HW 1

Sourav Mayla.

Problem 1: Ordering functions asymptotically in slower to higher growth rate.

(1) $\ln\ln n$

(2) $\ln n$

(3) n

(4) $n \ln n$

(5) $\ln(n!)$

(6) n^2

(7) $\lfloor \ln n \rfloor !$

(8) $n^{\ln\ln n}$

(9) $(\ln n)^{\ln n}$

(10) $(3/2)^n$

(11) 2^n

(12) $n 2^n$

(13) $n!$

(14) $(n+1)!$

(15) 2^{2^n}

(16) $2^{2^{n+1}}$

$\Rightarrow \ln\ln n = o(\ln n)$ — (1)

$\Rightarrow \ln n = o(n)$ — (2)

$\Rightarrow n = o(n \ln n)$ — (3)

$\Rightarrow \ln(n!) = \Theta(n \ln n)$ — (4)

$\Rightarrow n \ln n = o(n^2)$ — (5)

$\Rightarrow n^2 = o(\lfloor \ln n \rfloor !)$ — (6)

$\Rightarrow \lfloor \ln n \rfloor ! = o(n^{\ln\ln n})$ — (7)

$\Rightarrow n^{\ln\ln n} = \Theta(\ln n)^{\ln n}$ — (8)

$\Rightarrow (\ln n)^{\ln n} = o((3/2)^n)$ — (9)

$\Rightarrow (3/2)^n = o(2^n)$ — (10)

$\Rightarrow 2^n = o(n 2^n)$ — (11)

$\Rightarrow n 2^n = o(n!)$ — (12)

$\Rightarrow n! = o((n+1)!)$ — (13)

$\Rightarrow (n+1)! = o(2^{2^n})$ — (14)

$\Rightarrow 2^{2^n} = o(2^{2^{n+1}})$ — (15)

Proofs :-

(1) $\ln\ln n = O(\ln n)$

$\Rightarrow \lim_{n\to\infty} \frac{\ln\ln n}{\ln n}$ (using theorem 2)

$\Rightarrow \lim_{n\to\infty} \left(\frac{\frac{d}{dn}(\ln\ln n)}{\frac{d}{dn}(\ln n)} \right)$ (using L'Hospitals rule)

$\Rightarrow \lim_{n\to\infty} \frac{1/n\ln n}{1/n} = \lim_{n\to\infty} \frac{1}{\ln n} = \frac{1}{\infty} = 0$

(2) $\ln n = O(n)$

$\Rightarrow \lim_{n\to\infty} \frac{\ln n}{n} \Rightarrow \lim_{n\to\infty} \frac{\frac{d}{dn}(\ln n)}{\frac{d}{dn}(n)}$

$\Rightarrow \lim_{n\to\infty} \frac{1/n}{1} = \frac{1}{\infty} = 0$

(3) $n = O(n\ln n)$

$\Rightarrow \lim_{n\to\infty} \frac{n}{n\ln n} = \lim_{n\to\infty} \frac{\frac{d}{dn}(n)}{\frac{d}{dn}(n\ln n)}$

$\Rightarrow \lim_{n\to\infty} \frac{-(1+\ln n)}{(n\ln n)^2} = \frac{1}{\infty} = 0$

(4) $\ln n! = \theta(n \ln n)$

using stirling approximation

$$n! = \theta\left(n^{1/2}\left(\frac{n}{e}\right)^n\right)$$

Applying ln

$$\ln n! = \theta\left(\frac{1}{2}\ln n + n\ln n - n\ln e\right)$$

$$\Rightarrow \theta(\theta(n \ln n)) \quad \{\text{by addition \& subtraction property}\}$$

$$\ln n! \Rightarrow \theta(n \ln n).$$

(5) $n \ln n = o(n^2)$

$$\lim_{n\to\infty} \frac{n \ln n}{n^2} \Rightarrow \lim_{n\to\infty} \frac{\frac{d(\ln n)}{dn}}{\frac{d(n)}{dn}} = \lim_{n\to\infty} \frac{1/n}{1} = \frac{1}{\infty} = 0$$

(6) $(\lfloor \ln n \rfloor)^{\ln n}$

$n^2 = o(\lfloor \ln n! \rfloor)$

$\Rightarrow 2^{\lg(\lfloor \ln n \rfloor)!} = 2^{\lg((\lfloor \ln n \rfloor)^{\ln \frac{n+1}{2}} e^{-\ln n})}$

$\Rightarrow 2^{o((\lfloor \ln n \rfloor)^{\ln \frac{n+1}{2}} e^{-\ln n})}$

$n^2 = 2^{\lg n^2} = 2^{o(n)}$

$o(n) = o(\ln n^{\ln \frac{n+1}{2}} e^{-\ln n})$

using exponential rule

$n^2 = o(\lfloor \ln n \rfloor!)$

(7) $\lfloor \ln n \rfloor! = o(n^{\ln \ln n})$

$n^{\ln \ln n} = 2^{\lg n \ln \ln n} = 2^{o(\ln \ln n)}$

Similar to above $\lfloor \ln n \rfloor! = 2^{o((\ln n)^{\ln \frac{n+1}{2}} e^{-\ln n})}$

using exponential rule

$\lfloor \ln n \rfloor! = o(n^{\ln \ln n})$

(8) $n^{\ln \ln n} = \Theta(\ln n^{\ln n})$

using log property

$a^{\log_b c} = c^{\log_b a}$

$a = \ln n, \quad b = e, \quad c = 1$

~~$(\ln n)$~~

$n^{\ln \ln n} = \Theta(\ln n^{\ln n})$

(9) $(\ln n)^{\ln n} = \Theta\left(\left(\frac{3}{2}\right)^n\right)$

$\Rightarrow (\ln n)^{\ln n} = \left(2^{\ln \ln n}\right)^{\ln n} = 2^{\Theta(\ln n \ln \ln n)}$

$\Rightarrow \left(\frac{3}{2}\right)^n = 2^{\ln 3/2} = 2^{\Theta(n)}$

since $\Theta(\ln n \ln \ln n) = O(n)$

$(\ln n)^{\ln n} = O\left(\left(\frac{3}{2}\right)^n\right)$

(10) $\left(\frac{3}{2}\right)^n = O(2^n)$

$\Rightarrow \lim_{n \to \infty} \frac{(3/2)^n}{2^n} \Rightarrow \lim_{n \to \infty} \left(\frac{3}{4}\right)^n =$

Applying Limit $(a^n) = 0$

$n \to \infty$

$\left(\frac{3}{2}\right)^n = O((2)^n)$

(11) $\lim_{n\to\infty} \frac{2^n}{n2^n}$

$\Rightarrow \lim_{n\to\infty} \frac{1}{n} = \frac{1}{\infty} = 0$

$2^n = o(n2^n)$

(12) $n2^n = O(n!)$

$n! = 2^{\lg(n!)} \doteq 2^{\Theta(n\lg n)}$

$n2^n = n2^{\Theta(n)}$

Since $\Theta(n) = o(n\lg n)$

By exponential property.

$n2^n = o(n!)$

(13) $n! = O((n+1)!)$

$\lim_{n\to\infty} \frac{n!}{(n+1)!} = \lim_{n\to\infty} \frac{n!}{(n+1)n!} = \frac{1}{\infty} = 0$

$\Rightarrow n! = o((n+1)!)$

(14) $(n+1)! = O(2^{2^n})$

$(n+1)! = 2^{\log((n+1)!)} = 2^{\Theta(n\log n)}$

And $2^{2^n} = 2^{\Theta(2^n)}$

$\theta(n \log n) = \theta(2^n)$

By exponential property

$(n+1)! = o(2^{2^n})$

(15) $2^{2^n} = o(2^{2^{n+1}})$

for any value of n 2^{2^n} grow slower than $2^{2^{n+1}}$.

Problem 2 :- Superpolynomial growth.

→ The conjecture is false, we can proved this using proof of contradiction.

→ We already know,

$\forall\ a>0,\ b>1 \quad (n^a = o(b^n))$

→ Polynomial function grows slower than exponential function.

Also, by an example.

let $f(n) = n^{\log n}$

We can say

$\log n = \omega(1)$

thus, $f(n) = \omega(n^c)$ for any constant c.

Since, $n = b^{\log n}$

$f(n) = (b^{\log n})^{\log n} = b^{\log^2 n}$

By taking limit between $\log^2 n$ & n,

$$\Rightarrow \lim_{n\to\infty} \frac{\log^2 n}{n} \Rightarrow \lim_{n\to\infty} \frac{\frac{d}{dn}(\log^2 n)}{\frac{d(n)}{dn}} \quad \text{(by L'Hospital's Rule)}$$

$$\Rightarrow \lim_{n\to\infty} \left(\frac{2\log n/n}{1}\right) \Rightarrow 2\lim_{n\to\infty} \frac{\log n}{n}$$

$$\Rightarrow 2\lim_{n\to\infty} \frac{1}{n} = \frac{1}{\infty} = 0$$

Therefore, $\log^2 n = o(n)$. So, $f(n) = b^{\log^2 n} = o(b^n)$. Since $f(n)$ is strictly upper bounded by some exponential function b^n, the conjecture is not true.

Problem 3 :- Solving recurrences.

(a)

Extended Specialized master theorem :-

$$T(n) := aT\left(\frac{n}{b}\right) + \Theta(n^c \log^d n)$$

$$T(n) = \begin{cases} \Theta(n^{\log_b a}), & c < \log_b a \\ \Theta(n^c \cdot \log^{d+1} n), & c = \log_b a \\ \Theta(n^c \log^d n), & c > \log_b a \end{cases}$$

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

After comparing this with theorem, we get

$a = 4,\ b = 2,\ c = 5/2$ & $d = 0$

$\Rightarrow \log_b a = 2$

Since $c > \log_b a$, therefore

$$T(n) = \Theta(n^c \log^d n)$$

$$\Rightarrow \Theta(n^{5/2} \cdot \log^0 n)$$

$$\Rightarrow \Theta(n^2\sqrt{n})$$

(b) $T(n) = 3T(n/2) + n \lg n$

After comparing this with theorem, we get

$a = 3,\ b = 2,\ c = 1$ & $d = 1$

$\Rightarrow \log_b a = 1.58$.

Since $c < \log_b a$,

$$T(n) = \theta(n^{\log_b a})$$

$$\Rightarrow \theta(n^{\log_2 3})$$

(c) $T(n) = 2T(n/2) + n$

$a = 2, b = 2, c = 1$ & $d = 0$

$\Rightarrow \log_2 2 = 1$, Since $c = \log_b a$

$$T(n) = \theta(n^c \log^{d+1} n)$$

$$\Rightarrow \theta(n \log n)$$

(d) $T(n) = 2T(n/2) + n\log n$

$a = 2, b = 2, c = 1$ & $d = 1$

$\Rightarrow \log_2 2 = 1$, Since $c = \log_b a$

$$T(n) = \theta(n \log^2 n).$$

Problem 4:- (Minimum Positive subarray):-

To find a minimum positive-sum subarray for array A[low, high], we divide the subarray into two subarrays at the midpoint mid, then the minimum positive-sum subarray A[i, j] must be in one of the following places:-

→ entirely in the subarray A[low, mid], such that $low \le i \le j \le mid$

→ entirely in the subarray A[mid+1, high], such that $mid+1 \le i \le j \le high$

→ crossing the midpoint, such that $low \le i \le mid < j \le high$

We can find the minimum positive sum subarrays for A[low, mid] & A[mid+1, high] recursively because they are just smaller instances of the same problem.

Thus, all that is left to do is find a minimum positive sum subarray that crosses the midpoint, & take the minimum of the three cases.

The Pseudocode is:

```
function findMinPosSumSubArray(A, low, high)
    if low == high
        return (low, high, A[low])
    mid := ⌊(low+high)/2⌋
    (left_low, left_high, left_sum) := findMinPosSumSubArray(
                                        A, low, mid)
    (right_low, right_high, right_sum) := findMinPosSumSubArray
                                        (A, mid+1, high)
```

```
B := [ ]
C := [ ]
ComputePrefixSum(B, A, low, mid)
ComputePrefixSum(C, A, mid+1, high)
MergeSort(B)
MergeSort(C)
(left_low, left_high, left_sum) := ScanPrefixSum(B)
(right_low, right_high, right_sum) :=
                                ScanPrefixSum(C)
(cross_low, cross_high, cross_sum) := MinCrossPosSumSubArr
                                (A, B, C, low, mid, high)
if (left_sum is min positive sum)
    return (left_low, left_high, left_sum)
elseif (right_sum is min positive sum)
    return (right_low, right_high, right_sum)
elseif (cross_sum is the minimum positive sum)
    return (cross_low, cross_high, cross_sum)
else
    return (no positive sum exists)
```

We have used 2 additional arrays B & C to store prefix sum for each subarray computed by the subroutine "ComputePrefixSum". Then we merged sort B & C. The subroutine "ScanPrefixSum" does a linear scan on B and C to find out the minimum positive sum of two subarrays.

The subroutine "MinCrossPosSumSubArr" is used to find out the minimum positive sum that cross the midpoints.

It does following things:

→ for each positive element x in B, use $1-x$ as a key and do a binary search against C to find the closest element to the key. Also keep track of the minimum positive sum & its index interval.
→ Do the same for C on B.

Runtime analysis:

ComputePrefixSum require $O(n)$ time
MergeSort require $O(n \log n)$ time
ScanPrefixSum require $O(n)$ time {since B & C are sorted}

MinCrossClassSumSubArr require $O(n \log n)$ times. Since both B & C are sorted & binary search require $O(\log n)$ time

Putting all together, we have

$$T(n) = 2T(n/2) + \Theta(n \log n)$$

By the extended form of master theorem,

$$a \geq 2,\ b \geq 2,\ c = 1 \text{ & } d = 1$$

$$\log_b a = c$$

$$T(n) = \Theta(n^c \log^{d+1} n)$$

$$\Rightarrow \Theta(n \log^2 n)$$

Problem 5 (identifying good chips by pairwise tests):-

(a) Suppose we have n chips: $c_1, c_2, \ldots c_n$ among which $c_1, c_2, \ldots c_k$ are good chips where $k \leq n/2$.

These good chips identify each other as good and all others as bad. It is possible that the same number of bad chips can behave exactly like the good ones, they can also identify each other as good and all the other as bad, says these bad chips are $c_{k+1}, c_{k+2}, \ldots, c_{2k}$. Then there are three possibilities :-

→ all chips are bad;
→ $c_1, c_2 \ldots c_k$ are good;
→ $c_{k+1}, c_{k+2}, \ldots c_{2k}$ are good;

Thus, there exists a symmetric behaviors in regards to the operation of pairwise comparison No strategy can find good chips by pairwise tests.

(b) Assumption :- There are more good chips than bad chips.

Algorithm :-

(i) If there is only 1 chip left, then it must be good.
(ii) Split all chips into pairs, If n is odd, there will be a stand alone chip.
(iii) Do pairwise tests, for each pair, if the result is both good, arbitrarily throw away one chip. otherwise throw away the pair.
(iv) Go back to step i recursively till we find one good chip.

The algorithm takes

$$O\left(\frac{n}{2} + \frac{n}{4} + \cdots + 1\right) = O(n) \text{ pairwise tests.}$$

After step i, ii, iii once, we have reduced the problem half of size & more than half of the remaining chips are good.

Suppose After step ii & step iii there are

a pairs showed good/good result
b pairs showed good/bad result
c pairs showed bad/bad result.

(4)

If the result is good/good then both chips are either good or bad, otherwise at least one of the chip is bad.

Since we throw away $a+2(b+c)$ chips, so only a chips remain & among $2(b+c)$ chips, at least half of them are bad.

→ If n is odd, then $n=2(a+b+c)+1$ & $a+1$ chip remain. If stand alone chip is bad the following relation must hold

$$\text{no of bad chip} \leq \frac{a-1}{2} \times 2 + b + c + 1$$

$$\Rightarrow a+b+c$$

this means among remaining a chips, there must be at least $\frac{(a-1)}{2}+1 = \frac{(a+1)}{2}$ good chips which means more than half remaing chips are good

→ If n is even, then $n=2(a+b+c)$, suppose more than half of the remaing a chips are bad, then the chips they were paired with are also bad, this means

$$\text{no. of bad chips} > \frac{a}{2} \times 2 + b + c$$

$$\Rightarrow a+b+c = \frac{n}{2}$$

This contradicts the facts that more than half of n chips are good.

Based on the above analysis, we can conclude that remaining chips contain more good chips in n/2 chips.

(c) - first we use the algorithm given in part b to find a good chip. This takes $\Theta(n)$ pairwise tests.

Then we use this good chip to test all the other n-1 chips to find other good one. This also takes $\Theta(n)$ pairwise tests.

Thus total no. of pairwise test is still $\Theta(n)$.