Problem 1

(1)

```
                     n
         /   /   /   \   \   \          -
       n/2 n/2 n/2 n/2 n/2 n/2          |
      /                                 |
    n/4 ...                             } log n 層
     .                                  |
    .                                   |
   .                                    -
```

(1%)

$T(n) = n + 3*n + 3^2 n + ... + 3^{(\log n)} n$

$= n(3^{\log n}-1)/(3-1)$

$= \theta(n*3^{(\log n)})$ (3%)

(2)

$n^{(4/5)} = O(n^{(1-\varepsilon)})$ (1%)

$\Longrightarrow T(n) = \theta(n)$ (2%)

(3)

$n^2 = \theta(n^2)$ (1%)

$\Longrightarrow T(n) = \theta(n^2 \log n)$ (2%)

(另外, θ 寫成 O 的扣 1%, 其他情況另計)

Problem 2

(1) not (2) not (3) Correct (4) not (抱歉,在課堂講解期中考解答時有誤)

(5) Correct (6) not (7) Correct

(因爲有 15%,你答對的第一題會有 3%,答對一題以上每題加 2%)

Problem 3

參考課本或講義上的答案.

(有列出第一個式子 3%,全對 10%)

Problem 4

(1) 參考課本或講義上的答案. (分數視情況而定)

(2) optimal -- merge sort, heap sort

non-optimal -- bubble-sort, insertion sort, quicksort

(以上一個 1%,如多寫不符者(不論幾個)扣 1%)

(3)

radix sort + counting sort (2%)

T(n) = O(n) (1%)

input: A[1..n] output: B[1..n] counter: C[1..10]

k =10, d = 新竹新電碼數(可設為 7 碼)

Counting-Sort(A,B,k,p) (p 為要 sort 的電話號碼位置)

```
for i <-- 0 to (k-1) do C[i] <-- 0
for i <-- 1 to n do    C[A[i][p]] <-- C[A[i][p]]+1
for i <-- 2 to k do    C[i] <-- C[i] + 1
for i <-- n downto 1 do
        B[C[A[i][p]] <-- A[i]
        C[A[i][p]] <-- C[A[i]] - 1
```

Radix-Sort (A,B)

```
k <-- 10
for i <-- 1 to d do
      Counting-Sort(A,B,k,i)
      A <-- B                          (2%)
```

(其他非 O(n) 的方法 1%, 加上有詳細步驟者 2%)

Problem 5

divide-and-conquer:

將一個大的問題分成 k 個小的問題去解,一般上問題都分得近乎相等.

.recursive 去解小問題,然後將小問題的解答 combine 成大問題的解答.

用這方法的解答通常是輸入的某種排列.

$T(n) = k\,T(n/k) + g(n)$

g(n) 主要是花在 combine 小問題 return 的解答上.

partition:

與 divide-and-conquer 相似,不過其分出來的小問題一般上都是 independent 的.

$T(n) = k\,T(n/k) + g(n)$

g(n) 主要是花在做 partition 的時間上.

prune-and-search:

這方法也是把大問題切成小問題來解,可是它不會每個小問題都去

recursive 做,而是把答案不可能存在的部份給去掉(prune),然後
繼續在比較小的範圍內 recursive 的去找尋答案.用這方法的答案
通常只是 input set 的一個 sub-set 而已.

$T(n) = T((a/b)n) + g(n) \quad (b>a)$

$g(n)$ 主要是花在尋找不可能的範圍上.

(分數視寫的內容而定)

Problem 6

Greedy algorithm:

假設: 1. 開始時的油量足夠車子走到第一個油站

2. 途中沒有任何兩個相鄰的油站相距超過 n miles

一開始將車子開到最遠所能到達的油站,然後加滿油,繼續開到最遠
所能到達的油站...重覆至到到達目的地為止.

(4%)

prove the correctness:

1. 一開始我們假設車子可以開到第 k 個油站, 則我們說有某個 optimal solution 一定包含第 k 個油站, 若不是, 則假設有一 optimal solution B 它不包含 k , 它一定在 k 之前停下 1 站或以上(因為 k 是它第一次所能到達最遠的點). 那我們可以把這個 optimal solution 解答中在 k 之前的點去掉, 換上 k, 則它還是一個 optimal solution, 因為後者在 k 這點時剛加滿油, 必然比前者的油量多.

(3%)

2. 如果 A 是一個包含 k 的 optimal solution, 則 A' = A-{k} 是不是 S' = { i 屬於 S; i > k) 的一個 optimal solution 呢? 如果不是, 表示有另一個 optimal solution B' for S', 則 B'+{k} 就比 A 還好, 那 A 就不是一個 optimal solution , 矛盾!

(3%)

Problem 7

(1)

$$C[i,j] = \begin{cases} 0 & \text{if } i=m+1 \text{ or } j=n+1 \\ C[i+1,j+1]+1 & \text{if } i<m+1 \text{ and } j<n+1 \text{ and } X_i=Y_j \\ \max(C[i,j+1],C[i+1,j]) & \text{if } i<m+1 \text{ and } j<n+1 \text{ and } X_i<>Y_j \end{cases}$$

(3%)

(2)

LCS-LENGTH(X,Y)

```
m <-- length[X]
n <-- length[Y]
for i <-- 1 to m
    do C[i,n+1] <-- 0
for j <-- 1 to n+1
    do C[m+1,j] <-- 0
for i <-- m to 1
    for j <-- n to 1
        do if Xi = Yj
                then C[i,j] <-- C[i+1,j+1]+1
                else if [i+1,j] >= C[i,j+1]
                        then C[i,j] <-- C[i+1,j]
                        else C[i,j] <-- C[i,j+1]
return C[1,1]
```

(4%)

(3) $T(n) = O(m*n)$

(3%)

Problem 8

```
K-QUANTILE(S,k)
if size(S) = n/k
    then A <-- {max(S)}
else
    q <-- select(S,floor(k/2))
    for i <-- 1 to size(S)
        do if S[i] <= q
            then S1 <-- S1 + {S[i]}
            else S2 <-- S2 + {S[i]}
    A1 <-- K-QUANTILE(S1,floor(k/2))
    A2 <-- K-QUANTILE(S2,ceil(k/2))
A <-- A1 + A2 - {max(S)}
return A
```

$T(n) = 2T(n/2) + O(n)$

recursive until $T(n/2^x) = T(n/k)$

then use $O(n/k)$ to find the max num

==> $n/2^x = n/k$

$x = \log k$

$T(n) = O(n) * \log k + k * O(n/k)$ (只算到 k 個)

$= O(n \log k)$

(註: 這樣其實有多算一個,就是 S 中的最大值,output 時去掉即可)

(分數視情況而定)

Bonus:

(1) 對 tree 做 DFS, 同時 maintain 兩個 stack, 一個存放目前走到的 node 到 root 的 path 上所有的 nodes, 另一個存放 path 上每一點到 root 的距離. DFS 到一個 node 時, 先把自己的 node number 加到第一個 stack, 然後讀取第二個 stack 最上面的值加上前一個 node 到自己的 length(總和是自己到 root 的距離), push 到 第二個 stack 上. 最後用 binary search 對第二個 stack 找出中點的 node 或是某條 edge 而 output 某個 node 或夾住 edge 的兩個 nodes. 退回時則 pop 掉這兩個 stack 的最上一個 element.

$T(n) = O(|E| + |V|) * \log n$ (binary search 為 $\log n$)

$= O(n-1 + n) * \log n$

$= O(n \log n)$

(Time complexity 不是 $n \log n$ 且結果正確者 3%, 是 $n \log n$ 者 5%)

(2) 如果 edge 的 weight 是 1 則不用第二個 stack, 直接用 index 去 access bisector node(s) 即可.

$T(n) = O(n)$

(非 O(n) 但有比 (1) 好者 3%, 是 O(n) 者 5%)