UM–SJTU Joint Institute
VE477 Intro to Algorithms

Homework 2

Wang, Tianze
515370910202

Question 1 Basic complexity

1. a)

We first prove that $n^3 - 3n^2 - n + 1 = \mathcal{O}(n^3)$. We choose $c = 2$ and $n = 4$, next we calculate

$$c \cdot g(n) - f(n) = 2n^3 - (n^3 - 3n^2 - n + 1) = n((n - \frac{3}{2})^2 - \frac{13}{4})$$

For $n > 4$, obviously the former equation yields to a result greater than 0. Since we have found the c and n to make the condition validate, which means $n^3 - 3n^2 - n + 1 = \mathcal{O}(n^3)$.

Next is $n^3 - 3n^2 - n + 1 = \Omega(n^3)$. We choose $c = \frac{1}{2}$ and $n = 7$.

$$f(n) - c \cdot g(n) = (n^3 - 3n^2 - n + 1) - \frac{1}{2}n^3 = \frac{1}{2}n[(n-3)^2 - 11] + 1$$

For $n \geq 7$, the former equation yields to a result greater than 0, which means $n^3 - 3n^2 - n + 1 = \Omega(n^3)$.

Since $n^3 - 3n^2 - n + 1 = \mathcal{O}(n^3)$ and $n^3 - 3n^2 - n + 1 = \Omega(n^3)$, we could conclude that

$$n^3 - 3n^2 - n + 1 = \Theta(n^3)$$

□

1. b)

We set $c = 1$ and $n = 2$. We will find that when $n = 2$, $2^n = n^2$, for easier comparison, we transform them into log basis. which is $2 \log n$ and $n \log 2$

then we use

$$f(x) = \int f'(x)$$

So next we need to compare $\frac{d}{dn} 2 \log n = \frac{2}{n}$ and $\log 2$.

Obviously, $\frac{2}{n} \leq 1$, $\forall n \geq 2$, so we have

$$\frac{d}{dn} 2 \log n \leq \frac{d}{dn} n \log 2$$

And then

$$f(n) = 2 \log 2 + \int_2^n f'(n)$$

and

$$g(n) = 2 \log 2 + \int_2^n g'(n)$$

So $\forall n \geq 2$, and $c = 1$,

$$f(n) \leq g(n)$$

namely

$$n^2 = \mathcal{O}(2^n)$$

□

2. a)

$f(n) = \mathcal{O}(g(n))$. We choose $c = 1$ and $n = 9$. For the base case, namely $f(9)$ and $g(9)$, $f(n) \leq g(n)$. And we apply the same methods as 1.b), since $f'(n) < g'(n)$, $\forall n \geq 9$, we could conclude that

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

3. a)

Not exist.

3. b)

$f(n) = n$, $g(n) = 10$

4

When n is approaching ∞,

$$f_4(n) > f_1(n) > f_3(n) > f_2(n)$$

It is easy to obtain the order of f_2 and f_3,

$$\frac{f_3}{f_2} = \frac{\sqrt{n}}{\sqrt{\log n}} > 1 \Rightarrow f_3 > f_2$$

Next we need to compare f_3 and f_1. After observing the form of f_3 and f_1, we divide them into pairs, namely $p_i = \sqrt{i} + \sqrt{n+1-i}$ for f_1 and $q = 2\sqrt{n}$ for f_3.

Note that $f_1 = \sum_{i=1}^{n/2} p_i$ and $f_3 = \sum_{i=1}^{n/2} q$. Then we calculate $p_1^2 - q^2$,

$$p_1^2 - q^2 = n + 2\sqrt{n} + 1 - 4\log n > n - 4\log n$$

when $n \geq 9$, we will have $f_1^2 - f_3^2 > 0$. Similarly, we can derive that for every pair, $p_i > q$. And this tells $f_1 > f_3$.

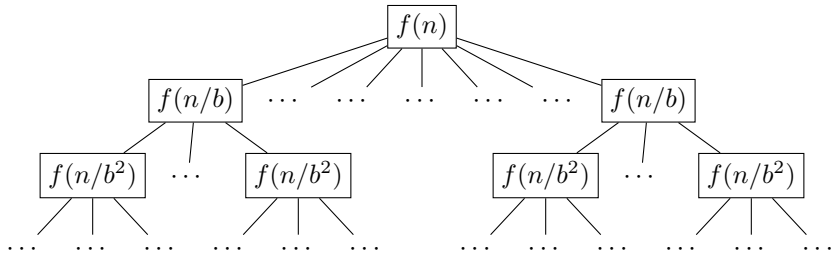
$f_4 > f_1$ is also obvious. That

$$f_4 > n\sqrt{n} > \underbrace{\sqrt{n} + \sqrt{n} + \cdots + \sqrt{n}}_{\text{totally n items}} > 1 + \sqrt{2} + \cdots + \sqrt{n} = f_1$$

□

Question 2 Master Theorem

1 a)



where each node has b number of child nodes.

1 b)

i) The depth of the tree is $\log_b n$

ii) The leaves are $a^{depth} = a^{\log_b n}$

iii) In each level, denote as level j, the cost is $a^j \cdot f(n/b^j)$

iv) T(n) is the rest items plus the calculation on each level,

$$T(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j) + \Theta(n^{log_b a})$$

Please note that the layer is calculated to the $log_b n - 1$ recursive call of $f(x)$, And for the bottom, namely the leaves, it will run several operations, which is $\Theta(a^{log_b n}) = \Theta(n^{log_b a})$, which is derived from the property of logarithm.

2 a)

1. We prove from the definition, that $\exists c1, c2, n_0$ s.t. $\forall n \geq n_0$, $c_1 \cdot n^{\log_b a} \leq f(n) \leq c_2 \cdot n^{\log_b a}$.

   We first prove the upper bound, and the lower bound can also be derived symmetrically.

   $$g(n) \leq \sum_{j=0}^{\log_b n - 1} a^j (c_2 \frac{n}{b^j})^{\log_b a} = c_2^{\log_b a} \left(\sum_{j=0}^{\log_b n - 1} a^j (\frac{n}{b^j})^{\log_b a} \right)$$

   Similarly,

   $$g(n) \geq c_1^{\log_b a} \left(\sum_{j=0}^{\log_b n - 1} a^j (\frac{n}{b^j})^{\log_b a} \right)$$

   So we find $c_2 = c_1^{\log_b a}$ and $c_3 = c_2^{\log_b a}$ s.t., $\forall n \geq n_0$,

   $$c_1^{\log_b a} \left(\sum_{j=0}^{\log_b n - 1} a^j (\frac{n}{b^j})^{\log_b a} \right) \leq g(n) \leq c_2^{\log_b a} \left(\sum_{j=0}^{\log_b n - 1} a^j (\frac{n}{b^j})^{\log_b a} \right)$$

   which means

   $$g(n) = \Theta(\sum_{j=0}^{\log_b n - 1} a^j (\frac{n}{b^j})^{\log_b a})$$

   □

2.

3. We use the result from *2.a.ii*, that

   $$\sum_{j=0}^{\log_b n - 1} a^j (\frac{n}{b^j})^{\log_b a} = n^{\log_b a} \log_b n$$

   And we substitute this into the result above, that

   $$g(n) = \Theta(n^{\log_b a} \log_b n)$$

2 b)

i) Not done yet.

ii) By observation, we can see that actually we need to show

$$\sum_{j=0}^{\log_b n-1} \frac{a^j}{(b^j)^{\log_b a-\varepsilon}} = \frac{n^\varepsilon - 1}{b^\varepsilon - 1}$$

And we apply transformations to the left part inside the sum, which is

$$\sum_{j=0}^{\log_b n-1} \frac{a^j}{(b^{\log_b a-\varepsilon})^j} = \sum_{j=0}^{\log_b n-1} b^{\varepsilon j} = \frac{1-b^{\varepsilon\cdot\log_b n}}{1-b^\varepsilon} = \frac{1-(b^{\log_b n})^\varepsilon}{1-b^\varepsilon} = \frac{n^\varepsilon - 1}{b^\varepsilon - 1}$$

□

iii)

$$\frac{n^\varepsilon - 1}{b^\varepsilon - 1} n^{\log_b a-\varepsilon} = \frac{n^\varepsilon - 1}{(b^\varepsilon - 1)\cdot n^\varepsilon} n^{log_b a}$$

We set $c_0 = 1$, and solve

$$\frac{n^\varepsilon - 1}{(b^\varepsilon - 1)\cdot n^\varepsilon} \leq 1$$

which is

$$n^\varepsilon \cdot (2 - b^\varepsilon) \leq 1$$

if $2 - b^\varepsilon \leq 0$, this equation obviously holds, and then we could conclude that $\forall n > n_0$, $\exists c_0 = 1$ s.t.

$$\frac{n^\varepsilon - 1}{b^\varepsilon - 1} n^{\log_b a-\varepsilon} \leq C \cdot n^{\log_b a}$$

which means

$$g(n) = O(n^{\log_b a})$$

2 c)

1. Simply set $c = 1$, and it is obvious that

$$\begin{aligned} g(n) &= a^0 f(n/b^0) + af(n/b) + a^2 f(n/b^2) \\ &> a^0 f(n/b^0) = f(n) \end{aligned}$$

which means

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

□

2. Let $t = n/b^{j-1}$,

$$a^j f(n/b^j) = a^j f(t/b) = a^{j-1} \cdot (a \cdot f(t/b)) \leq a^{j-1} c f(t) = c(a^{j-1} f(n/b^{j-1}))$$

Similarly, we apply the same method to $a^{j-1} f(n/b^{j-1})$, and so on and so forth,

$$a^j f(n/b^j \leq c \cdot a^{j-1} f(n/b^{j-1}) \leq c^2 \cdot a^{j-1} f(n/b^{j-2} \leq \cdots \leq c^j f(n)$$

□

3. We recall the definition of $g(n)$ is that

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$$

So

$$\begin{aligned} g(n) &= a^0 f(n/b^0) + af(n/b) + a^2 f(n/b^2) + \cdots + a^{\log_b n-1} f(n/b^{\log_b n-1}) \\ &\leq f(n) + cf(n) + c^2 f(n) + \cdots + c^{\log_b n-1} f(n) \\ &< \frac{1}{1-c} f(n) \text{ (Derived from infinite geometric sequence sum)} \end{aligned}$$

Let $c_0 = \frac{1}{1-c}$, this yields to

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

4. Since we can find c_0, c_1 s.t.

$$c_0 f(n) \leq g(n) \leq c_1 f(n)$$

so

$$g(n) = \Theta(f(n))$$

3

The master theorem is then given as:

Let $a \geq 1$, $b > 1$, be two constants, $f(n)$ be a function, and n is explicitly defined as a power of b. $T(n) = aT(n/b) + f(n)$ be a recurrence relation over the positive integers. Then the asymptotic bound on $T(n)$ is given by

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

Question 3 Ramanujam numbers

```
   Input  : An integer n
   Output : A list containing all numbers smaller the n that is formed by sum of two cubes
1  Function RamanHelper(n):
2      i ← 2;
3      temp ← 9;
       /* 9 = 1^3 + 2^3, which is the minimum possible sum of two different possible integers */
4      temp_list ← [] ;
       /* The temp_list is a data structure that allows repeated elements inside */
5      while temp ≤ n do
6          for all j ≤ i do
               /* It only checks at the next iteration after 2i^3 > n is because to avoid cases like
                  100^3 + 100^3 > 101^3 + 1^3 */
7              if j == i and 2(i − 1)^3 > n then
8                  return temp_list;
9              end if
10             if i^3 + j^3 ≤ n then
11                 temp_list.append(i^3 + j^3);
12             end if
13         end for
14         i ← i+1 ;
15     end while
16 end
   Input  : An integer n
   Output : All Ramanujam numbers smaller or equal to n
17 Function Raman(n):
18     L ← RamanHelper(n) ;
19     for All numbers in list L do
20         return All numbers that have occurred twice
21     end for
22 end
```

The complexity of the iteration in the helper function will be

$$\mathcal{O}(\sqrt[3]{n})$$

Since the iteration will mostly occur $2\sqrt[3]{n}$ times. And the count operation will be done for all the items in the array, the array length is

$$\mathcal{O}(\sqrt[3]{n^2})$$

And since we will consider the stricter limit, so we can conclude the total complexity

$$\mathcal{O}(\sqrt[3]{n^2})$$

Question 4 Pirates Sharing Gold

We use a strategy called trace-back, namely we use the what-if statement. We first consider the case for only one pirate, then two, so on and so forth, and 6 at last.

To make our life easier, we use a table, with each column as each pirate, the most left is the earliest, namely the last to share. This table is derived upside-down.

number of pirate alive	1	2	3	4	5	6
1	300	dead	dead	dead	dead	dead
2	0	300	dead	dead	dead	dead
3	1	0	299	dead	dead	dead
4	0	1	0	297	dead	dead
5	1	0	1	0	298	dead
6	0	1	0	1	0	298

When there is only one pirate, he will definitely get all the coins.

When there are two pirates, since one person's vote is enough for the **third rule**, namely the second pirate will also up vote, and the first pirate will always reject to get all the money. So the second pirate will take all the coins.

When there are three pirates, he will needs two votes from the remaining three pirates. The second pirate knows that he can get all the coins for his turn, so he will always vote no, whatever coins other gives him. So the third pirate needs to give no coin to him. And then he need to bribe the first pirate, since the first pirate will get no coin next turn, he will only need give this pirate one coin, which satisfies that he can get more money, and then pirate 1 will vote for him.

When there are four pirates, he needs two votes. Since pirate 2 will get no coin next turn, he will let this one has a little favor, namely one coin, and then take all the rest 299 coins. Then he is able to win the vote.

When there are five pirates, he needs three votes. Similarly, he only needs to give one coin to those who has nothing for next turn, namely pirate 1 and pirate 3, then he is able to win the vote.

When there are six pirates, similarly, he can win the vote by giving pirate 2 and pirate 4 one coin.