# Lab02-Divide and Conquer

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

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

∗ Name: ShaoxiongLin   Student ID: 517010910028   Email: Johnson-Lin@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.** In worst case, the Quicksort algorithm would partition the array into two parts with $n-1$ elements and 0 elements, and the partitioning of an array with $n$ elements takes $n-1$ comparisons. Thus, the recurrence function is $T(n) = T(n-1) + T(0) + (n-1)$. Since $T(0) = 0$, we have

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

   Apply this equation to $n-1, n-2, \dots, 1$ and sum them up, we get

   $$T(n) - T(0) = \sum_{i=0}^{n-1} i$$

   Since $T(0) = 0$, then $T(n) = \frac{(n-1)n}{2} = O(n^2)$. □

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).

$count \leftarrow 0$; $mid \leftarrow \lfloor \frac{low+high}{2} \rfloor$;
**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.** The algorithm is correct, the idea of this algorithm is that for an array with $n$ elements, the ranges satisfying Eqn. (1) can be divided into three cases

(a) Both $i$ and $j$ are in the left half of the array.

(b) Both $i$ and $j$ are in the right half of the array.

(c) $i$ is in the left half and $j$ is in the right half.

Thus the total number of ranges statisfying Eqn. (1) can be counted Recursively.

The algorithm first divides the problem of size $n$ into two similar subproblems of size $n/2$. After the subproblems are conquered, the for loop will run $n/2$ times, every time the BinarySearch used to find $m$ and $n$ costs $O(\log(n/2))$ time, the final merge operation costs $O(n)$ time.The recurrence relation of Alg. 1 is

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

By using recursion-tree method, the tree has $\log n$ levels, in the $j$th level($j$ starts from 0), there are $2^j$ subproblems, and each with a size of $n/2^j$, the solution is $T(n) = O(n \log^2 n)$ $\square$

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 $\langle a_1, a_2, \ldots, a_n \rangle$ 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) See Fig.1 Fig.4

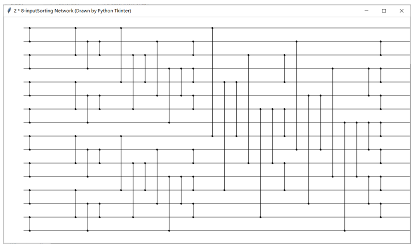

图 1: Graph $G_1$

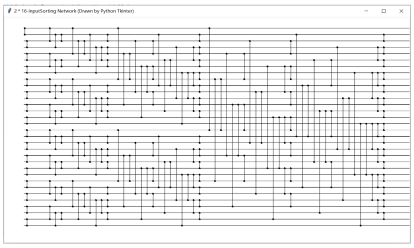

图 2: Graph $G_2$

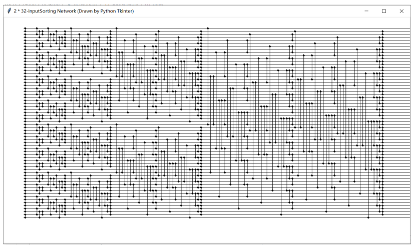

图 3: Graph $G_3$

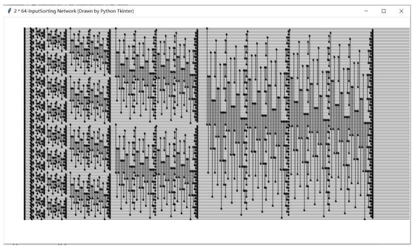

图 4: Graph $G_4$

(b) We denote the depth of a Since a $2n$-input odd-even sorting network as $D(2n)$. Since a $2n$-input odd-even sorting network can be recursively constructed, we have the recurrence function

$$D(2n) = D(n) + O(\log n)$$

By recurrence computation, the solution is $D(2n) = O(\log^2 n)$.

□

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