Chapter 2

Question 1:

proof: $p(n) = \sum_{i=0}^{d} a_i n^i$

$$= a_d \cdot n^d + \sum_{i=0}^{d-1} a_i n^i$$

$$= n^d \left(a_d + \sum_{i=0}^{d-1} a_i \frac{n^i}{n^d} \right)$$

$$= n^d \left(a_d + \sum_{i=0}^{d-1} a_i \frac{1}{n^{d-i}} \right)$$

Let $q(n) = \sum_{i=0}^{d-1} a_i \frac{1}{n^{d-i}}$, then $p(n) = n^d (a_d + q(n))$

$\because \lim_{n\to\infty} q(n) = 0$, $\therefore \exists\, n_0 \in \mathbb{Z}^+$, when $n \geq n_0$,

$|q(n)| \leq \frac{a_d}{2}$, $\therefore -\frac{a_d}{2} \leq q(n) \leq \frac{a_d}{2}$.

$\therefore n^d (a_d - \frac{a_d}{2}) \leq n^d (a_n + q(n)) \leq n^d (a_d + \frac{a_d}{2})$

Let $c_1 = \frac{a_d}{2}$, $c_2 = \frac{3}{2} a_d$; meanwhile, $p(n) = \Theta(n^d)$

then we only need to discuss the size relationship between k and d.

(a) If $k \geq d$, then, $0 \leq c_1 n^d \leq p(n) \leq c_2 n^d \leq c_2 n^k$.

According to the definition of $O(g(n))$:

$O(g(n)) = \{ f(n)$: there exist positive constants c and n_0, such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0 \}$; Let c_2 be c,

$0 \leq c_1 n^d \leq p(n) \leq c n^d \leq c n^k$. Let we extract the parts with underline

it is obvious that $p(n)$ satisfies the definition. QED.

(b) If $k \leq d$, similarly, $0 \leq c_1 n^k \leq c_1 n^d \leq p(n) \leq c_2 n^d$,

$\Omega(g(n)) = \{ f(n)$: there exist positive constants c and n_0, such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0 \}$; Let c_1 be c,

$\therefore 0 \leq c n^k \leq p(n)$, $p(n)$ satisfies the definition, QED

(c). If $k = d$, we have proved that $p(n) = \theta(n^d)$,

therefore, $p(n) = \theta(n^k)$, QED.

(d) If $k > d$, similarly, $0 \leq c_1 n^d \leq p(n) \leq c_2 n^d < c_2 n^k$,

$o(g(n)) = \{ f(n)$: there exist positive constants c and n_0, such that $0 \leq f(n) < cg(n)$ for all $n \geq n_0 \}$; Let c_2 be c,

$\therefore 0 \leq p(n) < c n^d$, $p(n)$ satisfies the definition, QED

(e) If $k<d$, similarly, $0 \le c_1 n^k < c_1 n^d \le p(n) \le c_2 n^d$,

$\Omega(g(n)) = \{f(n)$: there exist positive constants c and n_0, such that

$0 \le cg(n) < f(n)$ for all $n \ge n_0\}$; Let c_1 be c,

$\therefore 0 \le cn^k < p(n)$, $p(n)$ satisfies the definition, QED

Qestion 2.

proof:

① Assume that $\exists\, c > 0$ when $m < n$, $T(m) \le c \lg m$,

Let $m = \lceil \frac{n}{2} \rceil$, so m satisfies the assumption above.

$T(m) = T(\lceil \frac{n}{2} \rceil) \le c \lg \lceil \frac{n}{2} \rceil \le c \lg n - d$, $d \ge 1$ (1)

② when $m = n$, the original equition substituting formula (1)

$T(n) = T(\lceil \frac{n}{2} \rceil) + 1 \le c \lg \lceil \frac{n}{2} \rceil + 1 \le c \lg n - d + 1$.

③ Then we just need to prove $\exists\, n_0 > 0$, the induction satisfies the bundary condition.

when $m = n_0$, $T(m) = T(\frac{n_0}{2}\rceil) + 1 \le c \lg \lceil \frac{n_0}{2} \rceil + 1 \le c \lg n_0 - d + 1$

so,

$$c\lg\lceil\frac{n_0}{2}\rceil \le c\lg n_0 - d.$$

$$\lg\lceil\frac{n_0}{2}\rceil \le \lg n_0 - \frac{d}{c} \le 2\lg\lceil\frac{n_0}{2}\rceil - \frac{d}{c}$$

$$\frac{d}{c} \le \lg\lceil\frac{n_0}{2}\rceil$$

$$10^{\frac{d}{c}} \le \lceil\frac{n_0}{2}\rceil$$

$$2\cdot 10^{\frac{d}{c}} < n_0 + 2$$

$$\therefore\ n_0 > 2(10^{\frac{d}{c}} - 1)$$

it is obvious that when the bundary $m = n_0 > 2(10^{\frac{d}{c}} - 1)$, when $c > 0$ and $d \ge 1$, so $T(m) = T(n_0) + 1 \le c\lg n_0 - d + 1$

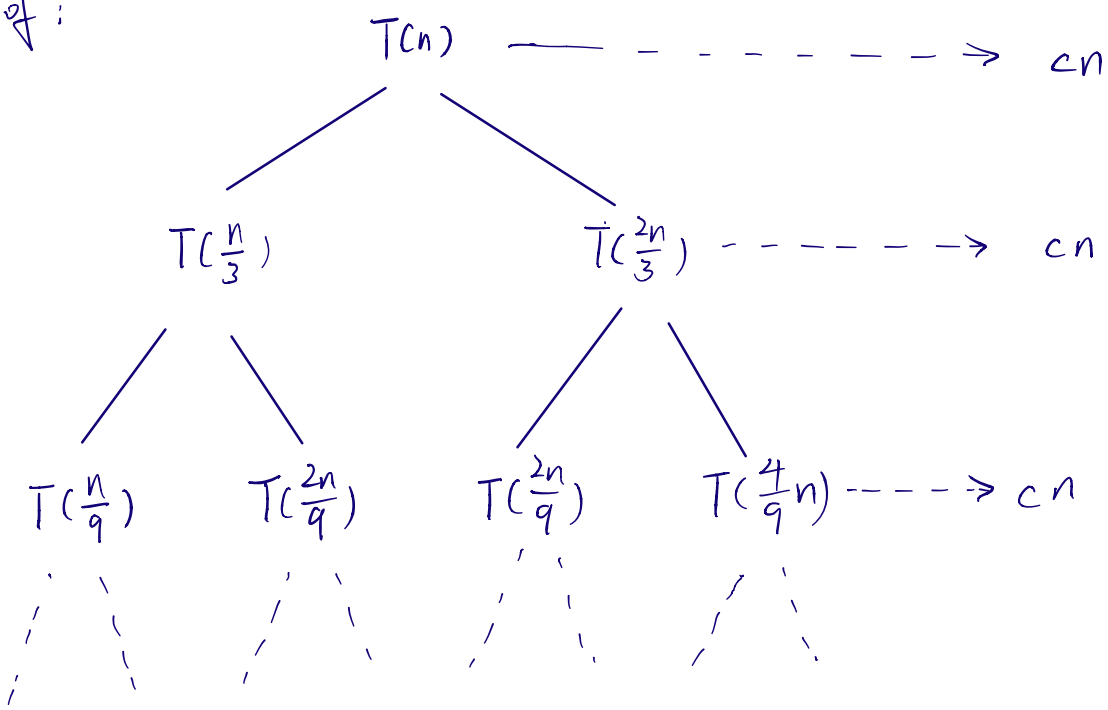
To sum up ① ② ③:

$$T(n) \le T(\lceil\frac{n}{2}\rceil) + 1 \le c\lg n - d + 1 \le c\lg n.$$

$\therefore T(n) = O(\lg n)$ QED.

Question 3.

proof:



the shortest path is $T(n) \to T(\frac{1}{3}n) \to T((\frac{1}{3})^2 n) \to \cdots \to 1$

solve $(\frac{1}{3})^k \cdot n = 1$, $k = \log_{\frac{1}{3}} \frac{1}{n} = \log_3 n$, so the height of the part of the tree in which every node has 2 children is $\log_3 n$. Since the value at each of these levels of the tree add up to cn, the solution of the recurrence is $cn \cdot \log_3 n = \Omega(n \log n)$. QED

Question 4.

master theorem summary : $T(n) = aT(\frac{n}{b}) + f(n)$

case 1 : $n^{\log_b a} = \omega(f(n)) \Rightarrow T(n) = \Theta(n^{\log_b a})$

case 2 : $n^{\log_b a} = o(f(n))$, and $\exists\, c < 1$, $af(\frac{n}{b}) \le cf(n)$

$\Rightarrow T(n) = \Theta(f(n))$

case 3 : $n^{\log_b a} = \Theta(f(n)) \Rightarrow T(n) = \Theta(n^{\log_b a} \log_2 n)$

(a). $T(n) = 4T(\frac{n}{2}) + n$, $a = 4$, $b = 2$, $f(n) = n$.

$n^{\log_b a} = n^{\log_2 4} = n^2 = \omega(f(n))$, $\therefore T(n) = \Theta(n^2)$

(b) $T(n) = 4T(\frac{n}{2}) + n^2$, $a = 4$, $b = 2$, $f(n) = n^2$

$n^{\log_b a} = n^{\log_2 4} = n^2 = \Theta(f(n))$, $\therefore T(n) = \Theta(n^2 \log_2 n)$

(c) $T(n) = 4T(\frac{n}{2}) + n^3$, $a = 4$, $b = 2$, $f(n) = n^3$

$n^{\log_b a} = n^{\log_2 4} = n^2 = o(f(n))$, and solve

$4f(\frac{n}{2}) = 4 \cdot \frac{n^3}{8} = \frac{n^3}{2} \le c \cdot f(n) = cn^3$, $cn^3 \ge \frac{n^3}{2}$, $c \ge \frac{1}{2}$

$\therefore \exists\, \frac{1}{2} \le c < 1$ satisfies case 2 $\therefore T(n) = \Theta(n^3)$

Chapter 3

Question 1.

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]]=C[A[j]]+1
 6  //C[i] now contains the number of elements equal to i.
 7  for i = 1 to k
 8      C[i]=C[i]+C[i-1]
 9  //C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]]=A[j]
12      C[A[j]]=C[A[j]]-1
```

A is the original array, B is the sorted array. It is obvious that the elements in B are set according to the order in A. For example. Let A[j] and A[j+1] have the same value, $1 \le j \le n-1$; and in the final loop, the position of A[j+1] in B is C[A[j+1]], the position of A[j] is C[A[j]] = C[A[j+1]] - 1, which is just before A[j+1]; so the elements with the same values are in the same order in A and B. So it is stable.

Question 2.

Theorem: Given that n d-digit integers in which each digit can take on up to k possible value, then the radix sort algorithm sorts these number in $\Theta(d(n+k))$ time.

When d is a constant and $k = O(n)$, the radix sort costs linear expected running time $T(n) = O(n)$, and it is required the range of input is n^2-1. So, we can difine a n-base number, and then $k = n$, therefore. it is obvious that $d=2$ for 2 dimension being able to cover the range enough. And the inner stable sort althorithm can be counting sort.

So we sort n n-base numbers in the range of n^2 just need expected running time $O(2(n+n))$, satisfying $T(n) = O(n)$.

Question 3.

The worst case running time for bucket-sort occurs when the input does not obey uniformly distribution, instead, all the input will be put in the same bucket, and then the sort algorithm degenerates to insertion sort that needs $O(n^2)$ time.

One simple change to preverse its linear expected running time and make the worst cast running time rise to $O(n\log n)$ is to apply $O(n\log n)$ sort algorithm, like quick sort or merge sort to the sort processing in every bucket. In other word, using bucket sort devides data into different bucket and combining $O(n\log n)$ sort executes inner sort.