

# Lab02-Divide and Conquer

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

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

\* Name:\_\_\_\_\_ Student ID:\_\_\_\_\_ Email: \_\_\_\_\_

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.

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.)

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.)
  - (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.

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