411
0608 2261 3715 5103 6035 7188 9797 9914 1163 4411
```

Merge sort

(b)
```
0608 1163 5103 3715 6035 2261 9797 7188 9914 4411
0608 1163 2261 3715 6035 5103 9797 7188 9914 4411
```

Selection sort

(c)
```
9797 7188 5103 4411 6035 2261 0608 3715 1163 9914
4411 3715 2261 0608 1163 5103 6035 7188 9797 9914
```

Heap sort

(d)
```
5103 0608 3715 2261 1163 4411 6035 9914 9797 7188
0608 2261 1163 3715 5103 4411 6035 9914 9797 7188
```

Quicksort

(e)
```
0608 5103 9914 3715 6035 2261 9797 7188 1163 4411
0608 2261 3715 5103 6035 9914 9797 7188 1163 4411
```

Insertion sort