Lab02-Divide and Conquer

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2021.

$*$ If there is any problem, please contact TA Haolin Zhou.

$*$ Name:Zirui Liu   Student ID:519021910343   Email: L.prime@sjtu.edu.cn

1. *Recurrence examples.* Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for sufficiently small n. Make your bounds as tight as possible.

   (a) $T(n) = 4T(n/3) + n\log n$

   (b) $T(n) = 4T(n/2) + n^2\sqrt{n}$

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

   (d) $T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \log n$

   Solution. we apply the following conclusion: Let $f(n)$ be $n\log n$, if $f(n) = O\left(n^{\log_b a} * \lg^k n\right)$ then $T(n) = O\left(n^{\log_b a} * \lg^{k+1} n\right)$. So $T(n)$ for (a) is

   (1) for $T(n) = 4T(n/3) + n\log n$, $a = 4$, $b = 3$, $1 < d < \log_3 4$, since $n\log n < n^{\frac{4}{3}}$, so applying the Master Theorem, we can conclude that $T(n) = O(n^{\log_3 4})$.

   (2) for $T(n) = 4T(n/2) + n^2\sqrt{n}$, we can simply apply the Master Theorem, $a = 4$, $b = 2$, $d = 2.5$, so $T(n) = O(n^{2.5})$.

   (3) for $T(n) = T(n-1) + n$, by simply applying recursion, we can know that $T(n) = n!$

   (4) for $T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \log n$ if we do some simple algorithm transformations, let $m = \log n$, then we have $T(2^m) = 2T\left(2^{\frac{m}{2}}\right) + m$, then we let $S(m) = T(2^m)$, so we got $S(m) = 2S\left(\frac{m}{2}\right) + m$, so $S(m) = O(m\lg m)$. Then we take this equation back to its former relation, we can conclude that $T(n) = O(\lg n \lg\lg n)$.

   □

2. *Divide-and-conquer.* Given an integer array $A[1..n]$ and two integers $lower \le upper$, design an algorithm using **divide-and-conquer** method to count the number of ranges (i, j) $(1 \le i \le j \le n)$ satisfying

$$lower \le \sum_{k=i}^{j} A[k] \le upper.$$

   Example:

   Given $A = [1, -1, 2]$, $lower = 1$, $upper = 2$, return 4.

   The resulting four ranges are $(1, 1)$, $(3, 3)$, $(2, 3)$ and $(1, 3)$.

   (a) Complete the implementation in the provided C/C++ source code (The source code *Code-Range.cpp* is attached on the course webpage).

   Solution. Please check out in the appendix. □

   (b) Write a recurrence for the running time of the algorithm and solve it by recurrence tree (You can modify the figure sources *Fig-RecurrenceTree.vsdx* or *Fig-RecurrenceTree.pptx* to illustrate your derivation).

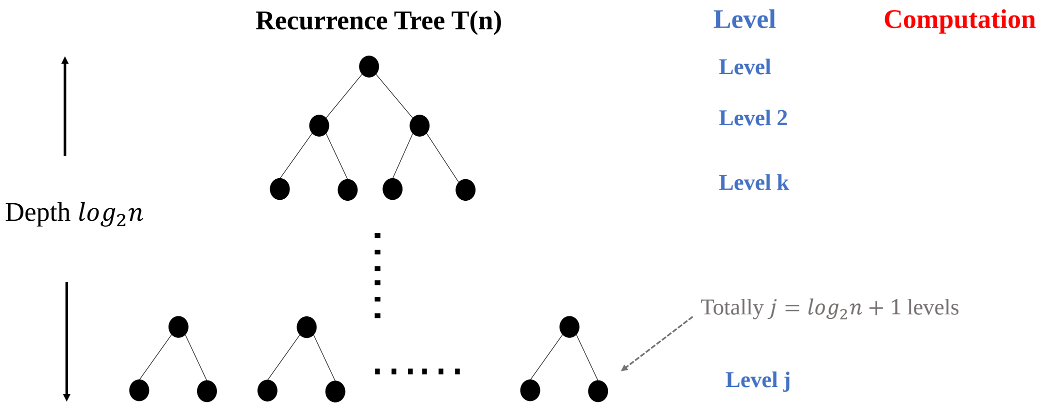

Figure 1: RecurrenceTree

Solution. Since the algorithm has time complexity of $T(n) = 2T\left(\frac{n}{2}\right) + n\log n$, We can conclude from the recursive tree that in the j-th level, there exists a time complexity of $2^{j-1} * O\left(\frac{n}{2^{j-1}} * \log\frac{n}{2^{j-1}}\right)$, so we calculate the sum of this equation, and we ignore the constant items of the function, we get $(\log_2 n + 1) * n * \log n$, that is the time complexity of $T(n) = n\log^2 n$. $\square$

(c) Can we use the Master Theorem to solve the recurrence above? Please explain your answer.

Solution. We can't apply the original edition of Master Theorem, but we can apply a modified version. We apply the following conclusion: Let $f(n)$ be $n\log n$, if $f(n) = O\left(n^{\log_b a} * \lg^k n\right)$ then $T(n) = O\left(n^{\log_b a} * \lg^{k+1} n\right)$. So $T(n)$ for this situation is $T(n) = O\left(n\log^2 n\right)$. $\square$

3. *Transposition Sorting Network.* A comparison network is a **transposition network** if each comparator connects adjacent lines, as in the network in Fig. 2.

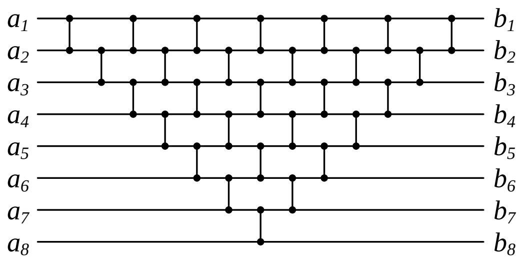

Figure 2: A Transposition Network Example

(a) Prove that a transposition network with n inputs is a sorting network if and only if it sorts the sequence $\langle n, n-1, \cdots, 1\rangle$. (Hint: Use an induction argument analogous to the *Domain Conversion Lemma.*)

(b) (Optional Sub-question with Bonus) Given any $n \in \mathbb{N}$, write a program using Tkinter in Python to draw a figure similar to Fig. 2 with n input wires.

Solution. Consider an input $\langle n, n-1, \cdots, 1\rangle$ to a transposition network and the state of the network after the k-th comparator. We denote the data held by line after the k-th comparator by i_{xk}. We use induction over the comparators in order. For a base case, consider the network before any comparators. For any i and j with $i < j$, $i_D, 0 > j_D, 0$ because the input is descending. Thus, we make no claim about any of the $i_X, 0$. Assume that the claim holds after the $k-1$-th comparator. Consider the k-th comparator. Because we have a transposition network, we compare two adjacent lines i and $i+1$. We need to show that the claim holds (for all pairs) of lines after the k-th comparator. That is, consider any two lines a and b with $1 \leq a < b \leq r$. Suppose that $a_D, k < b_D, k$. Then we need to show that $a_X, k < b_X, k$.

To be honest, I myself can't proof this and I think it is too difficult. I thought it for many hours and I haven't figured it out. But I found the original answer in [1].

$\square$

1 First appendix

Input C++ source1:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

/*
You need to complete the implementation of
binary_search_for_m and binary_search_for_n.
*/

int binary_search_for_m(vector<long>& sum, long sum_i, int LOWER, int low, in
        /*
        Find the smallest m in [low, high - 1]
        such that sum[m] - sum_i >= LOWER
        using binary search. If not found,
        return high.
        */
        int left=low;
        int right=high-1;
        int mid=(left+right)/2;
        while(left<right){
                mid=(left+right)/2;
                if(sum[mid]-sum_i>=LOWER){
                        right=mid;
                }else if(sum[mid]-sum_i<LOWER){
                        left=mid+1;
                }
        }
        if(sum[right]-sum_i<LOWER && right==high-1){
                return high;
        }else{
                return right;
        }

}

int binary_search_for_n(vector<long>& sum, long sum_i, int UPPER, int low, in
        /*
        Find the smallest n in [low, high - 1]
        such that sum[n] - sum_i > UPPER
        using binary search. If not found,
        return high.
        */
        int left=low;
        int right=high-1;
        int mid=(left+right)/2;
        while(left<right){
```

```
                mid=(left+right)/2;
                if(sum[mid]-sum_i>UPPER){
                        right=mid;
                }else if(sum[mid]-sum_i<=UPPER){
                        left=mid+1;
                }
        }
        if(sum[right]-sum_i<=UPPER && right==high-1){
                return high;
        }else{
                return right;
        }
}

int merge_count(vector<long>& sum, int low, int high, int LOWER, int UPPER) {
        int mid = (low + high) / 2;
        if (mid == low)
                return 0;
        int count = merge_count(sum, low, mid, LOWER, UPPER)
                + merge_count(sum, mid, high, LOWER, UPPER);
        int m_low = mid, m_high = high;
        int n_low = mid, n_high = high;
        for (int i = low; i < mid; i++) {
                int m = binary_search_for_m(sum, sum[i], LOWER, m_low, m_high
                int n = binary_search_for_n(sum, sum[i], UPPER, n_low, n_high
                count += n - m;     //          nmn-      m
        }
        sort(sum.begin() + low, sum.begin() + high); // You may assume the ti
        return count;
}

int main() {
        int N, LOWER, UPPER;
        vector<int> A;
        vector<long> sum(1, 0); // sum[i]: sum of the first i elements in A
        //cout<<"Please cin: "<<endl;
        cin >> N >> LOWER >> UPPER;
        for (int tmp, i = 0; i < N; i++) {
                cin >> tmp;
                A.push_back(tmp);
                sum.push_back(sum.back() + A.back());
        }

        cout << merge_count(sum, 0, N + 1, LOWER, UPPER) << endl;

        return 0;
}
```

References

[1] https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-896-theory-of-parallel-hardware-sma-5511-spring-2004/assignments/sol6.pdf