

# Lab02-Divide and Conquer

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

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

\* Name: Yijia Diao(刁义嘉) Student ID: 518030910146 Email: diao-yijia@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.** When  $q = k, 1 \leq k \leq n$ , and since the time complexity of divide is  $O(n)$  (proofed in Lab01), the recurrence function of QuickSort is

$$T(n) = T(k-1) + T(n-k) + O(n)$$

Now compute its time complexity by cases:

- (1) **Best case:** the recurrence function is:

$$T(n) = 2T(\lceil n/2 \rceil) + O(n)$$

According to Master Theorem, since  $d = 1 = \log_2 2 = \log_b a$ ,  $T(n) = O(n \log n)$ .

- (2) **Worst case:** the recurrence function is:

$$T(n) = T(0) + T(n-1) + O(n)$$

After accumulation,  $T(n) = O(\frac{(n+2)(n-1)}{2}) = O(n^2)$

- (3) **Average case:** according to the definition of average time complexity, the recurrence function is:

$$T(n) = \sum_{k=1}^n \Pr(i=k)(O(n) + T(i-1) + T(n-i))$$

The Master Theorem does not apply to this case. According to Lab01 Problem1,  $T(n) = O(n \log n)$ .

□

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

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

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:**  $S[0, \dots, n + 1], t_l, t_u, low, high$ .

**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    $m \leftarrow \begin{cases} \min\{m \mid S[m] - S[i] \geq t_l, m \in [mid, high - 1]\}, & \text{if exists} \\ high, & \text{if not exist} \end{cases}$  ;
6    $n \leftarrow \begin{cases} \min\{n \mid S[n] - S[i] > t_u, n \in [mid, high - 1]\}, & \text{if exists} \\ high, & \text{if not exist} \end{cases}$  ;
   // BinarySearch is used to find  $m, n$ 
7    $count \leftarrow count + n - m$ ;
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.)

**Solution.** Lucien's algorithm is **not correct**. The error is in Line 2; the correction should be:

**if**  $mid = high$  **then return** 0 ;

**Idea:** Alg. 1 uses top-down Divide-and-Conquer in general.

It divides input array in half, and do counting and merging in both sub-arrays; the merge operation is equivalent to Mergesort in sub-arrays.

After returning to the recursive call function, it make use of the ordered state of the latter sub-array, to count between the former and latter sub-array; add this result and sub-arrays' counting number, then get the counting number of input array. Before returning, finish the merge step so that input array would be sorted.

Suppose the time complexity of corrected Alg. 1 is  $T(n)$ .

Between line 4 and line 7, the time complexity of the **best case** is  $\frac{n}{2} \cdot 2 \cdot 1 = O(n)$ , the **worst and average case** is  $\frac{n}{2} \cdot 2 \log \frac{n}{2} = O(n \log n)$ ; the time complexity of Merge is  $O(n)$ . Therefore, we compute  $T(n)$  by case:

(1) **Best case:** the recurrence function is:

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

According to Master Theorem, since  $1 = \log_2 2$ ,  $T(n) = O(n \log n)$ .

(2) **Worst and Average case:** the recurrence function is:

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

Then we do iteration:

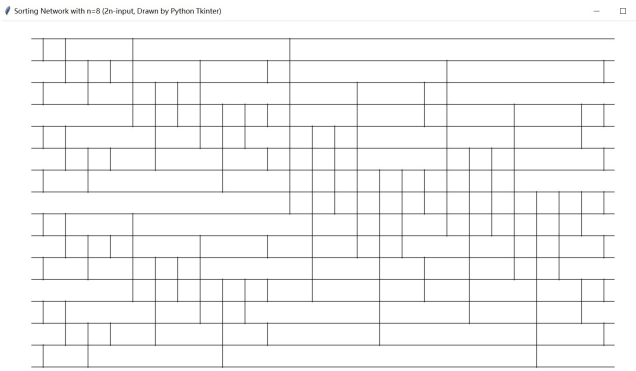
$$\begin{aligned}
T(n) &= 2(2T(\lceil \frac{n}{4} \rceil) + O(\frac{n}{2} \log \frac{n}{2})) + O(n \log n) = 2^2 T(\lceil \frac{n}{4} \rceil) + 2^1 O(\frac{n}{2} \log \frac{n}{2}) + 2^0 O(n \log n) \\
&= \dots \\
&= \sum_{i=0}^k 2^{k-i} \cdot O(2^i \cdot i), k = \log n
\end{aligned}$$

So we have  $T(n) = O(2^k \cdot k^2) = O(n \log^2 n)$ .

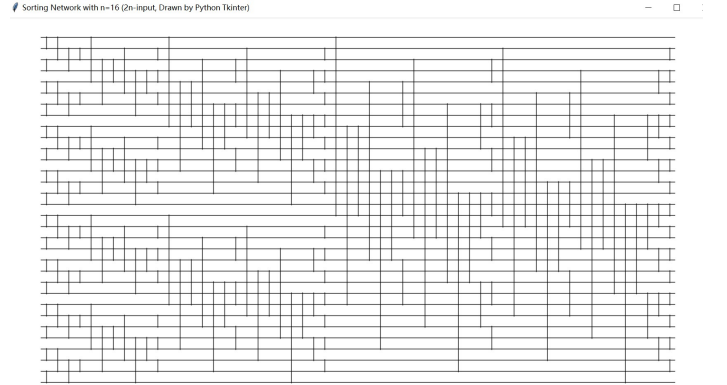
We can conclude that the time complexity of Alg. 1 is  $O(n \log^2 n)$ .  $\square$

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  $\langle a_1, a_2, \dots, a_n \rangle$  with those on lines  $\langle a_{n+1}, a_{n+2}, \dots, a_{2n} \rangle$ . 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  $\langle a_1, a_3, \dots, a_{n-1} \rangle$  with the sequence on lines  $\langle a_{n+1}, a_{n+3}, \dots, a_{2n-1} \rangle$  (the odd elements). The second merges  $\langle a_2, a_4, \dots, a_n \rangle$  with  $\langle a_{n+2}, a_{n+4}, \dots, a_{2n} \rangle$  (the even elements). To combine the two sorted subsequences, we put a comparator between  $a_{2i}$  and  $a_{2i+1}$  for  $i = 1, 2, \dots, n-1$ .

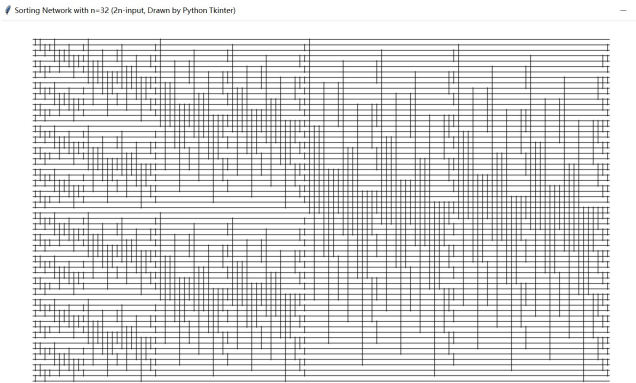
- (a) Replace the original Merger (taught in class) with Batcher's new Merger, and draw  $2n$ -input sorting networks for  $n = 8, 16, 32, 64$ . (Note: you are not forced to use Python Tkinter. Any visualization tool is welcome for this question.)



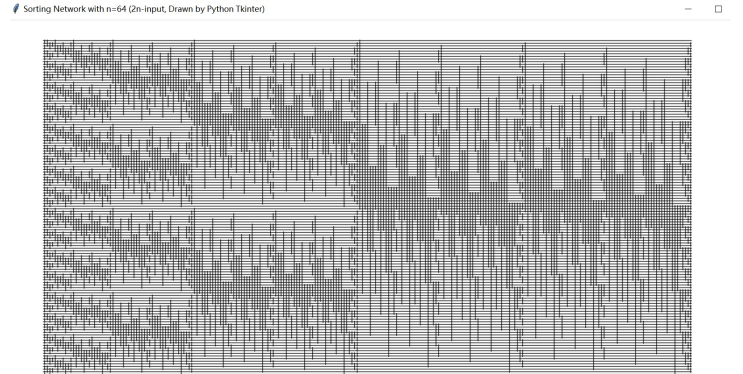
**Fig.1**  $n = 8$



**Fig.2**  $n = 16$



**Fig.3**  $n = 32$



**Fig.4**  $n = 64$

\* The [source code](#) lent part of the python code on the course website.

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

**Solution.** Suppose the depth of odd-even merge operation is  $M(n)$ , the depth of the network is  $D(n)$ . Then we have:

$$M(n) = \begin{cases} 1, & \text{if } n = 1; \\ M(\frac{n}{2}) + 1, & \text{if } n > 1; \end{cases}$$

According to Master Theorem,  $M(n) = O(\log n)$ . Since  $D(n) = D(\frac{n}{2}) + M(n)$ ,

$$\begin{aligned} D(n) &= D(\frac{n}{4}) + O(\log \frac{n}{2}) + O(\log n) \\ &= \dots \\ &= \sum_{i=0}^k O(i), k = \log n \\ &= O(k^2) \\ &= O(\log^2 n) \end{aligned}$$

We can conclude that the depth of Batcher's odd-even merging network is  $O(\log^2 n)$ . □

(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.

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