CSE431 HW1

Siqi Cheng, 50388579

Problem 1:

Here's the arrangement of functions in ascending order of growth rate:

1. f7(n) = log n
2. f6(n) = √n
3. f2(n) = n log n
4. f1(n) = n^2
5. f8(n) = n^2.5
6. f3(n) = n^3
7. f10(n) = n^ (log n)
8. f4(n) = 2^n
9. f9(n) = 3^n
10. f5(n) = n!

Explanation:

1. $f_7(n) = \log n$ & $f_6(n) = \sqrt{n}$

By using little o defination for every $n \geq n_0$

if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ then $f(n) = o(g(n))$ which means, $g(n)$ has fastest growth rate than $f(n)$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\log n}{\sqrt{n}} \xrightarrow{\text{L'Hopital's}} \lim_{n\to\infty} \frac{\frac{1}{n}}{\frac{1}{2\sqrt{n}}} = \lim_{n\to\infty} \frac{2}{\sqrt{n}} = 0$$

Therefore, $\underline{\sqrt{n}}$ has faster growth rate than $\underline{\log n}$

2. $f_6(n) = \sqrt{n}$ & $f_2(n) = n \log n$

By using little o defination

if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ then $f(n) = o(g(n))$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\sqrt{n}}{n \log n} = \lim_{n\to\infty} \frac{1}{\sqrt{n}\log(n)} = \frac{\lim_{n\to\infty} 1}{\lim_{n\to\infty}(\sqrt{n}\log(n))} = \frac{1}{\infty} = 0$$

Therefore, $n = \log n$ has faster graph rate than $\sqrt{n}$

3 $f_2(n) = n\log n$ & $f_1(n) = n^2$

By using little o defination

if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ then $f(n) = o(g(n))$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n\log n}{n^2} = \lim_{n\to\infty} \frac{\log n}{n} \xrightarrow[\text{rule}]{\text{L'Hopital's}} \frac{\frac{1}{x}}{1} = \frac{1}{x} = 0$

Therefore, n^2 has faster groth rate than $n\log n$

4. $f_1(n) = n^2$ & $f_8(n) = n^{2.5}$

By using Little o defination

if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, then $f(n) = o(g(n))$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2}{n^{2.5}} = \lim_{n\to\infty} \frac{1}{n^{0.5}} = 0$

Therefore, $n^{2.5}$ has faster growth rate than n^2

5. $f_3(n) = n^3$ & $f_8(n) = n^{2.5}$

By using Little o defination

if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, then $f(n) = o(g(n))$ for every $n \geq n_0$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^{2.5}}{n^3} = \lim_{n\to\infty} \frac{1}{n^{0.3}} = 0$

Therefore, n^3 has faster growth rate than $n^{2.5}$

6. $f_{10}(n) = n^{\log n}$ & $f_3(n) = n^3$

By using Little o defination

if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, then $f(n) = o(g(n))$ for every $n \geq n_0$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^3}{n^{\log n}} = \lim_{n\to\infty} n^{3-\log n} = \lim_{n\to\infty} e^{3-(\log n)\log n} = 0$

Therefore, $n^{\log n}$ has faster growth rate than n^3

7. $f_4(n) = 2^n$ & $f_\omega(n) = n^{\log n}$

By using little o defination

if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, then $f(n) = o(g(n))$ for every $n \geq n_0$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^{\log n}}{2^n} = \lim_{n\to\infty} \frac{\log n \cdot \log n}{n \log 2} \xrightarrow[\text{rule}]{\text{L' Hopital's}} \lim_{n\to\infty} \frac{\log n + 1}{\log 2}$$

$$= \lim_{n\to\infty} \frac{\log n + 1}{\log 2}$$

$$= 0$$

Hence, $n^{\log n}$ is in $o(2^n)$, which means $n^{\log n}$ grows strictly slower than 2^n

8. $f_8(n) = 3^n$ & $f_4(n) = 2^n$

By using little o defination

if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, then $f(n) = o(g(n))$ for every $n \geq n_0$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^n}{3^n} = \lim_{n\to\infty} \left(\frac{2}{3}\right)^n = 0 \quad \left(\lim_{n\to\infty} a^n = 0,\ 0 < a < 1\right)$$

Therefore, 3^n has faster growth rate than 2^n

9. $f_5(n) = n!$ & $f_8(n) = 3^n$

By using little o defination

Proof: for sufficiently large n, each term in factorial calculation: $n! = n \cdot (n-1) \cdot \cdots \cdot 2 \cdot 1$ is at least 3 (in n term).

if we replace each term in the product by 3, then:

$n! \geq 3 \cdot 3 \cdot 3 \cdot \cdots$ (for n) $= 3^n$

We also can use limit to proof:

$$\lim_{n\to\infty} \frac{3^n}{n!} = 0$$

The exponential function is in $o(n!)$. Therefore, 3^n grows slower than $n!$

Problem2

(1) By definition of BigO

O(g(n)) = f(n): for any positive constant c > 0, there exists a constant n_O >0 such that 0 ≤ f(n)< cg(n) for all $n \geq n_0$

In our case: T(n) is O(n^3) if $T(n) \leq c \cdot n^3$ for all $n \geq n_0$.

f(n)= $an^3 + bn^2 + cn + d \leq c \cdot n^3$, g(n) = n^3

we have to show that f(n) is in O(n^3), which means $f(n) \leq C \cdot n^3$ for all $n \geq n_0$

assume n_0= 1, and c is the biggest in {a, b, c, d}

we have: an^3 + bn^2 +$cn + d \leq cn^3 + cn^2 + cn + cd$ for all $n \geq 1, n^3$is the biggest.

-> $cn^3 + cn^2 + cn + cd \leq cn^3 + cn^3 + cn^3 + cn^3 = 4cn^3$

-> $an^3 + bn^2 + cn + d \leq c \cdot n^3$

Therefore, by using BigO, the run time is O(n^3)

(2):

Problem 2

(2) To prove (2),

we have to show that there exists constants $c_1, c_2, n_0 > 0$ for all $n > n_0$

$0 \leq c_1(f(n) + g(n)) \leq \max(f(n), g(n)) \leq c_2(f(n) + g(n))$, $f(n) \geq 0$. $g(n) \geq 0$

$f(n) + g(n) \geq \max(f(n), g(n))$

and also: $f(n) \leq \max(f(n), g(n))$ & $g(n) \leq \max(f(n), g(n))$

so, we have $\begin{cases} f(n) + g(n) \leq 2\max(f(n), g(n)) \\ \frac{1}{2}(f(n), g(n)) \leq \max(f(n), g(n)) \end{cases}$

By BigO defination: f(n): There exist positive constants c and n_0, such that

$0 \leq \frac{1}{2}(f(n) + g(n)) \leq \max(f(n), g(n)) \leq c f(n), g(n))$ for all $n \geq n_0$

Therefore, $O(\max(f(n), g(n))) = O(f(n), g(n))$.

Problem 3

```
function Heh(A[0..N-1], value) {
   l = 0
   h = N - 1
   while (l <= h) {
         m = l + ((h - l) / 2)
         if (A[m] > value)
                  h = m - 1
         else if (A[m] < value)
                  l = m + 1
         else
                  return m }
return "Heh" }
```

(1):

```
function Heh(A[0..N-1], l, h, value) {

   if (l > h) { return "Heh" }

   m = l + ((h - l) / 2)

   if (A[m] == value) {

      return m

   } else if (A[m] > value) {

      return Heh(A, l, m - 1, value)

   } else {

      return Heh(A, m + 1, h, value)

   }

}
```

(2):

T(n) = c if n <= 0

T(n) = T(n/2) + c if n > 0

The base case is when n <= 0. In this case the function just needs a set amount of time c to finish. so, it can be written as T(n) = c for n <= 0. For inputs n > 0, the function's running time is T(n) = T(n/2) + c. This means the function takes half the input size n and does some constant work c.

(3):

We defined for the running time is: T(n) = T(n/2) + c

By using Master Theorem:

We need to prove: T(n) = aT(n/b) + f(n), where a ≥ 1, b > 1.

In our case, a=1, b=2, f(n) = c = n^0

By Definition of Master Theorem case#2: if $f(n) = \theta\left(n^{\log_b a} \log^k n\right)$ and k>=0

Since $log_b\, a = log_2 1 = 0$, k=0, and $f(n) = C = \theta(n^0\, log^0\, n)$, Hence, it apply case#2.

Therefore, $T(n) = \theta(n^0\, log\, n) = \theta(log\, n)$

Problem 4

(1). T(n) = 100T(n/9)+n^2

4. (1) $T_1(n) = 100T_1(\frac{n}{9}) + n^2$

BY definition of Master Theorem.

$T(n) = aT(n/b) + f(n)$,

$a = 100,\ b = 9,\ f(n) = n^2$

∴ $\log_b a = \log_9 100 \approx 2.1$

Since, $f(n) = n^2 = n^{\log_b a - \epsilon} = n^{2.1-\epsilon}$ for some $\epsilon > 0$

∴ CASE I

Therefore, Apply case I, $T(n) = \Theta(n^{\log_b a})$

(2) T(n)=4T(n/2)+n^3

(2) $T_2(n) = 4T_2(\frac{n}{2}) + n^3$

BY definition of Master Theorem.

$T(n) = aT(n/b) + f(n)$

$a = 4,\ b = 2,\ f(n) = n^3$

∴ $\log_b a = \log_2 4 = 2$

∵ $f(n) = n^3 > n^{\log_b a} = n^2$

∴ $f(n) = n^3 = n^{2+\epsilon} = n^{\log_b a + \epsilon}$ for some $\epsilon > 0$

Moreover, The regularity condition is some constant $c < 1$,

$af(n/b) \le cf(n)$ for all sufficiently large n.

↓ $4(\frac{n}{2})^3 \le cn^3 \to 4/8\, n^3 = 0.5n^3$, if $c \le 0.5$.

Therefore, By CASE III, $T(n) = \Theta(f(n)) = \Theta(n^3)$

(3) T3(n) = 5T(n/2)+n^(log_2(5))

(3) $T_3(n) = 5T_3(\frac{n}{2}) + n^{\log_2 5}$

By defination of Marter Theorem,

$T(n) = a\,T(n/b) + f(n)$

In our case : $a = 5$, $b = 2$, $f(n) = n^{\log_2 5}$

$\because f(n) = n^{\log_2 5} = n^{\log_2 5} = n^{\log_b a}$

In our care $k = 0$ base on the $f(n) = \Theta(n^{\log_b a} \log^k n)$ in case II

Hence we are in CASE II

Therefore, $T_3(n) = 5T(\frac{n}{2}) + n^{\log_2 5}$ has a time complexity of

$\Theta(n^{\log_2 5} \log n)$

(4) T4(n) = T(n/4)+n!

(4) $T_4(n) = T_4(\frac{n}{4}) + n!$

$a = 1$, $b = 4$, $f(n) = n!$

we would like to verify if the condition $a f(n/b) \le c f(n)$ holds for some $c < 1$ and all sufficiently large n

$a f(\frac{n}{b}) \le c f(n)$

$a(\frac{n}{b})! \le c\,n! \Rightarrow 1 \cdot (\frac{n}{4})! \le c\,n! = (\frac{n}{4})! \le c\,n!$

We assume $n > n_0 = 4$, $c = 0.1 < 1$

we have : $(\frac{4}{4})! \le 0.1\,(4)!$

$1 \le 0.1 \cdot 24$

$1 \le 2.4$

So, We are in the CASE III of Master Theorem.

Therefore, Base on CASE III the time complexity is $T(n) = \Theta(f(n)) = \Theta(n!)$

Problem5

(1) Algorithm:

Recursive version:

```
function Med_Mountain(arr[0…N-1], low, high):
    if (low == high){
        return low
        mid = (low + high) / 2}(
    if (arr[mid] < arr[mid + 1]){
        return Med_Mountain (arr, mid + 1, right)}
    else{
        return Med_Mountain (arr, left, mid)}
```

(2)

In our algorithm, we start with two pointers, low and high, located at the beginning and end of the array, respectively. At each step, we either move the low pointer towards the high or vice versa. This means that the distance between the pointers gets progressively shorter until the two pointers meet at a single point. Here the algorithm stops, and this is where we find the peak.

The converse is true, if the point we converge on, "mid", is not a peak. Then it must be arr[mid] < arr[mid + 1] or arr[mid] < arr[mid - 1] (it can't be both, because it doesn't match the given input).

If arr[mid] < arr[mid + 1], our algorithm sets low to mid + 1 in the next iteration, which contradicts the fact that the algorithm converges to mid.

If arr[mid] < arr[mid - 1], this algorithm sets high to mid - 1, also contradicting our assumption.

(3) **T(n) = c if n <= 0**

T(n) = T(n/2) + c if n > 0

We defined for the running time is: T(n) = T(n/2) + c

By using Master Theorem:

We need to prove: T(n) = aT(n/b) + f(n), where a ≥ 1, b > 1.

In our case, a=1, b=2, f(n) = c = n^0

By Definition of Master Theorem case#2: if $f(n) = \theta\left(n^{\log_b a} \log^k n\right)$ and k>=0

Since $log_b\, a = log_2\, 1 = 0$, k=0, and $f(n) = C = \theta(n^0\, log^0\, n)$, Hence, it apply case#2.

Therefore, $T(n) = \theta(n^0\, log\, n) = \theta(log\, n)$