CSCI-SHU 220: Algorithms — Posted: February 5, 2024
Homework 1 — Due: 11:30pm (Shanghai time), February 23, 2024

This assignment has in total 70 base points and 10 extra points, and the cap is 70. Bonus questions are indicated using the $\star$ mark.

Please specify the following information before submission:

- Your Name: Yufeng Xu
- Your NetID: yx3038

Problem 1: Asymptotic analysis [$7 + 7 + 7$ pts]

(a) Show that $n! = \omega(n^{0.99n})$.

(b) Construct two functions $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$ such that $f(n) = O(g(n))$ but neither $f(n) = o(g(n))$ nor $f(n) = \Theta(g(n))$. Show the correctness of your construction.

(c) Let $f(n) = n^{0.6}$ and $g(n) = 2^{2^{\lfloor \log \log n \rfloor}}$. Show that none of the relations Θ, O, Ω, o, ω applies between $f(n)$ and $g(n)$.

Solution. **Please write down your solution to Problem 1 here.**

(a) want to show $n! = \omega(n^{0.99n}) \iff \forall c > 0, \exists n_0 > 0 \mid \forall n > n_0, n! > cn^{0.99n}$
Next, we prove it by showing $n! = \omega(n^{\frac{n}{2}})$
$n! > cn^{\frac{n}{2}} \iff \log 1 + \log 2 + \cdots + \log n > \log c + \frac{n}{2} \log n$
where LHS $= \log 1 + \log 2 + \cdots + \log \frac{n}{2} + \log\left(\frac{n}{2} + 1\right) + \cdots + \log(n-1) + \log n$
$> 0 + \log 2 + \cdots + \log 2 + \log 2 + \log \frac{n}{4} + \log \frac{n}{2} + \cdots + \log \frac{n}{2} + \log n$
$= \frac{n}{2}\left(\log \frac{n}{2} + \log 2\right) = \frac{n}{2} \log n$

(b)let $f(n) = \sin(\frac{n\pi}{2}) + 1, g(n) = \sin(\frac{n\pi}{2}) + 2$, take $b = 1, n_0 = 1, \forall n > n_0, f(n) \leq 1 \cdot g(n)$, so$f(n) = O(g(n))$
However, $f(n) \neq o(g(n))$ and $f(n) \neq \Theta(g(n))$

(i)If $f(n) = o(g(n))$, then $\forall b > 0, \exists n_0 > 0$ such that $\forall n > n_0, f(n) < b \cdot g(n)$
Take $b = 0.1$, assume $\exists n_0 > 0$ such that $\forall n > n_0, f(n) < 0.1 \cdot g(n)$
Take $n = 2\lceil n_0 \rceil > n_0, sin(\frac{2\lceil n_0 \rceil \pi}{2}) = sin(\lceil n_0 \rceil \pi) = 0$, $f(n) = 0 + 1 > 0.2 = 0.1 \cdot (0 + 2) = 0.1 \cdot g(n)$, contradictory to our assumption that $f(n) < c \cdot g(n)$. Therefore, $f(n) \neq o(g(n))$

(ii)If $f(n) = \Theta(g(n))$, then $\exists a, b > 0$ such that $\exists n_0 > 0, \forall n > n_0, a \cdot g(n) \leq f(n) \leq b \cdot g(n)$
Assume such a, b, n_0 exist, take $n = 4\lceil n_0 \rceil + 3, sin(\frac{n\pi}{2}) = sin\left((2\lceil n_0 \rceil + \frac{3}{2})\pi\right) = -1$, therefore $f(n) = 0, g(n) = 1$
we also know $a \cdot g(n) \leq f(n) \leq b \cdot g(n)$, therefore $a = 0$, contradictory to our essumption that $a > 0$.
Therefore, $f(n) \neq \Theta(g(n))$

(c)Consider $n = 2^{2^k}, k \in \mathbb{N}$, then $f(n) = 2^{0.6 \cdot 2^k}$, $g(n) = 2^{2^k}$, $\frac{g(n)}{f(n)} = 2^{0.4 \cdot 2^k}$
Consider $n = 2^{2^k - 1}, k \in \mathbb{N}$, then $f(n) = 2^{0.6 \cdot (2^k - 1)}$, $g(n) = 2^{2^{k-1}}$, $\frac{f(n)}{g(n)} = 2^{0.1 \cdot (2^k - 6)}$

(i)Assume $f(n) = O(g(n)), \exists b, n_0 > 0$ such that $\forall n > n_0, f(n) > b \cdot g(n)$

However, take $k = \max\{\lceil \log(\log n_0 + 1) \rceil + 1, \lceil \log(10 \cdot \log b + 6) \rceil + 1\}$, take $n = 2^{2^k - 1} > n_0$, then $\frac{f(n)}{g(n)} = 2^{0.1 \cdot (2^k - 6)} > b$, which is contradictory to $f(n) \leq b \cdot g(n)$.
Therefore, $f(n) \neq O(g(n))$, hence $f(n) \neq \Theta(g(n)), f(n) \neq o(g(n))$.

(ii)Assume $f(n) = \Omega(g(n))$, $\exists a, n_0$ such that $\forall n > n_0, f(n) \geq a \cdot g(n)$.
Take $k = \max\{\lceil \log \log n_0 \rceil + 1, \lceil \log(\frac{5}{2} \log \frac{1}{a}) \rceil + 1\}$, take $n = 2^{2^k} > n_0$, then $\frac{g(n)}{f(n)} = 2^{0.4 \cdot 2^k} > \frac{1}{a}$, which is contradictory to $f(n) \geq a \cdot g(n)$.
Therefore, $f(n) \neq \Omega(g(n))$, hence $f(n) \neq \omega(g(n))$. None of $\Theta, O, \Omega, o, \omega$ apply to $f(n)$ and $g(n)$.

Problem 2: Finding the maximum/minimum $[10 + 10^\star$ pts]

For an array A of n *different* numbers (not necessarily sorted), we want to find the *largest* number and the *smallest* number in A simultaneously. However, we have no access to A. Instead, we are given an oracle COMPARE that can be used to compare the numbers in A. For $i \neq j$, COMPARE(i, j) returns i if $A[i] > A[j]$ and returns j if $A[j] > A[i]$ (recall that the numbers in A are different by assumption, so we cannot have $A[i] = A[j]$ if $i \neq j$). For convenience, let us assume n is even.

(a) Design an algorithm which calls the COMPARE oracle at most $\frac{3}{2}n - 2$ times and returns a pair $(i_{\mathsf{max}}, i_{\mathsf{min}})$ such that $A[i_{\mathsf{max}}]$ (resp., $A[i_{\mathsf{min}}]$) is the largest (resp., smallest) number in A. Give the pseudocode and briefly justify its correctness.

(b)* Show that any algorithm has to call the COMPARE oracle $\frac{3}{2}n - 2$ times in worst case in order to find the largest and smallest numbers in A.

Solution. **Please write down your solution to Problem 2 here.**
(a) The pseudocode is shown below:

Algorithm 1 pseudocode for problem 2.1

```
max_idx ← COMPARE(0, 1)
min_idx ← 1− max_idx
i ← 2
while i < len(A) do
    tmp_max ← COMPARE(i, i + 1)
    tmp_min ← 2 · i + 1−tmp_max
    max_idx ← COMPARE(tmp_max,max_idx)
    min_idx ← tmp_min+min_idx −COMPARE(tmp_min,min_idx)
    i ← i + 2
end while
return (max_idx, min_idx)
```

This algorithm compares the first two elements for 1 time. For every two elements in the next $n - 2$ elements of the array, the algorithm compares for 3 times. In total, the algorithm compares $1 + (n - 2) \cdot \frac{3}{2} = \frac{3}{2}n - 2$ times.
Moreover, this algorithm is correct, because if we pick the larger number from every two numbers in the sequence, the global maximal number must be among this set. On the other hand, the global minimal number must be among the set of the smaller numbers from every two numbers.

(b) Let A be the set of numbers that are possibly minimum but not maximum, B be the set of numbers that are possibly maximum but not minimum, C be the set of numbers neither possibly

minimum nor maximum, D be the set of numbers both possibly minimum and maximum.
Assume the size of the four sets are (n_A, n_B, n_C, n_D).Initially, we have $(n_A, n_B, n_C, n_D) = (0, 0, 0, n)$. The ultimate goal is $(1, 1, n-2, 0)$. With direct comparison, there are 3 types of meaningful operations(all other operations contribute to this goal less efficiently):

- $(n_A, n_B, n_C, n_D) \to (n_A + 1, n_B + 1, n_C, n_D - 2)$
- $(n_A, n_B, n_C, n_D) \to (n_A - 1, n_B, n_C + 1, n_D)$
- $(n_A, n_B, n_C, n_D) \to (n_A, n_B - 1, n_C + 1, n_D)$

It is not hard to see the second and the third operations must be done $n-2$ times in total, whereas the first operation should be done $\frac{n}{2}$ times. Therefore, the total number of comparisons is at least $\frac{3n}{2} - 2$.

Problem 3: Solving recurrences [$4 \times 5 + 9$ pts]

(a) Find big-Θ bounds for the following recurrences (and show your bounds are correct). For the base case, simply assume $T(n) = 1$ for all $n \leq 2$.

(i) $T(n) = 8T(n/3) + n^{1.5}\log^4 n + 9n$

(ii) $T(n) = T(n - \sqrt{n}) + 6\log n$

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

(iv) $T(n) = 5\sqrt{n} \cdot T(\sqrt{n}) + 2n$

(b) Recall the Fibonacci sequence $F_0, F_1, F_2, \ldots$ defined using the recurrence $F_n = F_{n-1} + F_{n-2}$ with the base case $F_0 = 0$ and $F_1 = 1$. Prove by induction that $\phi^{n-2} \leq F_n < \phi^n$ for all $n \geq 1$, where $\phi = (1 + \sqrt{5})/2$. Based on this, further show that $F_n = \Theta(\phi^n)$.

Solution. **Please write down your solution to Problem 3 here.**

(a)(i) Let $T(n) = a \cdot T(\frac{n}{b}) + f(n)$, where $a = 8, b = 3, f(n) = \Theta(n^{1.5}\log^4 n)$
Because $\forall p, q > 0, n^p = \omega(\log^q n), \log_3 8 > 1.5$, $\exists \epsilon > 0$ such that $f(n) = O(n^{log_3 8 - \epsilon})$
According to Master theorem, $T(n) = \Theta(n^{log_3 8})$

(ii)Assume $T(n) = c\sqrt{n}\log n$, $T(n) - T(n - \sqrt{n}) = c\left(\sqrt{n}\log(n) - \sqrt{n - \sqrt{n}}\log(n - \sqrt{n})\right) = c\log n(\sqrt{n} - \sqrt{n - \sqrt{n}}\frac{\log(n-\sqrt{n})}{\log n}) > c\log n(\sqrt{n} - \sqrt{n - \sqrt{n}}) > c\log n \cdot \frac{1}{2}$(because$(\sqrt{n} - \frac{1}{2})^2 > n - \sqrt{n}$).
Therefore $LHS > 6\log n$ when $c \geq 12$.
On the other hand, $T(n) - 6\log n - T(n - \sqrt{n}) = (c\sqrt{n} - 6)\log(n) - c\sqrt{n - \sqrt{n}}\log(n - \sqrt{n})$. Let $c = 6$, we want to show $LHS < 0 \iff (\sqrt{n} - 1)\log(n) - \sqrt{n - \sqrt{n}}\log(n - \sqrt{n}) < 0 \iff \frac{1}{\sqrt{n}}\log(n) - \frac{1}{\sqrt{n - \sqrt{n}}}\log(n - \sqrt{n}) < 0$. Let $f(x) = 2\frac{\log(x)}{x}, f'(x) = \frac{2}{\ln(2)}\frac{1 - \ln(x)}{x^2} < 0$ for $x \geq 3$. Therefore, $f(\sqrt{n}) - f(\sqrt{n - \sqrt{n}}) < 0$ for $n > 16$, hence $T(n) - T(n - \sqrt{n}) < 6\log n$ when $n > 16$. Thus $T(n) = \Theta(\sqrt{n}\log n)$.

(iii)View $T(n)$ as a recursion tree. Assume the tree has k layers in total, the 0^{th} layer has one node; the 1^{st} layer has 2 nodes plus $\frac{n}{\log n}$; the 2^{nd} layer has 4 nodes plus $2 \cdot \frac{n}{2\log\frac{n}{2}} = \frac{n}{\log(n)-1} \ldots$ the k^{th} layer has 2^k nodes with value 1, plus $\frac{n}{\log(n)-k}$, where $k = \log n$

Therefore, $T(n) = 2^k + n \cdot \sum_{i=1}^{k} \frac{1}{i}$. By integral test, $\ln(k) < \sum_{i=1}^{k} \frac{1}{i} < \ln(k) + 1$. Therefore, $n \cdot \sum_{i=1}^{k} \frac{1}{i} = \Theta(n \log k) = \Theta(n \log \log n)$, $2^k = n = o(n \log \log n)$. Therefore, $T(n) = \Theta(n \log \log n)$.

(iv)View $T(n)$ as a recursion tree. Assume the tree has k layers in total, the 0^{th} layer has one node; the 1^{st} layer has $5n^{\frac{1}{2}}$ nodes plus $2n$; the 2^{nd} layer has $25n^{\frac{3}{4}}$ nodes plus $5\sqrt{n} \cdot 2\sqrt{n} = 10n \ldots$ the k^{th} layer has $5^k n^{1-\frac{1}{2^k}}$ nodes with value 1, plus $2 \cdot 5^{k-1} n$, where $k = \log \log n$.
Therefore, $T(n) = 5^k n^{1-\frac{1}{2^k}} + \sum_{i=1}^{k} 2 \cdot 5^{i-1} n = 5^k n^{1-\frac{1}{2^k}} + \frac{5^k - 1}{2} n$, where $5^k = 5^{\log \log n} = 2^{\log 5 \cdot \log \log n} = (\log n)^{\log 5} = \Theta\left((\log n)^{\log 5}\right)$, hence $5^k n^{1-\frac{1}{2^k}} = O(n(\log n)^{\log 5})$, $\frac{5^k - 1}{2} n = \Theta(n(\log n)^{\log 5})$. Therefore, $T(n) = \Theta(n(\log n)^{\log 5})$.

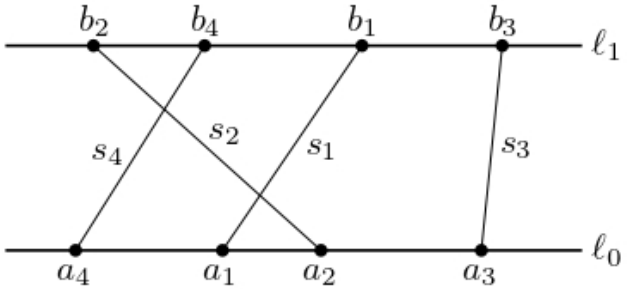
(b) (i) Check $\phi^{-2} \leq F_0 < \phi^0, \phi^{-1} \leq F_1 < \phi^1$.

(ii) Assume for $1, 2 \ldots k$, $\phi^{k-2} \leq F_k < \phi^k$. Then $F_{k+1} = F_k + F_{k-1} \geq \phi^{k-2} + \phi^{k-3} = \phi^{k-3} \cdot (1 + \phi) = \phi^{k-3} \cdot (1 + \frac{\sqrt{5}+1}{2}) = \phi^{k-3} \cdot \phi^2 = \phi^{k-1}$
$F_{k+1} < \phi^k + \phi^{k-1} = \phi^{k-1} \cdot (1 + \phi) = \phi^{k-1} \cdot \phi^2 = \phi^{k+1}$
Hence $\phi^{k-1} \leq F_{k+1} < \phi^{k+1}$. By induction, $\phi^{n-2} \leq F_n < \phi^n$ for all $n \geq 1$.

(iii) Now we show that $F_n = \Theta(\phi^n)$. We know $F_n = F_{n-1} + F_{n-2}$,
assume $F_n - A \cdot F_{n-1} = (1 - A) \cdot (F_{n-1} - A \cdot F_{n-2})$, then $1 = -A \cdot (1 - A)$, $A^2 - A - 1 = 0$, hence $A = \frac{1+\sqrt{5}}{2}$ or $\frac{1-\sqrt{5}}{2}$. Hence $F_n = C_1 \cdot (\frac{1+\sqrt{5}}{2})^n + C_2 \cdot (\frac{1-\sqrt{5}}{2})^n$. We know that $F_0 = 0, F_1 = 1$, hence $C_1 = \frac{\sqrt{5}}{5}, C_2 = -\frac{\sqrt{5}}{5}$, $F_n = \frac{\sqrt{5}}{5}(\frac{1+\sqrt{5}}{2})^n - \frac{\sqrt{5}}{5}(\frac{1-\sqrt{5}}{2})^n$, where $\frac{\sqrt{5}}{5}(\frac{1+\sqrt{5}}{2})^n = \Theta(\phi^n)$, $-\frac{\sqrt{5}}{5}(\frac{1-\sqrt{5}}{2})^n = O(\phi^n)$, hence $F_n = \Theta(\phi^n)$.

Problem 4: Counting intersection points [10 pts]

Consider two horizontal lines $\ell_0 : y = 0$ and $\ell_1 : y = 1$ in the plane. We have n distinct points $a_1, \ldots, a_n$ on ℓ_0 and n distinct points $b_1, \ldots, b_n$ on ℓ_1. Note that the points $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ are *not* necessarily sorted by their x-coordinates. Now for each $i \in \{1, \ldots, n\}$, we draw a segment s_i connecting a_i and b_i. See the figure below for an example of $n = 4$. These segments $s_1, \ldots, s_n$ may intersect with each other, and for simplicity we assume no three segments intersect at the same point. Our goal is to count the number of intersection points of $s_1, \ldots, s_n$. Formally, design an algorithm CountInt(n, A, B) where $A[1 \ldots n]$ stores the x-coordinates of $a_1, \ldots, a_n$ and $B[1 \ldots n]$ stores the x-coordinates of $b_1, \ldots, b_n$. The algorithm should return the number of intersection points of the segments $s_1, \ldots, s_n$, and should have time complexity $O(n \log n)$. Describe the basic idea and give the pseudocode. Then justify its correctness and show it runs in $O(n \log n)$ time.



Solution. **Please write down your solution to Problem 4 here.**
The pseudocode for this problem is shown in the next page.

Algorithm 2 pseudocode for problem 4.1

```
procedure CountInt(n, A, B)
    A_sorted ← sorted(A)          ▷ sorted is implemented by merge sort, which is O(n log n)
    B_sorted ← sorted(B)
    rank ← zeros(n)                          ▷ zeros(n) returns an array of zeros of length n
    for i in range(1, n + 1) do
        rank[B_sorted.rank(B[i])] = A_sorted.rank(A[i])
        ▷ rank applies binary search(O(log n)) and returns the rank of the target in a sequence.
    end for
    procedure DivConq(m, arr)
        if m ≤ 1 then
            return 0, arr
        end if
        mid ← m/2
        m ← len(arr)
        inv1, arr1 ← DivConq(mid, arr[: mid])
        inv2, arr2 ← DivConq(mid, arr[mid :])
        inv ← inv1 + inv2
        i, j ← 1
        res ← emptylist
        while i ≤ len(arr1) do
            if arr1[i] < arr2[j] OR j > len(arr2) then
                res.append(arr1[i])
                i ← i + 1
                inv ← inv + j − 1
            else
                res.append(arr2[j])
                j ← j + 1
            end if
        end while
        return inv, arr
    end procedure
    cnt, arr ← DivConq(n, rank)
    return cnt
end procedure
```

This algorithm applies merge sort 2 times and binary search $2n$ times, which is $O(n \log n)$. In DivConq, the sorting of $arr1$ and $arr2$, two sorted arrays, is $O(n)$. In total, the time complexity of DivConq is $O(n \log n)$. Therefore, the time complexity of the whole algorithm is $O(n \log n)$. Next, we will show the correctness of this algorithm.

We know if $a_i < a_j, b_i > b_j$, then there's an intersection between $a_i b_i$ and $a_j b_j$. In other words, the task of counting intersection is equivalent to counting inversed pairs. # inversed pairs in a sequence is equivalent to # inversed pairs within its left part and right part, plus # inversed pairs across the two parts, i.e., for each entry x in the left part, how many entries in the right part are smaller than x. By sorting $arr1$ and $arr2$, this algorithm keeps the returned array sorted and counts how many inversed pairs are there across the left and the right part.