

LINEAR SORTING

Reading: CLR Chapter 9

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

Given n distinct elements, how many different arrays can we make with these elements?

Can view comparison-based sorts as **decision trees** that process sets of arrays. Each leaf of the tree must hold a singleton array. For example:

What is the minimum height of a decision tree for arrays of length n ?

(Note that $\lg(n!) = (n \lg(n))$ 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 array of length n containing keys in range [1..k].
 B is output array of length n}
Count <- new array(k)
for i <- 1 to k do
  Count[i] <- 0
{Find count of each key in A.}
for j <- 1 to length[A] do
  Count[A[j]] <- Count[A[j]] + 1
{Find partial sums of key counts in A.}
for i <- 2 to k do
  C[i] <- C[i] + C[i-1]
{Put elements in final position.}
for j <- length[A] downto 1 do
  B[C[A[j]]] <- A[j]
  C[A[j]] <- C[A[j]] - 1
```

Example:

How much time does Counting Sort need?

How much space does Counting Sort need?

Is Counting-Sort stable?

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

General 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 to sort the resulting lists in the buckets.

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

Example: Sorting 10 numbers between 0 and 1:

.63
.75
.16
.65
.61
.17
.89
.92
.28
.56

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

Idea: Suppose you had a machine that could perform a stable sort on the i th digit of a d -digit number. How could you use the machine to sort a "pile" of n d -digit numbers?

E.g.

443
124
232
431
132
123
321
211
121
441
122
442

To avoid lots of intermediate piles, process digits right-to-left, not left-to-right!

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

Any stable sort will do; Counting Sort is a good candidate since it's $\Theta(n)$.

What is the running time for sorting n d -digit numbers, where digits are in the range $[1..k]$?

Note: in many applications, $d = \log_k(n)$, so the sort ends up being $\Theta(n \log_k(n))$ for a given k .

How can we use radix sort to sort a million 32-bit numbers?