

LINEAR SORTING**Reading:** CLR Chapter 9-----

The Best Worst-Case Running Time for Comparison-Based Sorts

A few lectures back, we saw that *any* comparison-based sort must perform $(n \lg(n))$ comparisons in the worst case. This was based on a simple counting argument involving **decision trees**, binary trees that represent how each permutation (represented by a leaf) is ultimately determined by a sequence of comparisons (represented by the internal nodes of the tree).

Proof sketch:

- There are $n!$ permutations = leaves in the tree.
- The minimum height of a tree holding $n!$ leaves is about $\lg(n!)$ for a balanced tree.
- $\lg(n!) = (n \lg(n))$ via Stirling's equation or calculus (see CLR Exercise 9.1-2).

Linear Sorting Algorithms

How can this be possible? Not comparison based! Take advantage of some feature of input.

E.g.

- Dutch National Flag
- Integer **bucket sort** for integers in the range $1..k$.

E.g. 4 2 6 3 2 4 2 6 1 2 4

Counting Sort (a.k.a. Distribution Counting)

Integer bucket sort has a problem when there is satellite data:

4₁ 2₁ 6₁ 3₁ 2₂ 4₂ 2₃ 6₂ 1₁ 2₄ 4₃

```
Counting-Sort(A,B,k)
{A is input, B is output}
for i <- 1 to k do
  Count[i] <- 0
for j <- 1 to length[A] do
  Count[A[j]] <- Count[A[j]] + 1
for i <- 2 to k do
  C[i] <- C[i] + C[i-1]
for j <- length[A] downto 1 do
  B[C[A[j]]] <- A[j]
  C[A[j]] <- C[A[j]] - 1
```

Is Counting-Sort stable?

Would Counting-Sort be stable if the for j ... loop counted up rather than down?

Bucket Sort

Can't use counting sort if keys aren't integers in a given range. What if keys are real numbers in a given range? If keys are evenly distributed can use a **general bucket sort**:

Step 1: If given n elements, make an array of n buckets, each of which contains an empty list.

Step 2: Insert each element into the list of the bucket chosen by matching its key with the result of dividing the range into n equal-sized intervals

Step 3: Use any $O(n^2)$ sorting algorithm (even quicksort) to sort the resulting lists in the buckets.

Step 4: Read the elements from the lists in order back into the input array.

In the worst case, all elements end up in same bucket and running time is $O(n^2)$.

In best case, there is one element per bucket, and running time is $O(n)$.

Assuming evenly distributed keys, can use probability analysis to show average case running time is $O(n)$. (See CLR for details.)

Radix Sort

```
Radix-Sort(A, d)
for i ← 1 to d do
  Stable-Sort(A, i) {use digit i for comparison}
```

E.g.

```
443
124
232
431
132
123
321
411
```

Must process digits right-to-left, not left-to-right!

Running time = $O(dn + kd)$, where d is number of digits and digits are in range $[1..k]$.

Note: in many applications, $d = \lg(n)$.
