# Lab02-Divide and Conquer

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

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

∗ Name:Shengyuan Hou   Student ID:518021910604   Email:445164577@qq.com

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.** Randomly select the pivot in the subarray. Since dividing the subarray into two parts which each element on the left is less than or equal to pivot and each element on the right is more than or equal to pivot at least takes up n-1 comparisons(n is the size of the subarray).So the time complexity of partitioning is O(n).The position of pivot is randomized. So the recurrence function T(n) of Quicksort is:

$$T(n) = T(a) + T(b) + n - 1 \quad (a + b + 1 = n, \quad a, b \geq 0)$$

   When $a = 0$,$b = n - 1$, it is the worst case that

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

$$T(0) = T(1) = 0$$

   Therefore, we get the closed form of the recursive function.

$$T(n) = T(0) + \sum_{i=0}^{n-1} i = \frac{n \times (n-1)}{2} = O(n^2)$$

   □

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       $m \leftarrow \begin{cases} \min\{m \mid S[m] - S[i] \geq t_l, m \in [mid, high-1]\}, & \text{if exists} \\ high, & \text{if not exist} \end{cases}$;
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.** Yes,it is correct.
**Explanation:** Lucien's idea can be described as follows. First, he makes use of Prefix Sum to rapidly calculate the sum of any range.

$$S[i] = \sum_{k=0}^{i} A[k]$$

$$\sum_{k=i}^{j} A[k] = S[j] - S[i-1] (1 \leq i \leq j \leq n)$$

Therefore the problem can be transformed as counting the number of $(i, j)$ pair which satisfies

$$t_l \leq S[j] - S[i-1] \leq t_u$$

Then Lucien's algorithm partition the array by the middle and divides the (i,j) pair into three categories:

(a) $S[i], S[j]$ are both on the left part

(b) $S[i], S[j]$ are both on the right part

(c) $S[i]$ is on the left part and $S[j]$ is on the right part

Since if the number of inversed pairs in condition (a) and (b) is true, the sequential change of S in the left and right parts will not influence the final result because all relational position of pair in condition (c) don't change (all $j$ on the right part are still greater than $i-1$ on the left part).
The number of the first two kinds of pairs can be calculated by calling recursive function,

while the rest one can be counted by fixing every index $i$ in the left part and determining the lower and upper bound of the right index $j$ by binary research.When the left index is fixed

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

$$t_l + S[i-1] \leq S[j] \leq t_u + S[i-1]$$

we could find the upper bound which is the minimum one that is greater than $t_u + S[i-1]$ and the lower bound which is the minimum one that is greater than or equal to $t_l + S[i-1]$ by binary search.
Finally, sum up the counts of three conditons and return the result.
**Running time:**
We divide the function into three parts:recurrence, iteration of binary search and merge.
Time complexity of recurrence is $2T(\frac{n}{2})$.
Time complexity of iteration is $2 \times \frac{n}{2} \times \log \frac{n}{2} = O(n \log n)$.
Time complexity of merge is $O(n)$.
So the recursive equation is

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

**Lemma 1.** *$\forall n \in N, \exists n'$ with $n \leq n' \leq 2n$ such that $n'$ is a power of* 2

According to the lemma, assume for convenience that $n$ is a power of 2, and $n = 2^k$. The recursive function can be described as a binary tree with $k+1$ layers. For the layer $i$, there are $2^{i-1}$ nodes, and every node has a time consumption of $\frac{n}{2^{i-1}} \log \frac{n}{2^{i-1}}$. So the total time consumption is

$$\sum_{i=1}^{k+1} 2^{i-1} \frac{n}{2^{i-1}} \log \frac{n}{2^{i-1}} = \sum_{i=1}^{k+1} n \log \frac{n}{2^{i-1}} = n \log \frac{n^{k+1}}{2^{\frac{k(k+1)}{2}}} = \frac{k+1}{2} n \log n = \frac{\log n + 1}{2} n \log n = O(n \log^2 n)$$

□

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.

**Solution.**

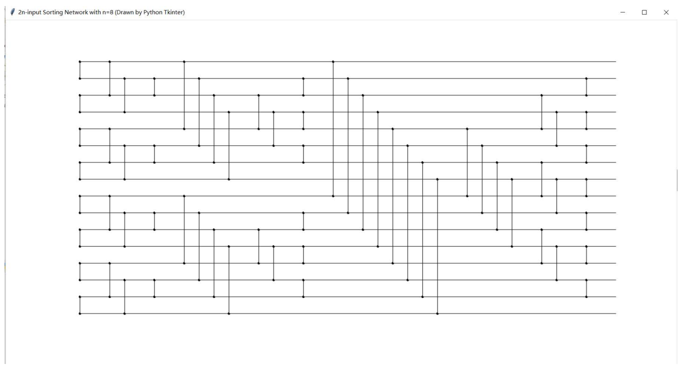


图 1: $n = 8, 2n = 16$

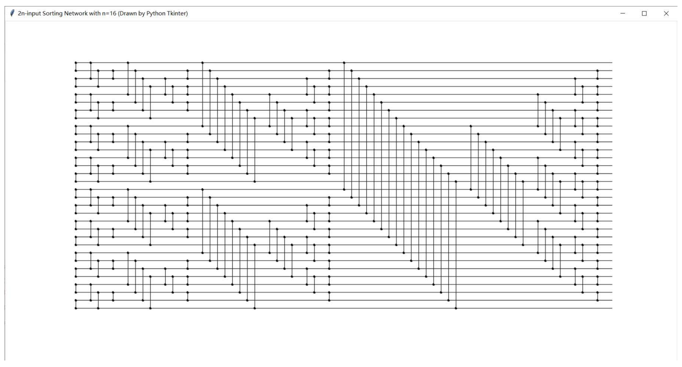

2n-input Sorting Network with n=16 (Drawn by Python Tkinter)

图 2: $n = 16, 2n = 32$

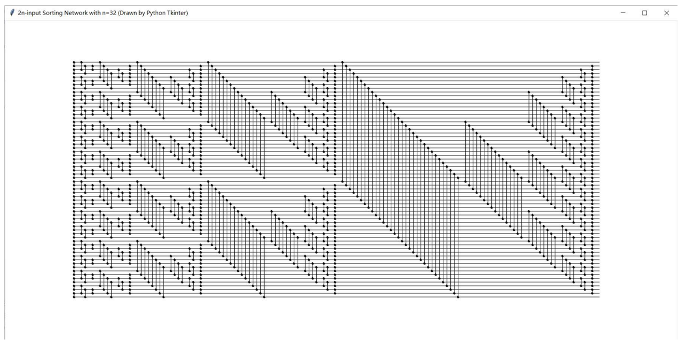

2n-input Sorting Network with n=32 (Drawn by Python Tkinter)

图 3: $n = 32, 2n = 64$

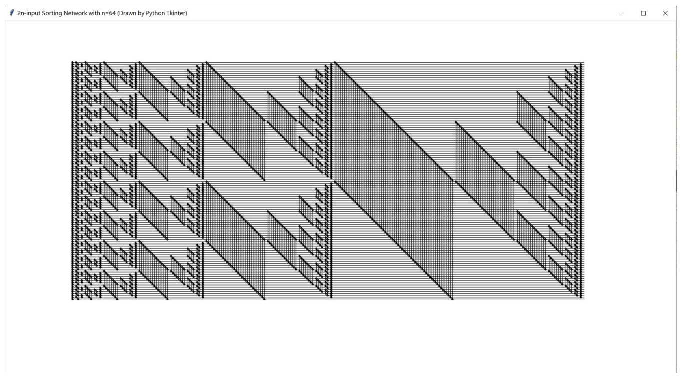

2n-input Sorting Network with n=64 (Drawn by Python Tkinter)

图 4: $n = 64, 2n = 128$

a Attention please! Since we are asked to draw $2n$-input sorting networks rather than n-inputs. The actual number of horizental lines is $2n$.

b Since the $2n$-input odd-even sorting network can de divied into two same subsorters and a merger. The depth of it can be calculated as a sum of the subsorter depth and merger depth. Let $T(2n)$ be the depth of $2n$-input odd-even sorting network.
For the merger part, by inspection we could find that the i-th layer of comparison is between numbers which have a difference value of $\frac{2n}{2^i}$ ,and the $i$ is up to $\log_2(2n)$. And in every layer the depth of the network could be added 1, so the total layer of merger part is $\log_2(2n)$.
So we can write the recursive equation.

$$T(2n) = T(n) + \log_2(2n)$$

Since $n = 2^k (k > 0)$, $T(2) = 1$, $T(1) = 0$.

$$T(2n) = T(n) + \log_2(2n) \tag{2}$$
$$T(n) = T(n/2) + \log_2(n) \tag{3}$$
$$\ldots \tag{4}$$
$$T(2) = T(1) + \log_2(2) \tag{5}$$

Add up all the equations, we can get the result.

$$T(2n) = \frac{(k+2)(k+1)}{2} = \frac{(\log_2 n + 2)(\log_2 n + 1)}{2} = O(\log^2 n)$$

c

**Theorem 2.** *If a comparison network with n inputs sorts all $2^n$ possible sequences of 0's and 1's correctly, then it sorts all sequences of arbitrary numbers correctly.*

We use the principle of induction to prove that any odd-even merging network with 2n-inputs of $0-1$ sequence is indeed a merging network.
**Basis Step.**
When $n = 1$, apparently the merging process of 2 ordered one-element 0-1 sequences could work successfully with a comparator.
**Induction Hypothesis.**
Suppose that for $n = 2^k$, the merging process of two ordered 0-1 sequences both with $2^k$ elements $(a_1, a_2, ..., a_{2^k})$ $(a_{2^k+1}, ..., a_{2^{k+1}})$ could be merged into one ordered sequence nondecreasingly after the merging process as described in the problem.
**Proof of Induction Step.**
Then for $n = 2^{k+1}$, before merging there are two ordered inputs of 0-1 sequences, which is signed as

$$S_1 :< a_1, a_2, ..., a_{2^{k+1}} >$$

$$S_2 :< a_{2^{k+1}+1}, a_{2^{k+1}+2}, ..., a_{2^{k+2}} >$$

As described in the question , we should merge two groups of sequences.

$$S_3 :< a1, a_3, ..., a_{2^{k+1}-1} > \quad S_4 :< a_{2^{k+1}+1}, a_{2^{k+1}+3}, ..., a_{2^{k+2}-1} >$$

$$S_5 :< a_2, a_4, ..., a_{2^{k+1}} > \quad S_6 :< a_{2^{k+1}+2}, a_{2^{k+1}+4}, ..., a_{2^{k+2}} >$$

. By induction hypothesis, the two merged sequece with $2 * 2^k$ elements is ordered, which can be formulated as $0^i 1^j$ and $0^p 1^q$.

The sequence $< a_1, a_2, ..., a_{2^{k+1}} >$ and $< a_{2^{k+1}+1}, a_{2^{k+1}+2}, ..., a_{2^{k+2}} >$ can be also formulated as $0^c 1^d$.
Define some variants as follows

$$u_o : number\ of\ zeros\ which\ have\ oven\ index\ in\ sequence\ S_1$$

$$u_e : number\ of\ zeros\ which\ have\ even\ index\ in\ sequence\ S_1$$

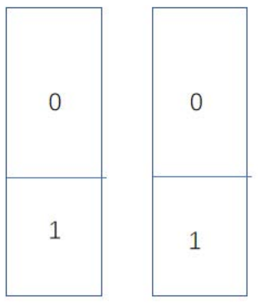
$$l_o : number\ of\ zeros\ which\ have\ oven\ index\ in\ sequence\ S_2$$

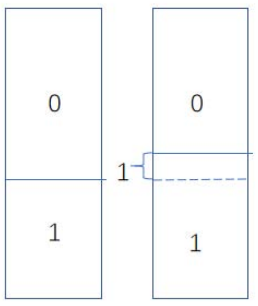
$$l_e : number\ of\ zeros\ which\ have\ even\ index\ in\ sequence\ S_2$$

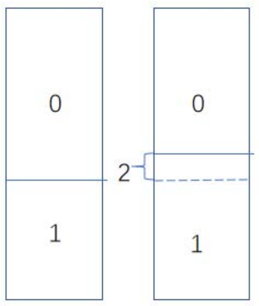
$$t_o : u_o + l_o$$

$$t_e : u_e + l_e$$

Since $S_1$ and $S_2$ are both ordered, $|u_o - u_e| \le 1$,$|l_o - l_e| \le 1$. So we get $|t_o - t_e| \le 2$. Therefore after merging $S_3, S_4$ and $S_5, S_6$, there are three possibilities.
In the first two conditions, the overall sequence is 00...011...11,and after adjacent sorting except the endpoint it remains stable.
In the third condition, the overall sequence is 00...01011...11. So after adjacent sorting except the endpoint, 00...001011...11 is transformed as 00...011...11, which is ordered, thus statement is proved.
According to zero-one principle, any sequence of $n = 2^k$ elements could be sorted correctly and any odd-even merging network with 2n-inputs of $0 - 1$ sequence is indeed a merging network.







$\square$

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