Lab02-Divide and Conquer

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

∗ If there is any problem, please contact TA Yiming Liu.
∗ Name:Haotian Xue Student ID:518021910506 Email: xavihart@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.

   Proof. For the **Best Case**, each time the array is divided into two subsarrays that have the same length. So $T(n)$ can be recursively calcualted as:

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

   according to the master method, $\log_2 2 = 1 = d$, so we can get $T(n) = n \log n$

   For the **Worst Case**, each time the array is divided into a array lengthed one and another lenghted $n-2$.

$$T(n) = T(n-2) + O(n)$$

   So by recursion we can get:(the % means mod operation)

$$T(n) = O(n) + O(n-2) + O(n-4) + ... + O(n\%2+2) + T(n\%2) = O(n^2)$$

   At last, for the **Average Case**, in each process of dividing, we assume the probability of each dividing method to be the same, so:

$$T(n) = O(n) + \sum_{j=0}^{n-1} \frac{1}{n}(T(j) + T(n-j-1))$$

   So we can get $T(n+1)$ use the above equation:

$$(n+1)T(n+1) = (n+1)^2 + 2(T(0) + T(1) + ... + T(n))$$

   After subtracting $T(n+1)$ by $T(n)$ and dividing the both side of the equation by $(n+2)(n+1)$:

$$\frac{T(n+1)}{n+2} = \frac{T(n)}{n+1} + \frac{2n+1}{(n+1)(n+2)}$$

$$\frac{T(n)}{n+1} = \sum_{j=1}^{n-1} \frac{2(j)}{(j+1)(j+2)} + T(1)$$

When n is big enough, the fraction in the sum can be approximated as the **Harmonic Series**. So the sum is growing like $2ln(n)$. By this we can get $T(n) = O(n \log n)$

□

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

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

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).

1 $count \leftarrow 0$; $mid \leftarrow \lfloor \frac{low+high}{2} \rfloor$;
2 **if** $mid = low$ **then return** 0 ;
3 $count \leftarrow MergeCount(S, t_l, t_u, low, mid) + MergeCount(S, t_l, t_u, mid, high)$;
4 **for** $i = low$ ***to*** $mid - 1$ **do**
5 $\quad m \leftarrow \begin{cases} \min\{m \mid S[m] - S[i] \geq t_l, m \in [mid, high-1]\}, & \text{if exists} \\ high, & \text{if not exist} \end{cases}$;
6 $\quad n \leftarrow \begin{cases} \min\{n \mid S[n] - S[i] > t_u, n \in [mid, high-1]\}, & \text{if exists} \\ high, & \text{if not exist} \end{cases}$;
$\quad$ // BinarySearch is used to find m, n
7 $\quad count \leftarrow count + n - m$;
8 $Merge(S, low, mid-1, high-1)$; // Merge is used for two sorted arrays
9 **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.)

Brief explanation

Divide : Given an array A, first we do some preprocess work like calcualting S in $O(n)$, then we divide A into two sections by the middle element.

After that we recursively call the function to find the (i, j) pairs in the two subarrays. It should be noted that the array two subarrays are actually sorted! Because in the end of each recursion we call the $Merge()$ function, which can be seemed to do the same work int the Merging Sort Algorithm from bottom to top.

So we can use BinarySearch to find the satisfying indexes in line 5 and 6 in $O(\log n)$.

Conquer: Then we just add the number of pairs in the two subarrays and the pairs which cross the middle element to get the final answer. In the end of the function we use merge method to keep the array sorted.

Proof. From the analysis we can get the recurrence relation:

$$T(n) = 2T(n/2) + 2 \cdot \frac{n}{2}\log(\frac{n}{2}) + \frac{n}{2}$$

we still cannotuse the Master Method, so we just list the whole equation($k = \log n, 2^k = n$):

$$T(n) = \sum_{j=1}^{k}(\frac{n}{2^j} + \frac{n}{2^{j-1}}\log\frac{n}{2^j}) + 2^k \cdot T(1)$$

the above summation can be divided into three parts:

$$\begin{cases} \sum_{j=1}^{k} \frac{n}{2^j} = 2n \cdot (1 - \frac{1}{n}) = O(n) \\ \sum_{j=1}^{k} \frac{n}{2^{j-1}}(\log n - j) = O(n \log n) \\ 2^k \cdot T(1) = O(2^k) = O(n) \end{cases}$$

So actually $T(n) = O(n \log n)$.

□

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.

Proof. (a) The figure for $2n = 16, 32, 64, 128$ can be referred to in Fig1, which are all drawn by Tkinter.

(b) First we let $n = 2^k$, then we define the depth of a n-input odd-even sorting network $S(n)$ and the depth of a n-input batcher's merging network $M(n)$.

As can be seen in Fig2, we can get the recurrence relation:

$$S[2^k] = S[2^{k-1}] + M[2^k]$$

$$M[2^k] = 2M[2^{k-1}] + 1$$

So it is easy to get: $M[2^k] = 2^k - 1$, and $S[2^k] = 2^{k+1} - k - 2$. So for 2n-input, we can get:

$$S[2n] = 4n - \log n - 3$$

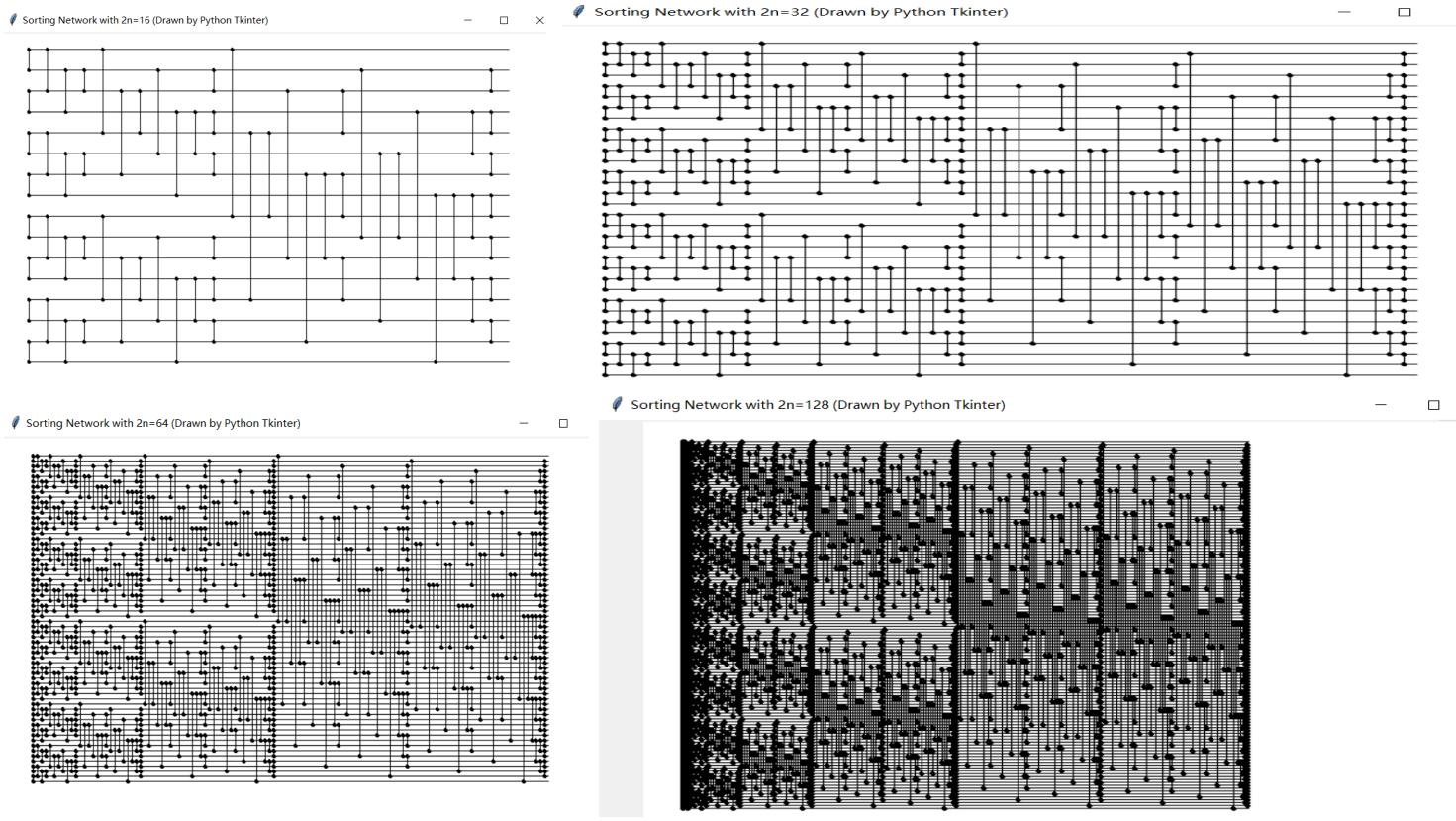

Fig. 1. Fig1

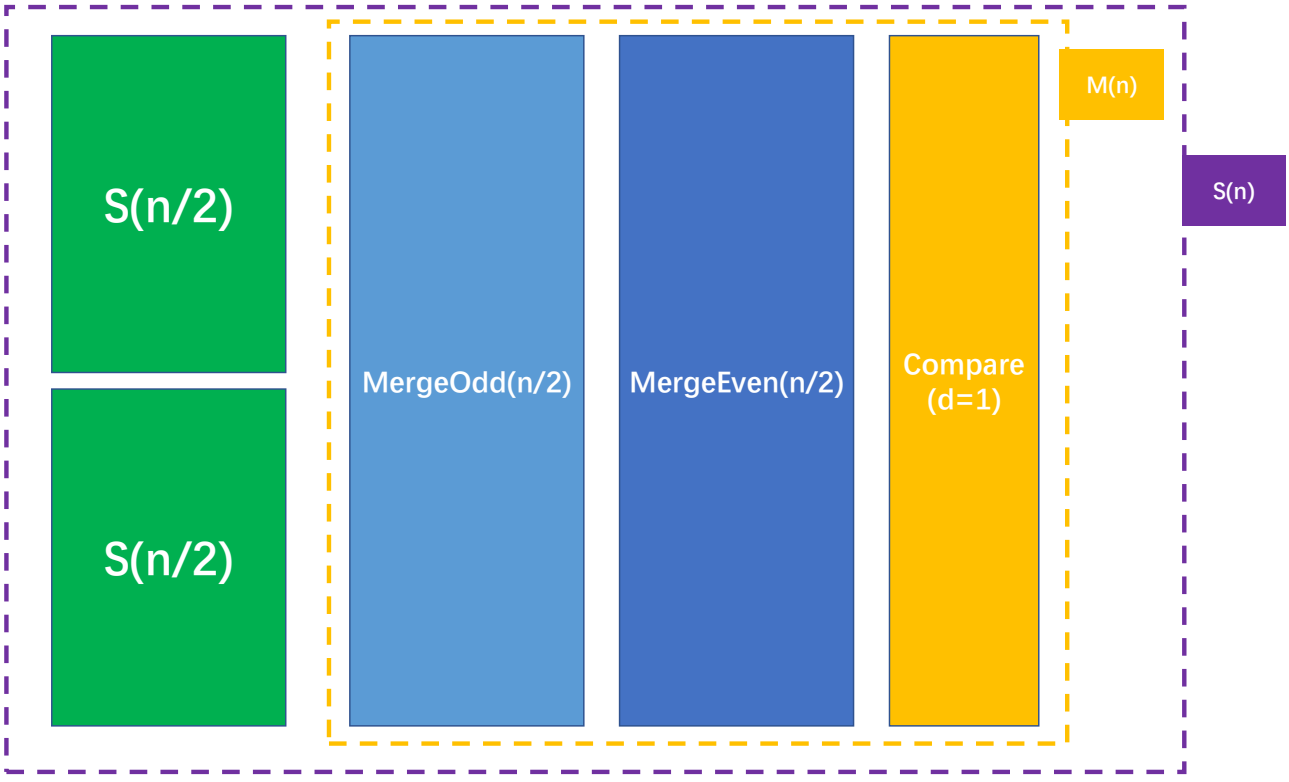

Fig. 2. Fig1

(c) The merging operation can be divided into three parts: first we input two sored arrays, in the zero-one principle proof $< 0, 0, ..., 0, 1, ..., 1 >$, then we sorted the odd and even subarray respectively, at the end we check each a_{2i} and a_{2i+1} as is illustrated in the context.

Actually I want to prove the correctness of the above operations by proving a more general lemma which is:

Lemma 1. *Given two sorted arrays* $< a_1, a_2, ..., a_n >$ *and* $< b_1, b_2, ..., b_n >$*, we respectively sort* $< a_1, a_3, ..., a_{n-1}, b_1, b_3, ..., b_{n-1} >$ *and* $< a_2, a_4, ..., a_n, b_2, b_4, ..., b_n >$*. Then we can get a merged array* $< a_1^{'}, a_2^{'}, a_n^{'}, b_1^{'}, b_2^{'}, ..., b_n^{'} >$*. For simplicity we name it* $< c_1, c_2, ..., c_{2n} >$*, where* $n = 2k, k \in N$ *, then we have:*

$$
\begin{gathered}
c_1 < min\{c_2, c_3\} \\
min\{c_{2n-2}, c_{2n-1}\} < c_{2n} \\
max\{c_{2k}, c_{2k+1}\} < min\{c_{2k+2}, c_{2k+3}\} (k \in \{1, 2, ..., n-2\})
\end{gathered}
$$

The lemma actually give an insght into why we should exchange a_{2i} and a_{2i+1}. We can prove it by **Mathematical Induction**:

First for $n = 2$, since $c_1 < c_3$, $c_1 = min\{a1, b1\}$ and $c_2 = min\{a2, b2\}$, we can get $c_1 < min\{c_2, c_3\}$, samely: $c_4 > max\{c2, c3\}$.

Then when $n = 2k + 2$, by induction we can get:

$$
\begin{gathered}
c_1 < min\{c_2, c_3\} \\
min\{c_{2k-2}, c_{2k-1}\} < c_{2k} \\
max\{c_{2j}, c_{2j+1}\} < min\{c_{2j+2}, c_{2j+3}\} (j \in \{1, 2, ..., n-2\})
\end{gathered}
$$

we just need to prove that

□

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