VE 477 Homework 2

Wang Yichao, ID: 517370910011

- **Exercise 1.**

1. (a) $\lim\limits_{n\to\infty} \frac{n^3-3n^2-n+1}{n^3} = 1$. Q.E.D.

(b) $\frac{\ln n^2}{\ln 2^n} = \frac{2\cdot \ln n}{\ln 2\cdot n}$. We take derivative and get it is decreasing when $n > 10$. So $\frac{\ln n^2}{\ln 2^n} < 1$ when $n > 10$. Thus $n^2 < 2^n$ when $n > 10$. Q.E.D.

(c) $\lim\limits_{n\to\infty} \frac{(n+a)^b}{n^b} = (\lim\limits_{n\to\infty} \frac{n+a}{n})^b = 1^b = 1$. Q.E.D.

2. (a) $f(n) = \mathcal{O}(g(n)$

(b) $f(n) = \Omega(g(n))$

3. (a) can not find

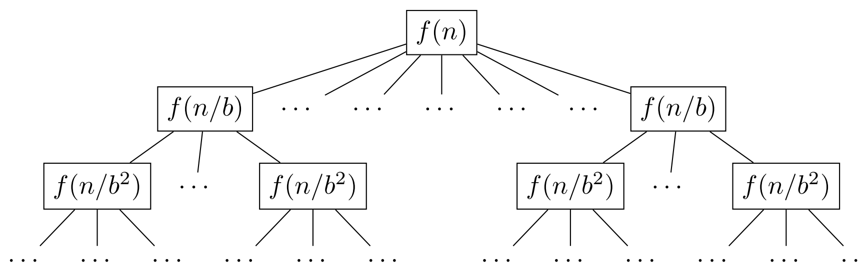
(b) f(n) = n and g(n) = 1

4. increasing

with the help of $\log n \le n$, we get $f_2(n) < f_1(n) < f_3(n) < f_4(n)$

- **Exercise 2.**

1. (a)



(b) depth is $\log_b n + 1$, number of leaves is $a^{\log_b n}$, the total cost at depth k is $a^k \cdot f(\frac{n}{b^k})$. Just recursive plug in the formula with n, $\frac{n}{b}$, $\frac{n}{b^2}$ ... We can get $T(n) = a^{\log_b n}\cdot T(1) + \sum_{j=0}^{\log_b n-1} a^j f\left(n/b^j\right) = n^{\log_b a}\cdot T(1) + \sum_{j=0}^{\log_b n-1} a^j f\left(n/b^j\right) = \Theta\left(n^{\log_b a}\right) + \sum_{j=0}^{\log_b n-1} a^j f\left(n/b^j\right)$.

2. (a) (i)since $f(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(a^{\log_b n}\right)$, we get $f(\frac{n}{b^j}) = \Theta\left(a^{\log_b \frac{n}{b^j}}\right)$, thus $a^j\cdot f(\frac{n}{b^j}) = \Theta\left(a^j\cdot a^{\log_b \frac{n}{b^j}}\right) = \Theta\left(a^j\cdot \frac{n}{b^j}^{\log_b a}\right)$. Adding them up and we can derive the answer. Q.E.D.

(ii) actually $a^j\cdot a^{\log_b \frac{n}{b^j}} = a^j\cdot a^{\log_b n-j} = a^{\log_b n} = n^{\log_b a}$. Since $\log_b n$ terms on LHS, the value of LHS should be $n^{\log_b a}\log_b n$. Q.E.D.

(iii) Since there is only a constant difference between $\log_b n$ and $\log n$, it is obvious that $g(n) = \Theta\left(n^{\log_b a}\log n\right)$ with the help of (ii) i just proved.

(b) (i) The proof is exactly the same(only difference is the big O instead of theta)(it is dirty work if we want prove by definition)

(ii) $a^j\left(\frac{n}{b^j}\right)^{\log_b a-\varepsilon} = a^j\left(a-\varepsilon\right)^{\log_b \frac{n}{b^j}} = \frac{a^j}{(a-\varepsilon)^j}\cdot n^{\log_b a-\varepsilon}$.

Thus we can get $\sum_{j=0}^{\log_b n-1} a^j\left(\frac{n}{b^j}\right)^{\log_b a-\varepsilon} = n^{\log_b a-\varepsilon}\cdot\sum_{j=0}^{\log_b n-1} b^{\varepsilon j} = n^{\log_b a-\varepsilon}\frac{n^\varepsilon-1}{b^\varepsilon-1}$.

(iii) plug in the result of (ii), it is obvious since $n^{\log_b a-\varepsilon} = n^{\log_b a}/n^\varepsilon$, and $\varepsilon > 0$.

$\lim\limits_{n\to\infty} \frac{n^\varepsilon-1}{b^\varepsilon-1} n^{\log_b a-\varepsilon}/n^{\log_b a} = \frac{1-n^{-\varepsilon}}{b^\varepsilon-1} = \frac{1}{b^\varepsilon-1}$, so $g(n) = \mathcal{O}\left(n^{\log_b a}\right)$.

(c) (i) 0when j = 0, we have term f(n), also it is obvious that other terms of $g(n) > 0$. So $g(n) = \Omega(f(n))$.

(ii) it is trivial. just use $af(n/b) \le cf(n)$ j times with $n = n, n/b, n/b^2 ... n/b^{j-1}$, since ¡ has transition property.

(iii) from (ii), we have $g(n) \le \sum_{j=0}^{\log_b n-1} c^j\cdot f(n) = f(n)\cdot\frac{1-c^{\log_b n}}{1-c}$, which is the stuff we want to prove. Q.E.D.

(iv) from (i) and (iii), we can get the result.

3. Weak Master Theorem

assume $a \geq 1, b > 1$ two constants. the recurrence relation is $T(n) = aT(\frac{n}{b}) + f(n)$. Then we can get the asymptotic bound of T(n) by

$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta\left(n^{\log_b a}\right) & f(n) = O\left(n^{\log_b a - \varepsilon}\right) \\ \Theta(f(n)) & af(n/b) \leq cf(n) \end{cases}$$

- **Exercise 3.**

Algorithm 1 mult

Input: a positive integer n

Output: all the Ramanujam numbers smaller or equal to n

```
let list[ ] be an array of n zeros.
let result[ ] be an empty list
for each interger 1 ≤ i ≤ ∛n do
    for each interger 1 ≤ j ≤ i do
        if i^3 + j^3 ≤ n then
            list[i^3 + j^3] ++;
        end if
    end for
end for
for each interger 1 ≤ k ≤ n do
    if 2 ≤ list[k] then result.append(k)
    end if
end for
return result[ ]
```

the iteration with two fors has time complexity of $1^2 + 2^2 + \cdots + \sqrt[3]{n}^2 = \mathcal{O}(n)$, for the second part that check duplicate elements, the time complexity is simply $\mathcal{O}(n)$. So the total complexity is still $\mathcal{O}(n)$.

- **Exercise 4.** WLOG, we can assume pirate a b c d e f with increasing age. We simply use backward induction. 6 iterations are needed.

  For a, it will propose itself 300.

  For b, since half vote can ensure winning, b will give itself 300.

  For c, c should give a 1 gold to get a happy and vote for c. no need to give b money.(notice that c need 300 to make b happy, which is nonsense)

  similarly, for d, it will give 1 gold to b. For e, it will give a c 1 gold each. For f, it will give b d 1 gold each.

  So the final result should be 6 pirates alive and the coin distribution is $\{0, 1, 0, 1, 0, 298\}$ from young to old.