Zhengyi Xiao
Sort-Algo-Project
October 12, 2019

Insertion sort on small arrays in merge sort
Although merge sort runs in $\Theta(n \lg n)$ worst-case time and insertion sort runs in.$\Theta(n^2)$ worst-case time, the constant factors in insertion sort can make it faster in practice for small problem sizes on many machines. Thus, it makes sense to coarsen the leaves of the recursion by using insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge sort in which $n = k$ sublists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.
a. Show that insertion sort can sort the n/k sublists, each of length k, in $\Theta(nk)$ worst-case time.

Because for the insertion sort, we have $\Theta(k^2)$, which means that the run-time is $ak^2 + bk + c$. Because there are n/k many sublists, we have

$$\frac{n}{k}\left(ak^2 + bk + c)\right) = ank + bn + \frac{cn}{k} = \Theta(nk)$$

b. Show how to merge the sublists in $\Theta(n \lg(n/k)$ worst-case time.

Since the merge sort on the sets of length of α has the overall runtime of

$$T(\alpha) = \begin{cases} 0 & \text{if } \alpha = 1, \\ 2T(\alpha/2) + D(\alpha) + C(\alpha) & \text{if } \alpha = 2^p \text{ with } p > 0.. \end{cases}$$

The divide step just computes the middle of the subarray, which takes constant time. Thus, $D(\alpha) = \Theta(1)$. Merge procedure on an k-element subarray takes time $\Theta(k\alpha)$ from the previous question. We then have

$$T(\alpha) = \begin{cases} 0 & \text{if } \alpha = 1, \\ 2T(\alpha/2) + k\alpha & \text{if } \alpha = 2^p \text{ with } p > 0. \end{cases}$$

In order to show that it has the $\Theta(n \lg(n/k)$ worst-case time, we want to use induction here and make the guess that $T(\alpha) = \Theta(\alpha k \lg \alpha)$.

The base case: $T(1) = 1k \lg 1 = 0$.

The inductive step: Assume that $T(\alpha) = \alpha k \lg \alpha$ is true, we want to prove that $T(2\alpha) = 2\alpha k \lg(2\alpha)$(since $\alpha = 2^p$, the next one is 2^{p+1}). We then have

$$\begin{aligned} T(2\alpha) &= 2T(\alpha) + 2k\alpha \\ &= 2(\alpha k \lg \alpha) + 2k\alpha \\ &= 2k\alpha(\lg \alpha + 1) \\ &= 2k\alpha(\lg \alpha + \lg 2) \\ &= 2k\alpha(\lg(2\alpha)), \end{aligned}$$

which is exactly what we want to show. And it completes the induction proof.

Now, we have the list of n/k elements and by plugging in we find

$$\begin{aligned} T(n/k) &= k\frac{n}{k} \lg\left(\frac{n}{k}\right) \\ &= n \lg(n/k). \end{aligned}$$

Thus, this merge sort has $\Theta(n \lg(n/k)$ in the worst-case time.

c. Given that the modified algorithm runs in $\Theta(nk + n\lg(n/k))$ worst-case time, what is the largest value of k as a function of n for which the modified algorithm has the same running time as standard merge sort, in terms of Θ notation?

By noticing that if $k = \lg n$, we have $\Theta(nk + n\lg(n/k)) = \Theta(n\lg n + n\lg(n/\lg n) = \Theta(n\lg n)$. Thus, if $k < \lg n$, the modified algorithm is better than the original method.

d. How should we choose k in practice?

k should be the largest possible of the length of list that makes the insertion sort is faster than the normal merge combining.

Median-of-3 partition
One way to improve the RANDOMIZED-QUICKSORT procedure is to partition around a pivot that is chosen more carefully than by picking a random element from the subarray. One common approach is the **median-of-3** method: choose the pivot as the median (middle element) of a set of 3 elements randomly selected from the subarray. (See Exercise 7.4-6.) For this problem, let us assume that the elements in the input array $A[1...n]$ are distinct and that $n \geq 3$. We denote the sorted output array by $A'[1...n]$. Using the median-of-3 method to choose the pivot element x, define $p_i = \Pr\{x = A'[i]\}$.

a. Give an exact formula for p_i as a function of n and i for $i = 2, 3, ..., n-1$. (Note that $p_1 = p_n = 0$.)

It a probability question because we are choosing three from n elements. Since the order of pick does not matter here, since we have $\frac{n!}{(n-3)!}$ ways to choose. Moreover, there are $i-1$ ways ways to pick a smaller one, $n-i$ ways to pick the larger one, and 3! ways to arrange them, we get

$$p_i = \frac{6(i-1)(n-i)}{n(n-1)(n-2)}.$$

b. By what amount have we increased the likelihood of choosing the pivot as $x = A'[\lfloor (n+1)/2 \rfloor]$, the median of $A[1..n]$, compared with the ordinary implementation? Assume that $n \to \infty$, and give the limiting ratio of these probabilities.

Originally we have the likelihood of $1/n$, but now it is p_i. Therefore, we get

$$\begin{aligned}\lim_{n\to\infty} \frac{6(i-1)(n-i)}{n(n-1)(n-2)} \Big/ \frac{1}{n} &= \lim_{n\to\infty} \frac{-(6-3n)(n/2)}{(-2+n)(-1+n)} \\ &= \frac{6}{4}\end{aligned}$$

c. If we define a "good" split to mean choosing the pivot as $x = A'[i]$, where $n/3 \leq i \leq 2n/3$, by what amount have we increased the likelihood of getting a good split compared with the ordinary implementation? (Hint: Approximate the sum by an integral.)

We want to find the pdf from $n/3$ to $2n/3$, which is

$$\begin{aligned}\lim_{n\to\infty}\int_{n/3}^{2n/3}\frac{6(i-1)(n-i)}{n(n-1)(n-2)}di &= \lim_{n\to\infty}\frac{6}{n(n-1)(n-2)}\int_{n/3}^{2n/3}(i-1)(n-i)di \\ &= \lim_{n\to\infty}\frac{6}{2n-3n^2+n^3}\left(\frac{n(2-3n+n^2)}{6}\right) \\ &= \frac{13}{27}. \qquad \text{(By Wolfram Alpha)}\end{aligned}$$

Therefore, as n grows the improvement is $13/27 * 3 \approx 1.44$.

d. Argue that in the $\Omega(n \lg n)$ running time of quicksort, the median-of-3 method affects only the constant factor.

Because it does not change a fact that the modified quick sort algorithm is still a comparison sort, which always has a $\Omega(n \lg n)$(Theorem 8.1).