# ANLY550 Homework1

# Hongyang Zheng

***Question 1***

Programs see recursive.py, iterative.py and matrix.py

My experience with the programs:

1) The recursive method is the slowest one, because it needs to recalculate every fibonacci number before getting the $F_n$ we want. If $n = 0$, the program will return 0; if $n = 1$, the program will return 1; if $n \geq 2$, the program will recursively calculate $F_n$=$F_{n-1} + F_{n-2}$. Therefore, the time used for this method is $T(n) = T(n-1) + T(n-2) + O(1)$

2) The iterative method is faster, because it stores the numbers in an array and we don't need to recalculate every fibonacci number.

3) The matrix method is faster, because it uses matrix multiplication. For even index, the function first calculates $A^{n/2}$ and then squares it to get $A^n$. For odd index, the function first calculates $A^{n-1/2}$ and then squares it to get $A^{n-1}$ and then multiplies A one more time to get $A^n$. By this approach, it saves lots of time.

Overall, improving algorithm is really important for saving time and helping solve the task more efficiently

***Question 2***

| $A$ | $B$ | $O$ | $o$ | $\Omega$ | $\omega$ | $\Theta$ |
|---|---|---|---|---|---|---|
| $logn$ | $log(n^2)$ | Yes | No | Yes | No | Yes |
| $\sqrt[3]{n}$ | $(logn)^6$ | No | No | Yes | Yes | No |
| $n^2 2^n$ | $3^n$ | Yes | Yes | No | No | No |
| $\frac{n^2}{logn}$ | $nlog(n^2)$ | No | No | Yes | Yes | No |
| $(logn)^{logn}$ | $\frac{n}{log(n)}$ | No | No | Yes | Yes | No |
| $100n + logn$ | $(logn)^3 + n$ | Yes | No | Yes | No | Yes |

***row 1***

1.When $C = 1, N = 1$, after $n > 1$, $log(n) \leq log(n^2) = 2logn$, therefore, $A = O(B)$

2.When $C = 0.1, N = 1$, after $n > 1$, $log(n) \geq 0.1log(n^2) = 0.2logn$, therefore, $A = \Omega(B)$

3.Since $A = O(B)$ and $A = \Omega(B)$, $A = \Theta(B)$

4.$lim_{n\to\infty} \frac{logn}{log(n^2)} = \frac{logn}{2logn} = 0.5 \neq 0$, therefore, $A \neq o(B)$

5.$lim_{n\to\infty} \frac{log(n^2)}{logn} = \frac{2logn}{logn} = 2 \neq 0$, therefore, $A \neq \omega(B)$

***row 2***

$A = \sqrt[3]{n}, B = (logn)^6$, then we have $A = e^{\frac{1}{3}log(n)}, B = e^{6log(logn)}$.

Since $n$ grows faster than $logn$, then $A \geq C * B$ for all $C$ and $n$.

Therefore, $A \neq O(B)$, $A = \Omega(B)$ and $A \neq \Theta(B)$.

$lim_{n\to\infty} \frac{e^{\frac{1}{3}log(n)}}{e^{6log(logn)}} = lim_{n\to\infty} e^{\frac{1}{3}log(n) - 6log(logn)} = \infty, A \neq o(B)$.

When we reverse the limit, we have $lim_{n\to\infty} e^{6log(logn) - \frac{1}{3}log(n)} = e^{-\infty} = 0$. Therefore $A = \omega(B)$.

***row 3***

$A = n^2 2^n, B = 3^n$, then we have $A = e^{2logn + nlog2}, B = e^{nlog3}$.

$lim_{n\to\infty} \frac{e^{2logn+nlog2}}{e^{nlog3}} = lim_{n\to\infty} e^{(2logn + nlog2 - nlog3)} = e^{-\infty} = 0$, therefore $A = o(B)$, which indicates that $A = O(B)$.

When we reverse, we have $limit = \infty$, therefore $A \neq \omega(B)$. Since the limit is $\infty$, $A$ will be always less than $B$, therefore, $A \neq \Omega(B)$, and $A \neq \Theta$

***row 4***

$A = \frac{n^2}{logn}, B = nlog(n^2)$, then we have $A = e^{2logn - log(logn)}, B = e^{logn + 2log(logn)}$.

$lim_{n\to\infty} \frac{e^{2logn - log(logn)}}{e^{logn + 2log(logn)}} lim_{n\to\infty} e^{2logn - log(logn) - logn - 2log(logn)} = lim_{n\to\infty} e^{logn - 3log(logn)} = \infty$. Therefore $A \neq o(B)$. Since the limit is $\infty$, then $A$ will be always greater than $B$, therefore $A \neq O(B)$, and $A \neq \Theta(B)$.

When we reverse the limit, we will have $lim_{n\to\infty} e^{3log(logn) - logn} = e^{-\infty} = 0$. Therefore $A = \omega(B)$, which indicates that $A = \Omega(B)$.

***row 5***

$A = (logn)^{logn}, B = \frac{n}{logn}$, then we have $A = e^{logn * log(logn)}, B = e^{logn - log(logn)}$.

There is no $C$ and $n$ such that $e^{logn * log(logn)} \leq C * e^{logn - log(logn)}$, since with $n$ increasing, $log(logn)$ will be much greater than $1$.

Therefore, $A \neq O(B)$, $A = \Omega(B)$ and $A \neq \Theta(B)$.

$lim_{n\to\infty}\frac{e^{logn*log(logn)}}{e^{logn-log(logn)}}lim_{n\to\infty}e^{logn*log(logn)-logn+log(logn)} = lim_{n\to\infty}e^{(log(logn)-1)logn+log(logn)} = \infty$. Therefore $A \neq o(B)$.

When we reverse the limit, we have $lim_{n\to\infty}e^{(1-log(logn))logn-log(logn)} = e^{-\infty} = 0$. Therefore $A = \omega(B)$.

***row 6***

1.When $C = 100, N = 10$, after $n > 10$, $100n + logn \leq 100((logn)^3 + n)$, therefore, $A = O(B)$

2.Since $n$ is the most power term, so when $C$ is really small like $0.0001$,
$100n + logn \geq 0.0001((logn)^3 + n)$, therefore, $A = \Omega(B)$

3.Since $A = O(B)$ and $A = \Omega(B)$, $A = \Theta(B)$

4.$lim_{n\to\infty}\frac{100n+logn}{(logn)^3+n} = 100 \neq 0$, therefore, $A \neq o(B)$

5.$lim_{n\to\infty}\frac{(logn)^3+n}{100n+logn} = 1/100 \neq 0$, therefore, $A \neq \omega(B)$

***Question 3***

***1)***

$f_1(n) = n$

proof:

$f_1(2n) = 2n, f_1(n) = n$. For $n \geq 1$ and $C \geq 2$, $2n \leq C * n$.

Therefore, $f_1(2n) \leq C * f_1(n)$, which means $f_1(2n)$ is $O(f_1(n))$.

***2)***

$f_2(n) = 2^n$

proof:

$f_2(2n) = 2^{2n}, f_2(n) = 2^n$. Assume $2^{2n} \leq C * 2^n$ for some $n > N$, then divide both sides by $2^n$, we have $\frac{2^{2n}}{2^n} = \frac{(2^n)^2}{2^n} = 2^n \leq C$.

However, with n increasing, there is no such constant C can satisfy this situation, therefore, $f_2(2n) \geq C * f_2(n)$ for all the time, which means $f_2(2n)$ is not $O(f_2(n))$.

***3)***

If $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$, then there are constants $C_1, C_2$ such that $f(n) \leq C_1 * g(n)$ and $g(n) \leq C_2 * h(n)$.

Therefore, we have $f(n) \leq C_1 * g(n) \leq C_1 * C_2 * h(n)$, which is $f(n) \leq C_3 * h(n)$ and $C_3 = C_1 * C_2$.

By definition of Big-O notation, $f(n)$ is $O(h(n))$.

***4)***

Let $f = 2n$ and $g = (1 + sin(\frac{n\pi}{2}))n$

$f$ is not $O(g)$: there is no $C_1$ such that $2n \leq C_1 * (1 + sin(\frac{n\pi}{2}))n$, since some n can make the right side equal to $0$.

$g$ is not $O(f)$: there is no $C_2$ such that $2n \geq C_2 * (1 + sin(\frac{n\pi}{2}))n$, since some n can make the right side equal to $C_2 * 2n$, in which case, the right side can be greater than the left side.

***5)***

If f is $o(g)$, then we have $lim_{n\to\infty} \frac{f}{g} = 0$.

If f is $O(g)$, we have $f \leq C * g$. By dividing both sides $g$, we get $\frac{f}{g} \leq C$ after some n.

Since when n becomes very big, $\frac{f}{g}$ will approach $0$, $\frac{f}{g} \leq C$ for any $C \geq 0$.

Therefore, $f$ is $O(g)$

***Question 4***

Proof: assume the input size is divisible by three

(Base case): When the input size is 3, we divide it into three parts a, b, c. After first phase, b will be greater than a; after second phase, c will be greater than b; after third phase, b will still be greater than a. Therefore, the number in the three parts are sorted from smallest to biggest. Therefore, the base case holds.

(Inductive case): Assume for any input size $N < n$, the list of number can be sorted correctly by StoogeSort.

Then when input size is n, we divide it into three parts B1, B2, B3. In the first phase, we sort B1, B2. According to the assumption, B1, B2 can be recursively sorted correctly, so the numbers in B2 will be greater than that in B1. In the second phase, we sort B2, B3. According to the assumption, B2, B3 can be recursively sorted correctly, so B3 will be greater than B2. In the third phase, we sort B1, B2 again and B2 will be greater than B1. Finally, we get the numbers in B3 > the numbers in B2 > the numbers in B1. Therefore, the inductive case holds.

Assume the input size is divisible by three, then every time we need to divide the number of size to get index and then recursively sort for each phase. Since there are three phases, we will recursively use StoogeSort three times for 2/3 input. Therefore, the running time is: $T(n) = O(1) + 3 * T(\frac{2n}{3})$

By master theorem, we have $a = 3, b = 3/2, k = 0$, and $a > b^k$, therefore, $T(n) = O(n^{log_{3/2}3})$

***Question 5***

1)

$T(n) = 4T(n/2) + n^3$, according to Master Theorem, we have $a = 4, b = 2, c = 1, k = 3$.

Since $a = 4, b^k = 2^3 = 8$, then $a < b^k$, therefore, $T(n)$ is $O(n^3)$.

2)

$T(n) = 17T(n/4) + n^2$, according to Master Theorem, we have $a = 17, b = 4, c = 1, k = 2$.

Since $a = 17, b^k = 4^2 = 16$, then $a > b^k$, therefore, $T(n)$ is $O(n^{log_4 17})$.

3)

$T(n) = 9T(n/3) + n^2$, according to Master Theorem, we have $a = 9, b = 3, c = 1, k = 2$.

Since $a = 9, b^k = 3^2 = 9$, then $a = b^k$, therefore, $T(n)$ is $O(n^2 logn)$.

4)

$T(n) = T(\sqrt{n}) + 1$, let $m = log_2 n$, then $n = 2^{log_2 n} = 2^m$

$T(n) = T(2^m) = T((2^m)^{\frac{1}{2}}) + 1 = T(2^{\frac{m}{2}}) + 1$

Let $S(m) = T(2^m)$, then $S(\frac{m}{2}) = T(2^{\frac{m}{2}})$

Therefore, $T(n) = S(m) = S(m/2) + 1$, according to Master Theorem, we have $a = 1, b = 2, c = 1, k = 0$.

Since $a = 1, b^k = 2^0 = 1$, then $a = b^k$, therefore, $T(n)$ is $O(logm) = O(loglog_2 n)$.

***Question 6***

This pseudolyrics contains three parts:

1) from beginning to the first for loop

There are 45 words, and it takes 11.25s

2) from the first for loop to the second for loop

There are 14 words, and it takes 3.5s for each turn

3) from the second for loop to the end

There are 24 words, and it taks 6s for each turn

Let $T(n)$ be the time to sing BARLEYMOW(n), then $T(1) = 20.75, T(n) = T(n-1) + 3.5 + 6n$ for $n >= 2$

The total time is the sum of time for the above three parts:

```
T(n):
  t=11.25
  for i <- 1 to n
      t=t+3.5
      for j <- i downto 1
          t=t+6
  return(t)
```

Therefore,
$T(n) = 11.25 + 3.5n + (1 + 2 + 3 + \ldots + n) * 6 = 11.25 + 3.5n + \frac{n(n+1)}{2} * 6 = 11.25 + 3.5n + 3n(n+1$

Proof: (base case): when $n = 1$, $T(n) = 3 + 6.5 + 11.25 = 20.75$. Therefore, the base case holds

(induction): Assume $T(n-1) = 3(n-1)^2 + 6.5(n-1) + 11.25$ is true for n>=2

$T(n) = T(n-1) + 3.5 + 6n = 3(n-1)^2 + 6.5(n-1) + 11.25 + 3.5 + 6n = 3n^2 - 6n + 3 + 6.5n - 6.5$

Therefore, $T(n) = 3n^2 + 6.5n + 11.25$ is true.

The tight asymptotic bound is $\Theta(n^2)$

proof: $3n^2 + 6.5n + 11.25 \leq C_1 * n^2$, divide both sides by $n^2$, then we have $3 + \frac{6.5}{n} + \frac{11.25}{n^2} \leq C_1$

Since with $n$ increasing, $\frac{6.5}{n} + \frac{11.25}{n^2}$ becomes smaller and smaller.

Therefore, when $C_1 = 3 + 6.5 + 11.25 = 20.75$ and $N_1 = 1$, $3n^2 + 6.5n + 11.25 \leq 20.75 * n^2$, after $n > 1$, and $T(n)$ is $O(n^2)$.

$3n^2 + 6.5n + 11.25 \geq C_2 * n^2$, divide both sides by $n^2$, then we have $3 + \frac{6.5}{n} + \frac{11.25}{n^2} \geq C_2$

Since with $n$ increasing, $\frac{6.5}{n} + \frac{11.25}{n^2}$ becomes smaller and smaller, so the most power part is 3.

Therefore, when $C_2 = 0.01$ and $N_2 = 1$, $3n^2 + 6.5n + 11.25 \geq 0.01 * n^2$, after $n > 1$, and $T(n)$ is $\Omega(n^2)$, and $T(n)$ is $\Theta(n^2)$.

***Question 7***

See mergesort.py

Write up:

1) When the input list has a length greater than 1, the function first calculates the index for the middle of the list, and then separate the list into two subarrays: left and right.

2) For each subarray, the function is recursively applied until the length of the subarray is 1.

3) Then the function merges subarrays. It compares the numbers in the left and right subarrays and then puts the smaller number back into the list. The function does this step until all numbers in the right subarrays or all numbers in the left subarrays are merged.

4) In the last step, the function checks whether there are numbers left and then puts them into the list.