Lab02-Divide and Conquer

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2020.

$*$ If there is any problem, please contact TA Yiming Liu.

$*$ Name: Futao Wei Student ID: 518021910750 Email: weifutao@sjtu.edu.cn

1. **Quicksort** is based on the Divide-and-Conquer method. Here is the two-step divide-and-conquer process for sorting a typical subarray $A[p \dots r]$:

   (a) **Divide:** Partition the array $A[p \dots r]$ into two subarrays $A[p \dots q-1]$ and $A[q+1 \dots r]$ such that each element of $A[p \dots q-1]$ is less than or equal to $A[q]$, which is, in turn, less than or equal to each element of $A[q+1 \dots r]$. Compute the index q as part of this partitioning procedure.

   (b) **Conquer:** Sort $A[p \dots q-1]$ and $A[q+1 \dots r]$ respectively by recursive calls to Quicksort.

   Write down the recurrence function $T(n)$ of QuickSort and compute its time complexity.

   Hint: At this time $T(n)$ is split into two subarrays with different sizes (usually), and you need to describe its recurrence relation by the sum of two subfunctions plus additional operations.

   Solution.

   $$T(n) = T(i-1) + T(n-i) + O(n),$$

   where i is the final position of the *pivot* chosen. $O(n)$ stands for the time complexity to determine the *pivot's* final position.
   The worst case happens when the *pivots* chosen always end up at either ends of the array, that is, $i = n$ or $i = 1$. For the sake of simplicity, assume $i = n$ all the time.
   Hence we have

   $$\begin{aligned} T(n) &= T(n-1) + O(n) \\ &= T(n-2) + O(n-1) + O(n) \\ &\dots \\ &= T(0) + O(1) + O(2) + \cdots + O(n) \\ &= O(n^2) \end{aligned}$$

   □

2. **MergeCount**. Given an integer array $A[1 \dots n]$ and two integer thresholds $t_l \le t_u$, Lucien designed an algorithm using divide-and-conquer method (As shown in Alg. 1) to count the number of ranges (i, j) $(1 \le i \le j \le n)$ satisfying

   $$t_l \le \sum_{k=i}^{j} A[k] \le t_u. \tag{1}$$

   Before computation, he firstly constructed $S[0 \dots n+1]$, where $S[i]$ denotes the sum of the first i elements of $A[1 \dots n]$. Initially, set $S[0] = S[n+1] = 0$, $low = 0$, $high = n+1$.

Algorithm 1: MergeCount(S, t_l, t_u, low, $high$)

Input: $S[0, \cdots, n+1]$, t_l, t_u, low, $high$.
Output: $count$ = number of ranges satisfying Eqn. (1).

$count \leftarrow 0$; $mid \leftarrow \lfloor \frac{low+high}{2} \rfloor$;
if $mid = low$ **then** **return** 0 ;
$count \leftarrow MergeCount(S, t_l, t_u, low, mid) + MergeCount(S, t_l, t_u, mid, high)$;
for $i = low$ **to** $mid - 1$ **do**

$$m \leftarrow \begin{cases} \min\{m \mid S[m] - S[i] \geq t_l, m \in [mid, high-1]\}, & \text{if exists} \\ high, & \text{if not exist} \end{cases};$$

$$n \leftarrow \begin{cases} \min\{n \mid S[n] - S[i] > t_u, n \in [mid, high-1]\}, & \text{if exists} \\ high, & \text{if not exist} \end{cases};$$

// BinarySearch is used to find m, n
$count \leftarrow count + n - m$;

$Merge(S, low, mid-1, high-1)$; // Merge is used for two sorted arrays
return $count$;

Example: Given $A = [1, -1, 2]$, $lower = 1$, $upper = 2$, return 4. The resulting four ranges should be $(1,1)$, $(1,3)$, $(2,3)$, and $(3,3)$.

Is Lucien's algorithm correct? Explain his idea and make correction if needed. Besides, compute the running time of Alg. 1 (or the corrected version) by recurrence relation. (Note: we can't implement Master's Theorem in this case. Refer Reference06 for more details.)

Solution.
Correct. The counting of size n can be divided into two same-type sub-problems of size $\frac{n}{2}$ and the work of examining ranges which cover the division *mid*, which is done by the *for* loop. Moreover, a *Merge* is conducted so that m and n can be obtained more efficiently.
We can derive a guess of the time complexity $T(n)$ by the recursion-tree method:

$$\begin{aligned} T(n) &= \sum_{k=0}^{\log_2 n} cn \log_2 \frac{n}{2^k} \\ &= cn \log_2 \prod_{k=0}^{\log_2 n} \frac{n}{2^k} \\ &= cn \log_2 n^{\log_2 n + 1} (2 - \frac{1}{n}) \\ &\sim O(n \log^2 n) \end{aligned}$$

where k is the depth in the tree.
Now we can use the substitution method to verify that our guess was correct, that is, $T(n) = O(n \log^2 n)$ is an upper bound for the recurrence $T(n) = 2T(\frac{n}{2}) + O(n \log n)$. We want to show that $T(n) \leq dn \log_2^2 n$ for some constant $d > 0$. Using the same constant $c > 0$ as before, we have

$$
\begin{aligned}
T(n) &= 2T(\frac{n}{2}) + O(n \log n) \\
&\leq 2d \cdot \frac{n}{2} \cdot \log_2^2 \frac{n}{2} + cn \log_2 n \\
&= dn \log_2^2 \frac{n}{2} + cn \log_2 n \\
&\leq dn \log_2^2 n
\end{aligned}
$$

where the last step holds as long as $d \geq \lambda c$ (λ is a constant satisfying $\lambda \geq \frac{\log_2 n}{\log_2^2 n - \log_2^2 \frac{n}{2}}$ for $n \geq 2$). $\square$

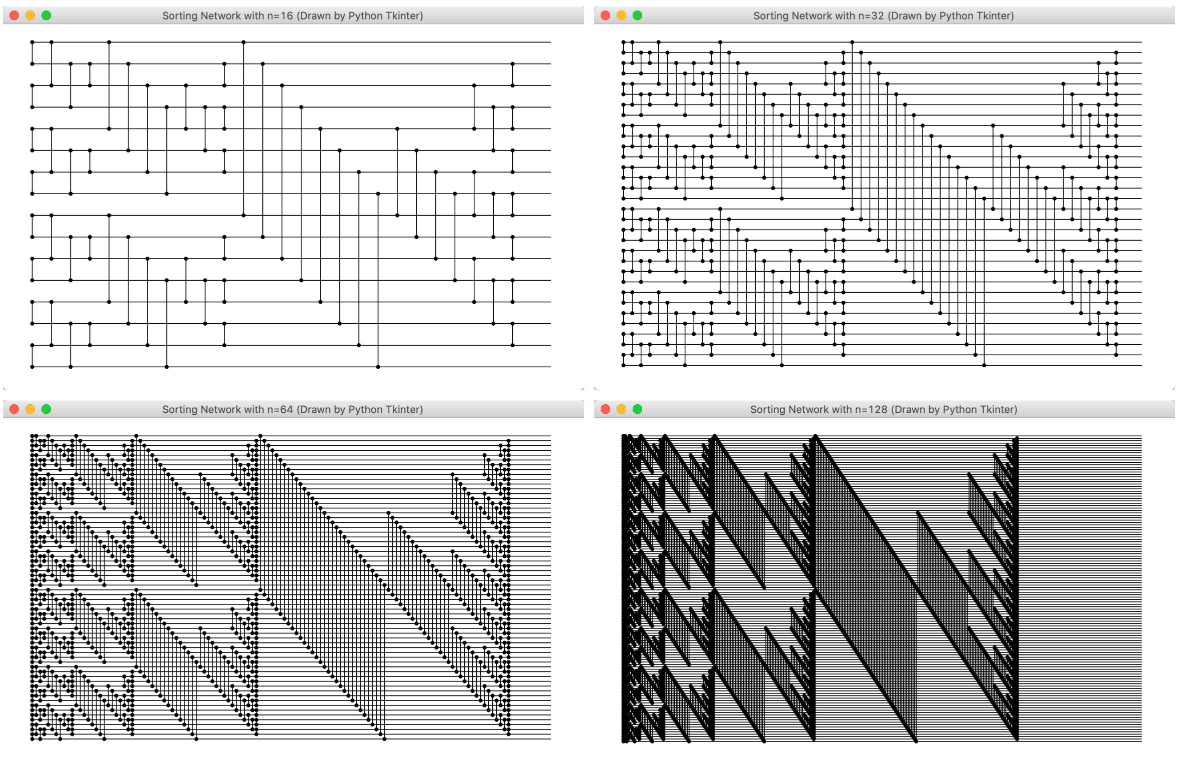
3. **Batcher's odd-even merging network.** In this problem, we shall construct an ***odd-even merging network***. We assume that n is an exact power of 2, and we wish to merge the sorted sequence of elements on lines $\langle a_1, a_2, \ldots, a_n \rangle$ with those on lines $\langle a_{n+1}, a_{n+2}, \ldots, a_{2n} \rangle$. If $n = 1$, we put a comparator between lines a_1 and a_2. Otherwise, we recursively construct two odd-even merging networks that operate in parallel. The first merges the sequence on lines $\langle a_1, a_3, \ldots, a_{n-1} \rangle$ with the sequence on lines $\langle a_{n+1}, a_{n+3}, \ldots, a_{2n-1} \rangle$ (the odd elements). The second merges $\langle a_2, a_4, \ldots, a_n \rangle$ with $\langle a_{n+2}, a_{n+4}, \ldots\ a_{2n} \rangle$ (the even elements). To combine the two sorted subsequences, we put a comparator between a_{2i} and a_{2i+1} for $i = 1, 2, \ldots, n-1$.

   (a) Replace the original Merger (taught in class) with Batcher's new Merger, and draw $2n$-input sorting networks for $n = 8, 16, 32, 64$. (Note: you are not forced to use Python Tkinter. Any visualization tool is welcome for this question.)

   (b) What is the depth of a $2n$-input odd-even sorting network?

   (c) (Optional Sub-question with Bonus) Use the zero-one principle to prove that any $2n$-input odd-even merging network is indeed a merging network.

Solution.

(a) 

(b) The depth of Sorter[$2n$] is given by the recurrence:

$$D(2n) = \begin{cases} 1 & \text{for } n = 1 \\ D(n) + \log_2 2n & \text{for } n = 2^k \text{ and } k \geq 1 \end{cases}$$

whose solution is $D(2n) = \Theta(\log^2 n)$.

(c) Firstly, we'll prove that the merging network works fine for 0-1 sequences.
Suppose that $\langle a_1, a_2, \ldots, a_n \rangle$ starts with m_1 0's, $\langle a_{n+1}, a_{n+2}, \ldots, a_{2n} \rangle$ starts with m_2 0's. Then $\langle a_1, a_3, \ldots, a_{2n-1} \rangle$ has $\lceil \frac{m_1}{2} \rceil + \lceil \frac{m_2}{2} \rceil$ 0's, while $\langle a_2, a_4, \ldots, a_{2n} \rangle$ has $\lfloor \frac{m_1}{2} \rfloor + \lfloor \frac{m_2}{2} \rfloor$ 0's. Hence the odd sequence has $0, 1,$ or 2 more 0's than the even one.
After merging on both the odd and even sequences, 0's go up while 1's go down. If the odd sequence has 0 or 1 more 0's than the even one, the whole sequence is already merged. If the odd sequence had 2 more 0's than the even one, the comparators between a_{2i} and a_{2i+1} $(i = 1, 2, \ldots, n-1)$ will fix the problem, i.e., exchanging the 1 of the even sequence and the 0 of the odd sequence right below it.
In conclusion, the merging network is correct for 0-1 sequences. As ***Zero-one principle*** indicates, the network sorts all sequences of arbitrary numbers correctly.

$\square$

Remark: You need to include your .pdf, .tex and .py files (or other possible sources) in your uploaded .rar or .zip file.