# Midterm Solution

1. (解釋和舉例各佔兩分)

   An abstract data type consists of one or more object **domains** and **operations** on elements from the domain. An abstract data type can be implemented by means of a data structure.

   For example, stack is a kind of abstract data type, and data structures such as array or linked-list can be used to implement stack.

2.

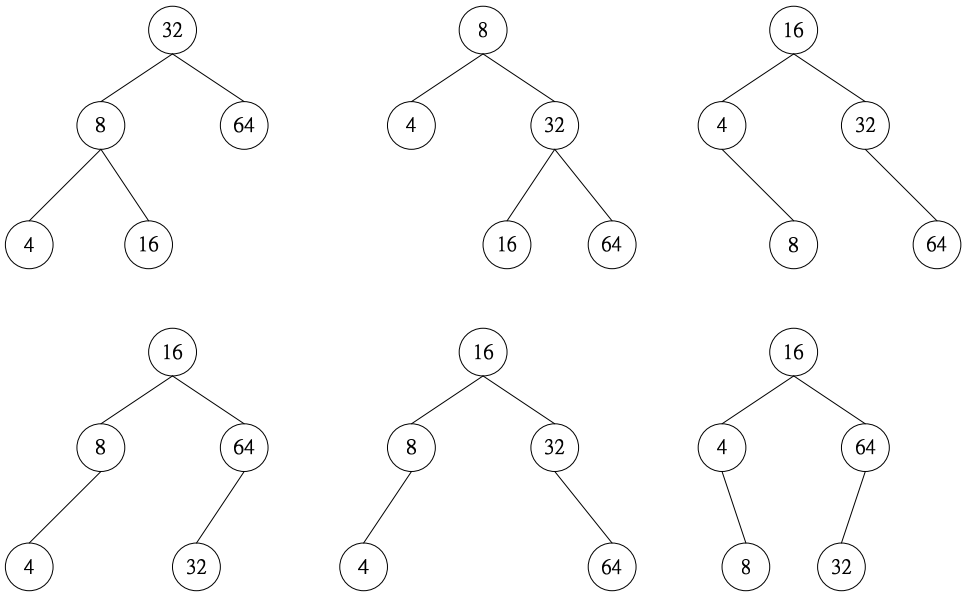
| Implementation | Push | pop |
|---|---|---|
| 1 | $\theta(n)$ | $\theta(n)$ |
| 2 | $\theta(1)$ | $\theta(1)$ |
| 3 | $\theta(1)$ | $\theta(1)$ |
| 4 | $\theta(1)$ | $\theta(n)$ |

   ps. Implementation 2 可以是 Push=Pop=$\theta(1)$ 或 Push=Pop=$\theta(n)$。(但不會是一個 $\theta(1)$ 一個 $\theta(n)$ )

   ps2. implementation 4 的 pop 是 $\theta(n)$，因為 pop 後想更新 tail pointer 得要 traverse 整個 list。

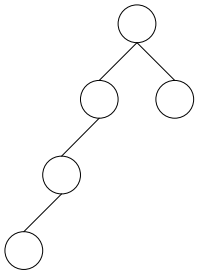
3. $1^2+2^2+....+i^2=\sum_{0}^{n-1} i^2=\frac{n(n+1)(2n+1)}{2}=O(n^3)$

4.




5.

   1.

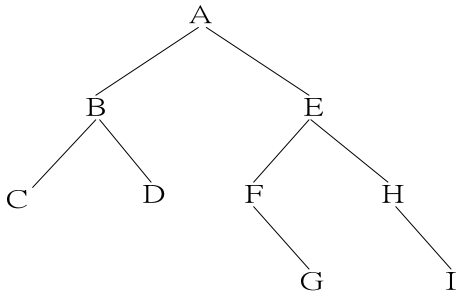


2. **minimum** height 是出現在 complete binary tree 的狀況下，tree height 為 $\theta(\log n)$ 。

   **maximum** height 是出現在其中一邊 (以及當中較高的 subtree) node 總數多達 $\frac{2}{3}$ ，所以 n 個 nodes 的最大樹高為 $h(n)=1+h(\left\lfloor \frac{2}{3}n \right\rfloor)$ 。已知 $h(0)=0$ ，解 recursive relation，可知 $h(n)=\theta(\log_{\frac{2}{3}} n)=\theta(\log n)$ 。

   所以樹高為 $\theta(\log n)$ 。

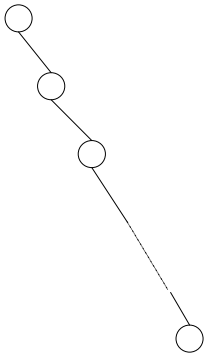
6.




7. (評分：全對不扣分。方向正確但細節說明不足，酌扣 1~3 分。方向不對且其中有明顯錯誤者，酌扣 5~9 分。)

   Proof outline:

   1. 證明：任何一個由 n 個 nodes 所組成的 binary tree 可以在 n-1 個 single rotations 內轉變成下圖這個完全斜向右側的 tree：

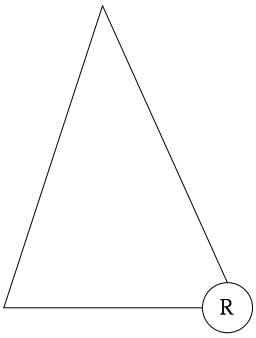


   2. 因為 $T_1$ 變成上圖需要**不到 n-1 個** single rotations、同樣 $T_2$ 變成上圖也要**不到 n-1 個** single rotations，且 single rotation 為 reversible operation，所以由 $T_1$ 變成上圖、再由上圖變成 $T_2$，總共需要不到 2(n-1) 個的 single

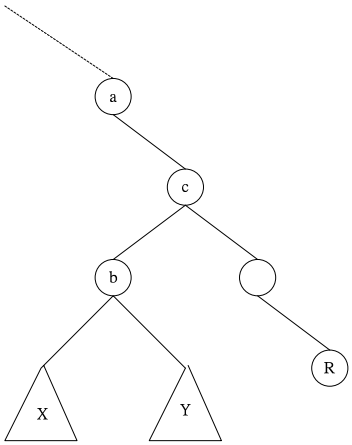
rotations。

**證明**：任意一個由 n 個 nodes 所組成的 binary tree 可以在 n-1 個 single rotations 內轉變成完全斜向右側的 tree：

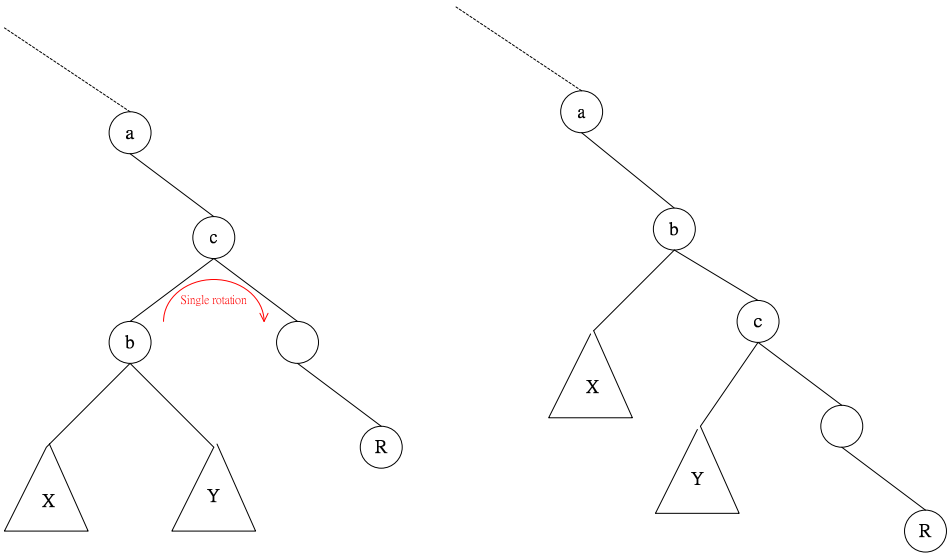
i. 對於此 binary tree，從 root 沿著 right child pointer 找到最右下角的 node：



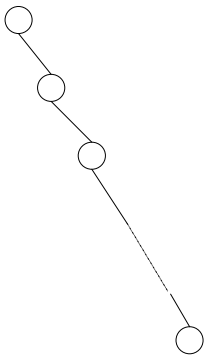

ii. 朝左上往回找，直到遇到第一個有 left child 的 node：



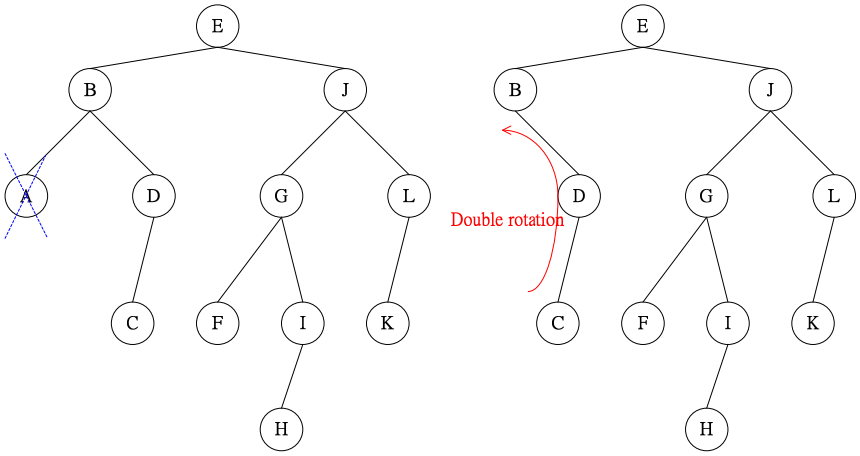

iii. 將此 node 之 left subtree 的 root 給 single rotate 進由 root 沿著 right child pointer 一路下來所形成的 path 上：

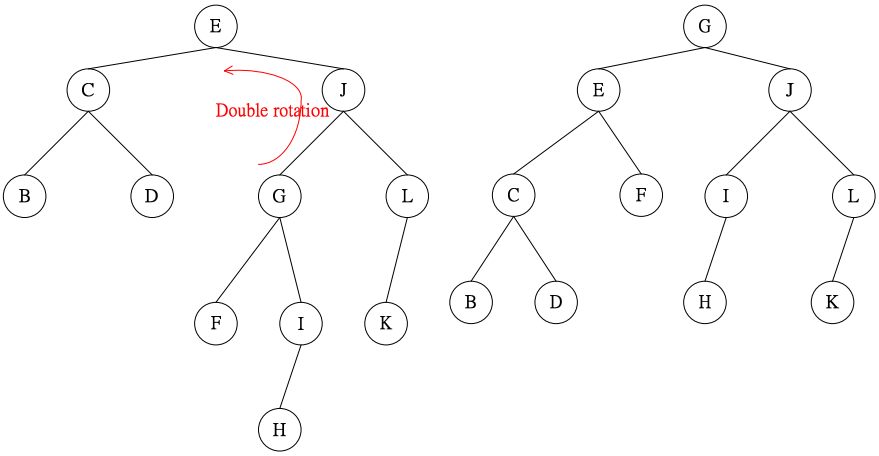


重覆 i~iii 的動作，在 n-1 次 single rotation 內便可把所有的 node 納入由 root 沿著 right child pointer 一路下來所形成的 path 上：

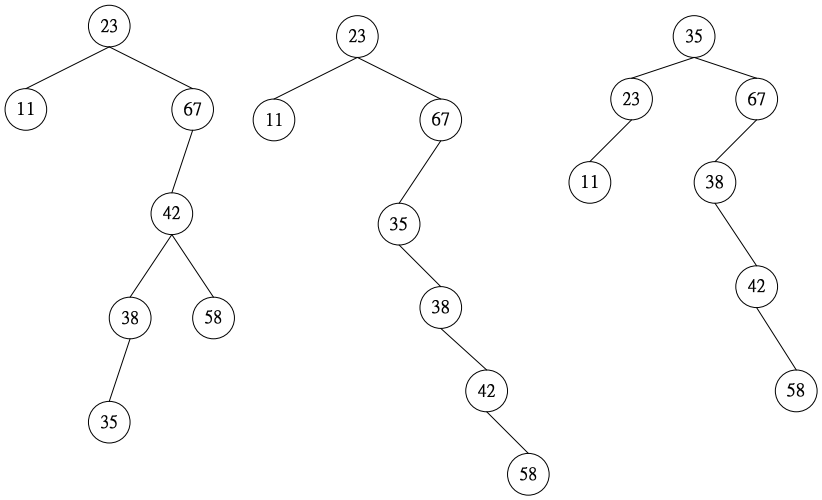


8.

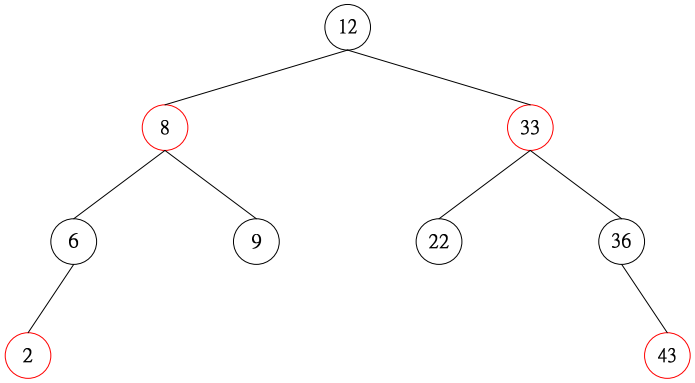


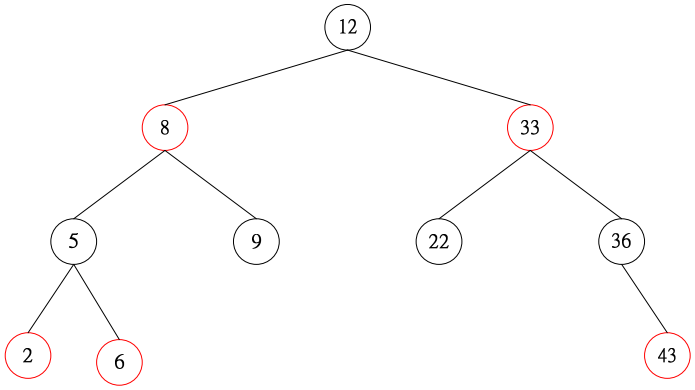




9.




10.




11.

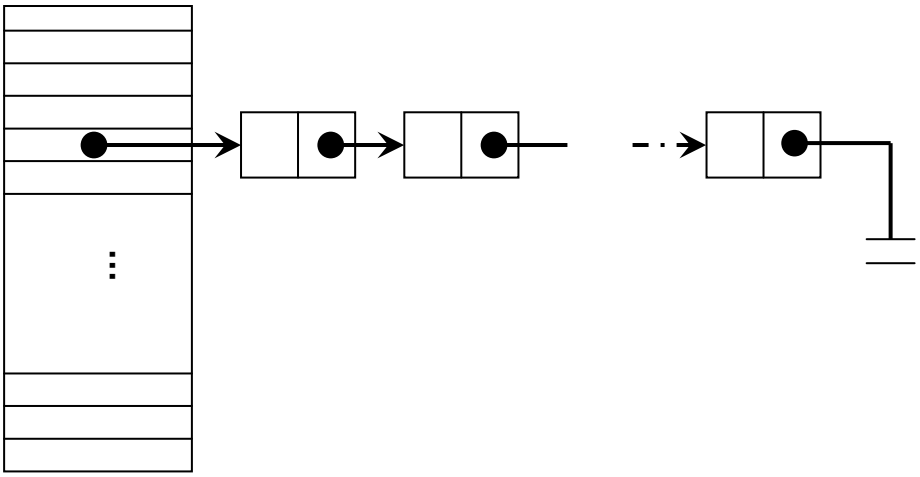
1.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 6 | | | | | | 8 | 1 | 16 | 5 | 13 |

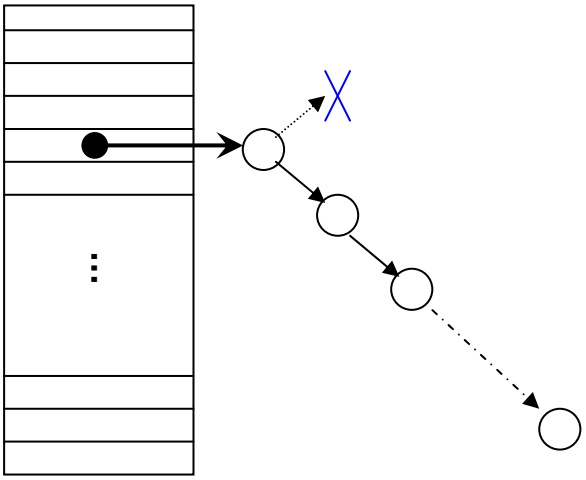
2. (評分：每一小題 3 分。其中 complexity order 佔一分，解釋的部份佔兩分。)

   i. worst-case 發生在當 n 個 operations 都發生在同一個 hash table cell

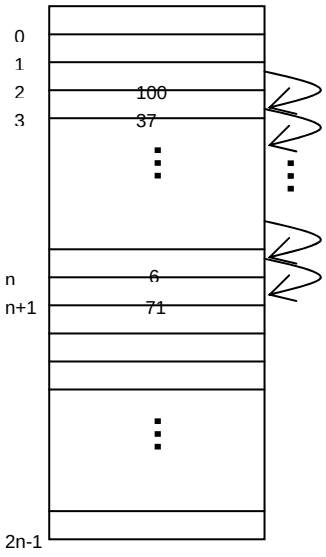
上；此時一次 insertion 為 $\theta(1)$ 但一次 query 為 $\theta(n)$ ，所以要是 $\frac{n}{2}$ 個 insertion 後接著 $\frac{n}{2}$ 個 query，那麼 n 個 operations 所造成的複雜度為 $n \times \theta(n) = \theta(n^2)$ 。



ii. worst-case 發生在當 n 個 operations 都發生在同一個 hash table cell 上、且在這 cell 上所形成的 binary tree 是個完全不 balanced 的 tree，變得像一個 linked-list 一樣。此時一次 insertion 或 query 均為 $\theta(n)$。同 i，n 個 operations 所造成的複雜度為 $n \times \theta(n) = \theta(n^2)$



iii. worst-case 發生在當 n 個 operations 都由 hash function 得到同一個 key、結果造成 clustering。此時 insertion 和 query 都是 $\theta(n)$ ，n 個 operations 所造成的複雜度為 $n \times \theta(n) = \theta(n^2)$


0
1
2
3
100
37
n
n+1
6
71
2n-1