Chapter 5

Reserves Recitation.

5.1 More Sorting, More Correctness.

Until now, all the algorithms that we have seen were, in some sense, intuitive. We could describe in words, step by step, exactly what the algorithm does. For example, bubble and heapsort both bubble up the greatest element among the remaining elements in each iteration. Merge sort divides the task into subtasks on smaller inputs, starting with sorting the first and second halves of the given array, and then merging the sorted subarrays.

We are about to present another $\Theta(n^2)$-sorting algorithm, whose correctness is not so obvious. The algorithm was developed by Stanley P. Y. Fung, [Fun21], who coined its name - "ICan'tBelieveItCanSort" - due to the surprise of having such a simple sorting algorithm. It's worth mentioning that, despite its simplicity, Fung came up with this algorithm in 2021.

Result: Sorting $A_1, A_2, ..A_n$

1 **for** $i \in [n]$ **do**
2   **for** $j \in [n]$ **do**
3   **if** $A_j < A_i$ **then**
4   swap $A_i \leftrightarrow A_j$
5   **end**
6   **end**
7 **end**

Algorithm 1: "ICan'tBelieveItCanSort" alg.

Claim 5.1.1. *After the ith iteration, $A_1 \leq A_2 \leq A_3.. \leq A_i$ and A_i is the maximum of the whole array.*

Proof. By induction on the iteration number i.

1. Base. For $i = 1$, it is clear that when j reaches the position of the maximum element, an exchange will occur and A_1 will be set to be the maximum element. Thus, the condition on line (3) will not be satisfied for the remaining j-iterations of the inner loop. Therefore, at the end of the first iteration, A_1 is indeed the maximum.

2. Assumption. Assume the correctness of the claim for any $i' < i$.

3. Step. Consider the ith iteration. And observe that if $A_i = A_{i-1}$ then A_i is also the maximal element in A, namely no exchange will be made in the ith iteration, yet $A_1 \leq A_2 \leq .. \leq A_{i-1}$ by the induction assumption, thus $A_1 \leq A_2 \leq .. \leq A_{i-1} \leq A_i$ and A_i is the maximal element, so the claim holds in the end of the iteration. If $A_i < A_{i-1}$ then there exists $k \in [1, i-1]$ such that $A_k > A_i$. Set k to be the minimal position for which the inequality holds. For convenience, denote by $A^{(j)}$ the array in the beginning of the jth iteration of the inner loop. And let's split into cases according to j value.

   (a) $j < k$ By definition of k, for any $j < k$, $A_j^{(1)} < A_i^{(1)}$, Hence in the first $k-1$ iterations no exchange will be made and we can conclude that $A_l^{(j)} = A_l^{(1)}$ for any $l \in [n]$ and $j \leq k$.

   (b) $j \geq k$ and $j \leq i$, We claim that for each such j an exchange will always occur. (The proof is given below.)

   Claim 5.1.2. *For any $j \in [k, i]$ we have that in the end of the jth iteration:*

   - $A_j^{(j+1)} = A_i^{(j)}$.
   - $A_i^{(j+1)} = A_j^{(j)} = A_j^{(1)}$.
   - *For any $l > j$ and $l \neq i$ we have $A_l^{(j+1)} = A_l^{(1)}$.*

   (c) $j > i$, so we know that $A_i^{(i+1)}$ is the maximal element in A. Therefore, for any j, it holds that $A_i^{(i+1)} \geq A_j^{(i)}$. It follows that no exchange would be made and $A_i^{(j)}$ will remain the maximum til the end of the inner loop. Thus for any $j > i$:

   $$A_i^{(j)} = A_i^{(j-1)} = .. = A_i^{(i+2)} = A_i^{(i+1)} = A_{i-1}^{(i)} = A_{i-1}^{(0)} = \max A$$

   And

   $$\begin{aligned} &A_1^{(j)}, A_2^{(j)}, ..A_{k-1}^{(j)}, A_k^{(j)}, A_{k+1}^{(j)}, ..A_{i-1}^{(j)}, A_i^{(j)}, A_{i+1}^{(j)}, A_{i+2}^{(j)}, A_{i+3}^{(j)}.. \\ =&A_1^{(0)}, A_2^{(0)}, ..A_{k-1}^{(0)}, A_i^{(0)}, A_k^{(0)}, ..A_{i-2}^{(0)}, A_{i-1}^{(0)}, A_{i+1}^{(0)}, A_{i+2}^{(0)}, A_{i+3}^{(0)}.. \end{aligned}$$

   In particular, for $j = n+1$ (Note that there is no $n+1$th iteration). Clearly, the inequalities are satisfied and we are done.

□

Proof of Claim 5.1.2. Observe that the third section holds trivially by the definition of the algorithm. It doesn't touch any position greater than j in the first j iterations (inner loop) except the ith position. So we have to prove only the first two bullets. Again, we are going to prove them by induction on j.

1. Base. $A_k^{(1)}$ is greater than A_i, and by the above case, we have that at the beginning of the kth iteration $A_k^{(k)} = A_k^{(1)}, A_i^{(k)} = A_k^{(1)}$. Therefore, the condition on line (3) is satisfied, an exchange is made, and $A_k^{(k+1)} = A_i^{(k)} = A_i^{(1)}$ and $A_i^{(k+1)} = A_k^{(k)}$.

2. Assumption. Assume the correctness of the claim for any $k \leq j' < j \leq i$.

3. Step. Consider the $j \in (k, i]$ iteration. By the induction assumption, we have that $A_{j-1}^{(j)} = A_i^{(j-1)}$ and $A_i^{(j)} = A_{j-1}^{(j-1)} = A_{j-1}^{(1)}$. On the other hand, by the induction assumption of Claim 5.1.1, $j - 1 < i \Rightarrow A_{j-1}^{(1)} \leq A_j^{(1)}$. Combining the third bullet, we obtain that:

$$A_j^{(j)} = A_j^{(1)} \geq A_{j-1}^{(1)} = A_i^{(j)}$$

And therefore, either there is an inequality and an exchange is made or there is equality. In both cases, after the jth iteration, we have $A_j^{(j+1)} = A_i^{(j)}$ and $A_i^{(j+1)} = A_j^{(j)} = A_j^{(1)}$.

$\square$

5.2 Master Theorem.

Let $\alpha > 2, \beta > 0, \gamma > 0$ satisfing $2^{\alpha} < \beta$, defining the following running time function:

$$T\left(n\right) = \beta T\left(n - \alpha\right) + n^{\gamma}$$

Bound T asymptoticly tight.

Solution. Let define $S(m) = T(\log m)$. Thus the recursive relation expand to:

$$\begin{aligned}S\left(m\right) &= T\left(\log m\right) = \beta T\left(\log m - \alpha\right) + \log^{\gamma}(m) = \beta T\left(\log m - \log 2^{\alpha}\right) + \log^{\gamma}(m) \\ &= \beta T\left(\log\left(\frac{m}{2^{\alpha}}\right)\right) + \log^{\gamma}(m) = \beta S(\frac{m}{2^{\alpha}}) + \log^{\gamma}(m)\end{aligned}$$

Observes that $2^{\alpha} > 1$ and therefore we can use the generalized master theorem. By the given that $2^{\alpha} < \beta$ we have that $m^{\log_{2^{\alpha}}(\beta)}$ is (generalized) polynom with positive degree. And therefore there is a positvie ε such that $\log^{\gamma}(m) = O\left(m^{\log_{2^{\alpha}}(\beta) - \varepsilon}\right)$. Hence by the master theorem, we conclude that:

$$S(m) = \Theta\left(m^{\log_{2^{\alpha}}(\beta)}\right) \Longrightarrow T(n) = \Theta\left(2^{m \log_{2^{\alpha}}(\beta)}\right)$$

Bibliography

[Fun21] Stanley P. Y. Fung. "Is this the simplest (and most surprising) sorting algorithm ever?" In: *CoRR* abs/2110.01111 (2021). arXiv: 2110.01111. URL: https://arxiv.org/abs/2110.01111.