Sorting

CS 624 — Analysis of Algorithms

February 15, 2024

How Fast Can We Sort?

Of the sorting algorithms we have seen so far, we have:

- fastest best-case running time: $O(n)$
- fastest worst-case running time: $O(n \log n)$

Can we do better?

Is there a sorting method whose worst-case running time is $O(n)$? Obviously, we cannot do better than that. (Why?)

If there is a better sorting algorithm, how do we find it?

Comparison-Based Sorting

All of our sorting algorithms so far are based on comparison.

Recall the second analysis of Quicksort: The running time of Quicksort is proportional to the number of comparisons.

Any comparison-based sorting algorithm, for a particular input size, can be modeled by a **binary decision tree**:

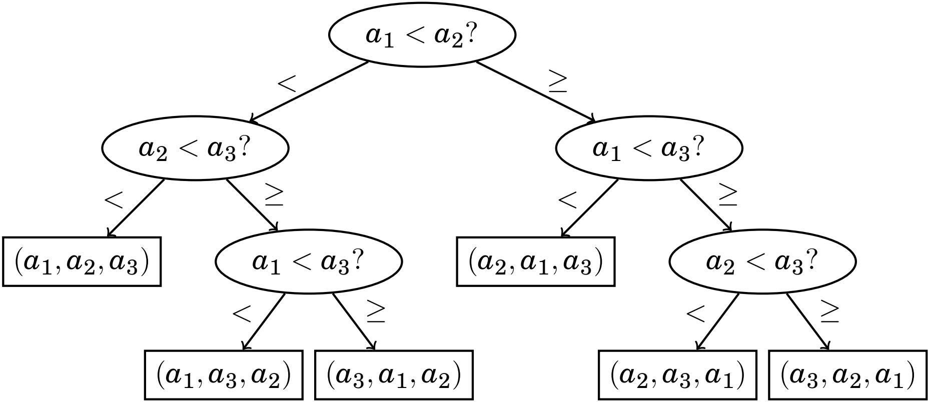
- Each non-leaf node represents a comparison, with different sub-trees depending on the result of the comparison.
- Each leaf node represents a final result.

What do known algorithms look like as binary decision trees?

What would an optimal algorithm look like as a binary decision tree?

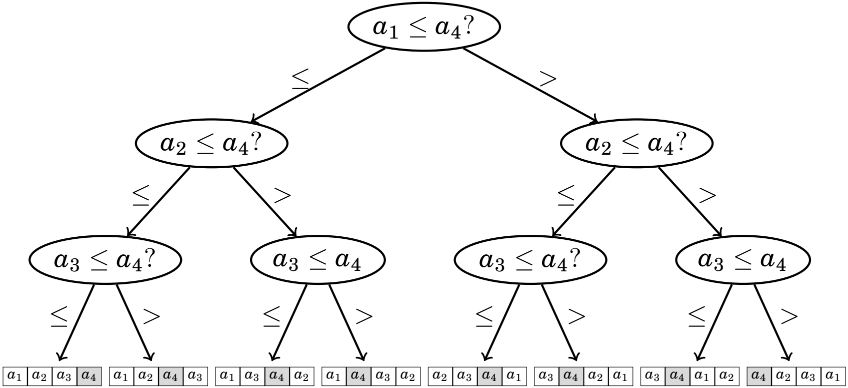
Binary Decision Tree for Insertion Sort

The run of InsertionSort on an array of 3 elements: $\{a_1, a_2, a_3\}$:



Binary Decision Tree for Partition

The run of Quicksort on an array of 4 elements: $\{a_1, a_2, a_3, a_4\}$, first partition:



Bound on Comparison-Based Sorting Algorithms

Theorem

Any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons in the worst case, so its worst-case running time is $\Omega(n \log n)$.

Proof.

The worst-case running time is bounded below by the depth of the decision tree. The number of leaves in the decision tree must be the number of possible permutations, which is $n!$.

The depth of a binary tree with L leaves is $\Omega(\log L)$.

Therefore the depth of the decision tree is $\Omega(\log n!) = \Omega(n \log n)$. $\square$

Still, Can We Do Better?

If there is a faster sorting algorithm, it is not based on comparisons.

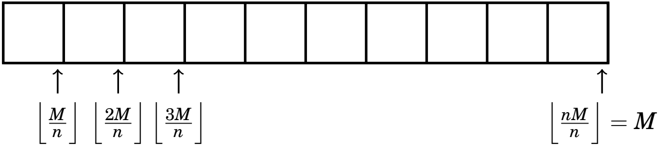
We need to rely on stronger assumptions about the array elements.

- Example: elements are integers in the range $1 \mathrel{..} M$
- We are sacrificing generality for speed.

A More Complicated Example

- Input: Given n integers, $\{a_1, a_2, \dots, a_n\}$.
- Each integer a_k is in the range of $1 \mathinner{..} M$ where $M \geq n$.
- Create an array $A[1 \mathinner{..} n]$ where the elements are **buckets**.
  Each bucket is a list of integers, initially empty.
  Each bucket corresponds to a contiguous range of integers in $1 \mathinner{..} M$.
- Each integer a_k is put in the appropriate bucket.
- At the end, each bucket is sorted, and the sorted buckets are concatenated.

Illustration of Bucketsort



The largest number bucket $A[j]$ can hold is $\left\lfloor \frac{jM}{n} \right\rfloor$. Therefore the index j of the bucket where number a_k belongs must satisfy

$$\left\lfloor \frac{(j-1)M}{n} \right\rfloor + 1 \leq a_k \leq \left\lfloor \frac{jM}{n} \right\rfloor$$

Since we always have $x - 1 < \lfloor x \rfloor$, this yields

$$\frac{(j-1)M}{n} < a_k \leq \frac{jM}{n} \implies j - 1 < \frac{a_k n}{M} \leq j \implies j = \left\lceil \frac{a_k n}{M} \right\rceil$$

Bucketsort Running Time

The running time of Bucketsort consists of

- the time to put the numbers into their buckets: $\Theta(n)$
- the time to sort each bucket: $???$
- the time to concatenate the sorted buckets: $\Theta(n)$

Sorting the Buckets

Problem: Elements are not necessarily uniformly distributed in buckets. Some buckets may be are empty, some may contain several elements.

- What is the average cost of sorting the buckets?
- Suppose we use InsertionSort to sort each bucket (good for small buckets).
- Do not assume that since the average number of elements per bucket is $O(1)$ it means that the average runtime is O(n).

Sorting the Buckets

- If bucket i has n_i elements, sorting it takes $O(n_i^2)$
- We can average over all the buckets.
- Since the distribution of elements in buckets is random we can average on n_1 (since it doesn't matter which bin we pick).
- The expected value of n_1 is

  $$\sum_{j=0}^{n} \text{(probability that } j \text{ numbers land in bucket 1)} \cdot j^2$$

Sorting the Buckets

- The probability of j items landing in bucket 1 is the probability of selecting j items out of n, $\binom{n}{j}$.
- The probability of a particular combination is: $(\frac{1}{n})^j(1 - \frac{1}{n})^{n-j}$.
- The probability of any j elements landing in bucket 1 is: $\left(\frac{1}{n}\right)^j \left(1 - \frac{1}{n}\right)^{n-j} \binom{n}{j}$.
- The expected runtime of sorting n_1 is then $\sum_{j=0}^{n} \left(\frac{1}{n}\right)^j \left(1 - \frac{1}{n}\right)^{n-j} \binom{n}{j} j^2$

Bucketsort analysis

This looks like a binomial generating function that has been differentiated. So let us set:

$$f(x) = \sum_{j=0}^{n} \left(\frac{1}{n}\right)^j \left(1 - \frac{1}{n}\right)^{n-j} \binom{n}{j} x^j$$

Then we have

$$f'(x) = \sum_{j=0}^{n} \left(\frac{1}{n}\right)^j \left(1 - \frac{1}{n}\right)^{n-j} \binom{n}{j} j x^{j-1}$$

We can't just differentiate again, because we would get $j(j-1)$. So we multiply by x first:

$$xf'(x) = \sum_{j=0}^{n} \left(\frac{1}{n}\right)^j \left(1 - \frac{1}{n}\right)^{n-j} \binom{n}{j} j x^j$$

and then we can differentiate:

$$(xf'(x))' = \sum_{j=0}^{n} \left(\frac{1}{n}\right)^j \left(1 - \frac{1}{n}\right)^{n-j} \binom{n}{j} j^2 x^{j-1}$$

Bucketsort analysis

- Let us set $g(x) = \left(xf'(x)\right)'$.
- Then we see that the expected value of n_1^2 is just $g(1)$.
- The closed form of f follows from the binomial theorem:

$$f(x) = \sum_{j=0}^{n} \left(\frac{1}{n}\right)^j \left(1 - \frac{1}{n}\right)^{n-j} \binom{n}{j} x^j$$

$$= \sum_{j=0}^{n} \left(\frac{x}{n}\right)^j \left(1 - \frac{1}{n}\right)^{n-j} \binom{n}{j} \qquad = \left(1 + \frac{x-1}{n}\right)^n$$

Bucketsort analysis

Going back to g:

$$f'(x) = n\left(1 + \frac{x-1}{n}\right)^{n-1} \cdot \frac{1}{n} = \left(1 + \frac{x-1}{n}\right)^{n-1}$$

and then

$$\begin{aligned} g(x) = \left(xf'(x)\right)' &= \left(x\left(1 + \frac{x-1}{n}\right)^{n-1}\right)' \\ &= \left(1 + \frac{x-1}{n}\right)^{n-1} + (n-1)x\left(1 + \frac{x-1}{n}\right)^{n-2} \cdot \frac{1}{n} \\ &= \left(1 + \frac{x-1}{n}\right)^{n-1} + \left(1 - \frac{1}{n}\right)x\left(1 + \frac{x-1}{n}\right)^{n-2} \end{aligned}$$

By substituting 1 for x, we get $g(1) = 1 + \left(1 - \frac{1}{n}\right) = 2 - \frac{1}{n}$. That is the expected value of n_1^2, and in fact is the expected value of n_i^2 for any i. In short – the average time for sorting each bucket in bucketsort is O(1) and the overall expected runtime is O(n).

So... Why Not Always Bucketsort?

- And what happened to our lower bound?
- We are not using a binary decision tree!
- This is only because we know something about the input.
- Also – we did only average case analysis. What is the worst case?