1. [5 pts] State whether each of the following is **true or false, Why?**

$5^{\frac{3n}{2}} = O(5^n)$

$5^{(3n/2)} <= c * 5^n$

$5^n * 5^{n/2} <= c * 5^n$

要找到 c, n0, 令 n >= n0 所有都成立

> 無法找到 n0 令所有 n >= n0 都成立

> 所以 false

$5^{\frac{3}{2}n} = 5^n \times 5^{\frac{1}{2}n} \quad O(5^n)$

$\lim_{n \to \infty} \frac{5^n \times 5^{\frac{1}{2}n}}{5^n} = \infty \to O(5^n)$ False

$$f(n) = O(g(n)) \implies \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

$$f(n) = \Omega(g(n)) \implies \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

$$f(n) = \Theta(g(n)) \implies 0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

2. [5 pts] Solve the recurrence and represented with *asymptotic notation*, T(*1*) = *c* and T(*n*) = 10T(*n/5*) +Θ(*n*)

a = 10, b = 5

Root =n, Leaf = $n^{log^{10}_5}$

>$f(n)/n^{log^a_b}$ = $n/n^{log^{10}_5}$=$n^{1-log^{10}_5}$ =0

>$f(n) << n^{log^a_b}$ >T(n) = Θ($n^{log^{10}_5}$)

$T(n) = 10T(n/5) + \Theta(n)$

$= 10\left[10T\left(\frac{n}{5^2}\right) + \Theta\left(\frac{n}{5}\right)\right] + \Theta(n)$

$= 10^2 T\left(\frac{n}{5^2}\right) + 10\Theta\left(\frac{n}{5}\right) + \Theta(n)$

$= 10^2\left[10T\left(\frac{n}{5^3}\right) + \Theta\left(\frac{n}{5^2}\right)\right] + 10\Theta\left(\frac{n}{5}\right) + \Theta(n)$

$= 10^3 T\left(\frac{n}{5^3}\right) + 10^2\Theta\left(\frac{n}{5^2}\right) + 10\Theta\left(\frac{n}{5}\right) + \Theta(n)$

$= 10^k T\left(\frac{n}{5^k}\right) + 10^{k-1}\Theta\left(\frac{n}{5^{k-1}}\right) + \cdots 10\Theta\left(\frac{n}{5}\right) + \Theta(n)$

令 $\frac{n}{5^k} = 1 \quad k = \log_5 n$

$\Rightarrow 10^{\log_5 n} \cdot c + 10^{\log_5 n - 1} \times 5\Theta\left(\frac{n}{5^{\log_5 n}}\right) + \cdots + 10\Theta\left(\frac{n}{5}\right) + \Theta(n)$

$= 10^{\log_5 n} + 10^{\log_5 n - 1}$

$= \underline{n^{\log_5 10}} + \frac{1}{10} \cdot \underline{n^{\log_5 10}} + \cdots + \Theta(n)$

$\Theta(n^{\log_5 10})$?

3. [10 pts] Compute *time complexity* of the following algorithm:

(Need to show the recurrence of time complexity and solve it)

```
sum (A, p, r) {
    if p = r
        return A[p]
    else {
        k = ⌊(p + r) / 2⌋
        return sum(A, p, k) + sum(A, k+1, r)
    }
}
```

* a 是看他分成幾個（sum1+sum2）
* b 是看他每一個是分多少(k)

T(n) = 2T[n/2] + Θ(1)
a = 2, b = 2
Root = 1, Leaf = n
>Root/Leaf = 1/n = 0
>Θ(n)

4. [10 pts] A teacher has sorted the exam sheets of all the students at a class on their student ID number. Now he would like to take that list and sort by exam score. However, he wishes to be certain that if there are several students same score, at the end of the sort by score, these students are still sorted by student ID number. **What sorting property is required for the algorithm and what is the time complexity of the algorithm?**

1. 該演算法必須具備穩定性，意指有多個元素具有相同的排序鍵，它們的相對順序在排序後仍然保持不變。
2. 選一個適合的演算法並敘述時間複雜度。

	Worst Case	Best Case	Stable? (Yes/No)
Selection sort	$\Theta(N^2)$	$\Theta(N^2)$	No
Insertion sort	$\Theta(N^2)$	$\Theta(N)$	Yes
Merge sort	$\Theta(N\log N)$	$\Theta(N\log N)$	Yes
Quicksort	$\Theta(N^2)$	$\Theta(N\log N)$	No
Heapsort	$\Theta(N\log N)$	$\Theta(N)$	No

count sort   stable
bubble sort   stable

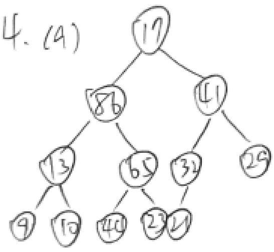
5. [10 pts] Using MAX-HEAPIFY algorithm,

(a) Show the operation on the array:

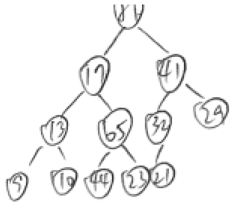
A= [17, 86, 41, 13, 65, 32, 29, 9, 10, 44, 23, 21]

(b) Show the worst-case running time.

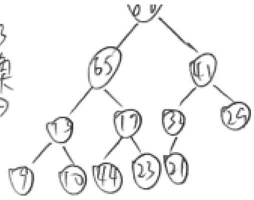
4. (a)



∵第1,2,3層
已是MAX HEAP
∴從第0層開始
86>17>41,
所以86,17交換

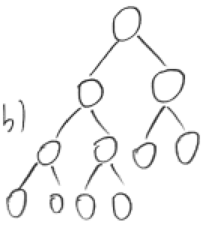


∵65>17>13
∴65,17交換



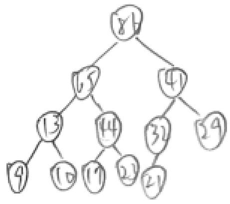
∵44>23>17
∴44,17交換

(b)



左圖為worst-case，樹的最後一層為半滿狀態。
右節點數為3 設為k，左節點數為7 = 2k+1
總節點數為 k+2k+1+1 = 3k+2
左節點數佔總節點數比率為 $\frac{2k+1}{3k+2}$

$\lim_{k\to\infty}\frac{2k+1}{3k+2} = \frac{2}{3} \Rightarrow$ if total number of nodes is n,
worst-case running time is $\frac{2n}{3}$ #



#

6. [10 pts] Design and analysis a variant QUICKSORT algorithm, which runs $O(n \lg n)$ in **worst-case**.

6. [10 pts] Design and analysis a variant QUICKSORT algorithm, which runs $O(n \lg n)$ in **worst-case**.

用select 演算法的前三步找近似中位數的數字，函式取名叫select_v2

Step 1 : 將陣列分成n/5個group ,每個group有5個元素，最後一個group可以小於5個元素
Step 2 : 用insertion sort 排序,找出每個group的中位數
Step 3 : 遞迴呼叫select_v2 來找出n/5個中位數之中的中位數

找到中位數後，將pivot=中位數，進行QuickSort

Ans : 使用 select algorithm 前三步驟，每次 pivot 都選到中位數，然後再進行 quicksort 。時間複雜度為 lg n (選中位數) * n(快速排序)。

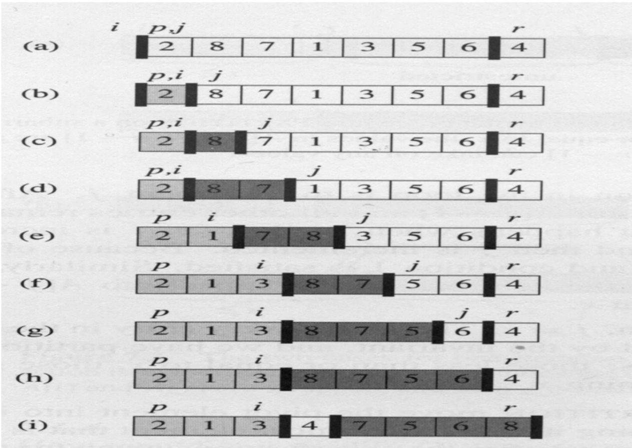
7. [10 pts] For the following input array A = [10, 80, 30, 90, 40, 50, 70], show the operation of PARTITION on the array A.

```
PARTITION(A, p, r)
    x ← A[r]
    i ← p − 1
    for j ← p to r − 1
         do if A[j] ≤ x
               then i ← i + 1
                    exchange A[i] ↔ A[j]
    exchange A[i + 1] ↔ A[r]
    return i + 1
```

i j x
10, 80, 30, 90, 40, 50, 70
$\downarrow$ 10 < 70
i j x
10, 80, 30, 90, 40, 50, 70
$\downarrow$ 80 > 70
i j x
10, 80, 30, 90, 40, 50, 70
$\downarrow$ 30 < 70
i j x
10, 30, 80, 90, 40, 50, 70
$\downarrow$ 90 > 70
i j x
10, 30, 80, 90, 40, 50, 70
$\downarrow$ 40 < 70
i j x
10, 30, 40, 90, 80, 50, 70
$\downarrow$ jump loop
10, 30, 40, 70, 80, 50, 90

return 3

The Operation of PARTITION



[10 pts] For the **SELECT** algorithm in section 9.3,

(a) Revise the **SELECT** algorithm to find the *median*.

(b) What is the running time of the algorithm? (Show the time complexity T (*n*) in recurrence, solve the recurrence and represented with *asymptotic notation*.)

```
3. SELECT_RE (A, start, end)
(a) if A < 5
        return  A[start + end/2]
    else
        num = ⌊len(A)/5⌋
        last_mid = ⌊(len(A) % 5)/2⌋
        for i = 1 to num
            new = A[1+5(i-1) : 5i]
            insertion_sort(new)
            midlist[i] = new[3]
        lastlist = A[5×num+1 : end]
        insertion_sort(lastlist)
        midlist[num+1] = lastlist[last_mid]
        SELECT_RE (midlist, start, end)
```

(b) $T(n) = T(\lfloor n/5 \rfloor) + O(n)$ $\quad f(n) \in O(n)$

$a = 1$ $\quad b = 5$ $\quad n^{\log_b a} = n^{\log_5 1} = 1$

$1 f(n/5) \le c f(n)$, $\quad \frac{n}{5} \le cn$

(regularity condition) $\quad c = \frac{1}{5}$

$T(n) \in \Theta(n)$

P, NP, and NP-complete problems

- Class **P**: **decision problems** (contrast to optimization problem) that are *solvable* in polynomial time, i.e. $O(n^k)$, a.k.a. *tractable*
- Class **NP** (Nondeterministic Polynomial): decision problems that are *verifiable* in polynomial time if you are given a *certificate* of a solution
- **Class NP-complete**
  - a subset of NP, whose status is unknown
  - no polynomial-time algorithm has yet been discovered
  - nor has anyone yet been able to prove that no polynomial-time algorithm can exist for any one of them
  - at least as *hard* as any problem in NP


```
QUICKSORT(A, p, r)
  if p < r
     then q ← PARTITION(A, p, r)
          QUICKSORT(A, p, q − 1)
          QUICKSORT(A, q + 1, r)
```

二分搜尋的虛擬碼大致如下：

```
void binarysearch(Type data[1..n], Type search)
{
    Index low = 1;
    Index high = n;

    while (low <= high)
    {
        Index mid = (low + high) / 2;

        if (data[mid] = search)
        {
            print mid;
            return;
        }
        else if (data[mid] > search)
        {
            high = mid - 1;
        }
        else if (data[mid] < search)
        {
            low = mid + 1;
        }
    }

    print "Not found";
}
```

若是有 n 筆資料，在最差的情況下，二分搜尋法總共需要比較 **[$\log_2 n$] + 1** 次。

顯然的，此種搜尋法較循序搜尋法(linear search)快速許多。

	worst case	best case	in place	stable
Insertion	N^2	N	Yes	Yes
selection	N^2	N^2	Yes	No
merge	NlogN	NlogN	No	Yes
bubble	NlogN	N	Yes	Yes
heap	NlogN	N	Yes	No
quick	N^2	NlogN	Yes	No
count	N	N	No	Yes
bucket	N^2	N	老師沒特別提	

- 樹的高度：$log_b n$
- 葉節點的個數：$a^{log_b n} = n^{log_b a}$

Case 1：If $f(n) = O(n^{log_b a - \varepsilon})$, for some constant ε > 0, then $T(n) = \theta(n^{log_b a})$

Case 2：If $f(n) = \theta(n^{log_b a})$, then $T(n) = \theta(n^{log_b a} lgn)$

Case 3：If $f(n) = \Omega(n^{log_b a + \varepsilon})$, for some constant ε > 0, and if $af(n/b) \leq cf(n)$ for some constant c < 1 and all sufficiently large n, then $T(n) = \theta(f(n))$
（中間紅色部分是為了證明 Regularity Condition (正定性)）

題目二：

$$T(n) = 5T(n/3) + n^2$$

Sol： $a=5$, $b=3$ Root: $n^{\log_3 5}$ $f(n) = n^2$

$\frac{n^2}{n^{\log_3 5}} = n^{\overset{(0.6..)}{2-\log_3 5}} \longrightarrow \infty$, $\log_3 5 \approx 1.4649....$

⇒ Case 3,再多證明它的 Regularity condition

$a f(\frac{n}{3}) \leq c \cdot f(n)$, (要找到那個c是存在的。)

⇒ $5 \cdot f(\frac{n}{3}) = 5 \cdot (\frac{n}{3})^2 \leq c \cdot n^2$

⇒ $5 \cdot \frac{n^2}{9} \leq c \cdot n^2$

⇒ $\frac{5}{9} \leq c$

符合定義 c < 1 and all sufficiently large n,

then $T(n) = \Theta(f(n)) = \Theta(n^2)$

Case 3比較麻煩一點，還需要多證明一個Regularity Condition

題目一：

$$T(n) = 3T(n/9) + \sqrt{n}$$

Sol:

$\boxed{\text{STEP 1}}$ $a=3$, $b=9$

Root: $n^{\log_b a} = n^{\log_9 3}$

Leaf: $f(n) = \sqrt{n}$

$\boxed{\text{STEP 2}}$ $\frac{f(n)}{n^{\log_b a}}$

$\frac{f(n)}{n^{\log_b a}} = \frac{n^{\frac{1}{2}}}{n^{\log_9 3}} = \frac{n^{\frac{1}{2}}}{n^{\frac{1}{2}}} = 1$ (const)

屬於 Case 2 情況，代入公式.

$T(n) = \theta(n^{\log_9 3} \lg n) = \theta(\sqrt{n} \lg n)$

題目三：

$$T(n) = 5T(n/2) + n^2$$

Sol: $a=5$, $b=2$ Root: $n^{\log_b a} = n^{\log_2 5}$ Leaf: n^2

代入 $\frac{f(n)}{n^{\log_b a}} = \frac{n^2}{n^{\log_2 5}} = n^{2-\log_2 5} \longrightarrow 0$, $n^{\log_2 5} \approx n^{2.321}$

Case 1 ⇒ $T(n) = \theta(n^{\log_b a}) = \theta(n^{\log_2 5})$

成長率大小，上面最大到下面最小。

$2^{2^{n+1}}$

2^{2^n}

$(n+1)!$

$n!$

e^n

$n \cdot 2^n$

2^n

$(3/2)^n$

$n^{\lg \lg n} = (\lg n)^{\lg n}$

$(\lg n)!$

n^3

$n^2 = 4^{\lg n}$

$\lg(n!) \approx n \lg n$

$2^{\lg n} = n$

$(\sqrt{2})^{\lg n} = \sqrt{n}$

$2^{\sqrt{2 \lg n}}$

$\lg^2 n$

$\ln n$

$\sqrt{\lg n}$

$\ln \ln n$

$2^{\lg^* n}$

$\lg^* n \approx \lg^*(\lg n)$

$\lg(\lg^* n)$

$n^{1/\lg n} = 1$

```
INSERTION-SORT(A)
for j ← 2 to length[A]
     do key ← A[j]
          ▷ Insert A[j] into the sorted sequence A[1 .. j − 1].
          i ← j − 1
          while i > 0 and A[i] > key
               do A[i + 1] ← A[i]
                    i ← i − 1
          A[i + 1] ← key
```

Loop invariants and the correctness of insertion sort

```
MERGE(A, p, q, r)
n1 ← q − p + 1
n2 ← r − q
create arrays L[1 .. n1 + 1] and R[1 .. n2 + 1]
for i ← 1 to n1
     do L[i] ← A[p + i − 1]
for j ← 1 to n2
     do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
     do if L[i] ≤ R[j]
           then A[k] ← L[i]
                i ← i + 1
           else A[k] ← R[j]
                j ← j + 1
```

```
MERGE-SORT(A, p, r)
if p < r
   then q ← ⌊(p + r)/2⌋
        MERGE-SORT(A, p, q)
        MERGE-SORT(A, q + 1, r)
        MERGE(A, p, q, r)
```

```
MAX-HEAPIFY(A, i)
l ← LEFT(i)
r ← RIGHT(i)
if l ≤ heap-size[A] and A[l] > A[i]
   then largest ← l
   else largest ← i
if r ≤ heap-size[A] and A[r] > A[largest]
   then largest ← r
if largest ≠ i
   then exchange A[i] ↔ A[largest]
        MAX-HEAPIFY(A, largest)
```

$T(n) = O(\lg n)$

```
BUILD-MAX-HEAP(A)
heap-size[A] ← length[A]
for i ← ⌊length[A]/2⌋ downto 1
     do MAX-HEAPIFY(A, i)
```

$T(n) = O(n)$

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i ← length[A] downto 2
     do exchange A[1] ↔ A[i]
        heap-size[A] ← heap-size[A] − 1
        MAX-HEAPIFY(A, 1)
```

COUNTING-SORT(A, B, k)

```
1  for i ← 0 to k
2      do C[i] ← 0
3  for j ← 1 to length[A]
4      do C[A[j]] ← C[A[j]] + 1
5  ▷ C[i] now contains the number of elements equal to i.
6  for i ← 1 to k
7      do C[i] ← C[i] + C[i − 1]
8  ▷ C[i] now contains the number of elements less than or equal to i.
9  for j ← length[A] downto 1
10     do B[C[A[j]]] ← A[j]
11        C[A[j]] ← C[A[j]] − 1
```

Counting - Sort

inplace X → 空間換時間

stable O

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

6~8

~~2~~ 2 ~~4~~ 7 7 8

1 3 6 7

0 5

4

9~11

B[7]=3

B[2]=0

B[6]=3

B[4]=2

B[1]=0

B[5]=3

B[8]=5

B[3]=2

COUNTING-SORT(A, B, k)

```
1  for i ← 0 to k
2      do C[i] ← 0
3  for j ← 1 to length[A]
4      do C[A[j]] ← C[A[j]] + 1
5  ▷ C[i] now contains the number of elements equal to i.
6  for i ← 1 to k
7      do C[i] ← C[i] + C[i − 1]
8  ▷ C[i] now contains the number of elements less than or equal to i.
9  for j ← length[A] downto 1
10     do B[C[A[j]]] ← A[j]
11        C[A[j]] ← C[A[j]] − 1
```

```
PARTITION(A, p, r)
x ← A[r]
i ← p − 1
for j ← p to r − 1
     do if A[j] ≤ x
          then i ← i + 1
               exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

```
QUICKSORT(A, p, r)
if p < r
   then q ← PARTITION(A, p, r)
        QUICKSORT(A, p, q − 1)
        QUICKSORT(A, q + 1, r)
```

SELECT (A, start, end, i) /* i is the ith order statistic. */

1. divide input array A into $\lfloor n/5 \rfloor$ groups of size 5 (and one leftover group if n % 5 != 0)
2. find the median of each group of size 5 by insertion sorting the groups of 5 and then picking the middle element.
3. call Select recursively to find x=A[k], the median of the $\lceil n/5 \rceil$ medians.
4. partition array around x, splitting it into two arrays A[start, pivot-1] and A[pivot+1, end]
5. if (i = k) return x
   else if (i < k) then /* like Randomized-Select */
   call SELECT (A, start, pivot-1, i)
   else call SELECT (A, pivot+1, end, i - (pivot - start + 1))

<u>Running Time (each step):</u>

1. O(n) (break into groups of 5)
2. O(n) (sorting 5 numbers and finding median is O(1) time)
3. $T(\lceil n/5 \rceil)$ (recursive call to find median of medians)
4. O(n) (partition is linear time)
5. $T(7n/10 + 6)$ (maximum size of subproblem)

$$T(n) = T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n)$$