《資料處理》

試題評析	今年的資料處理試題整體風格明顯回歸傳統，題型設計以基本與經典觀念為主軸，整體難度偏易到中等，主要考驗考生是否熟悉教材核心內容。第一題與第三題皆為純記憶性的定義題與資料庫正規化題型，只要讀過講義即可順利作答，並未加入情境或其他變化；第二題雖為實作題，但題目涵蓋的二元樹、堆積與運算式轉換等皆屬於標準資料結構題，作答步驟固定、可預測性高，因此難度不大，唯第四題程式遞迴追蹤題，題目概念雖有趣且能有效測試考生程式邏輯，但題目未清楚指定語言且存在型態轉換的瑕疵，使得實際答案在不同語言中可能不一致，是本次命題中較具爭議的一點。整體而言，整份考卷若命題偏向穩定、保守，對熟讀教科書與講義的考生相當有利，考生未來準備此科目時，仍應將基礎理論、標準算法與核心定義視為最重要的得分關鍵。

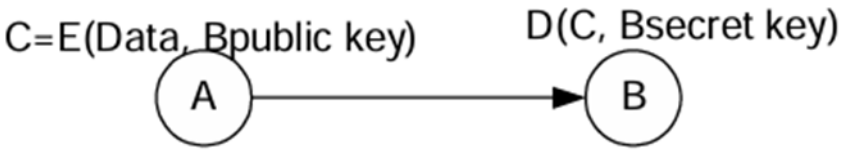
一、在資訊安全中，雜湊函數（Hash Function）與數位簽章（Digital Signature）常被用來確保資料的正確與完整。請敘述兩者的功能，並說明雜湊函數的主要特性及數位簽章的運作過程。（25分）

試題評析	本題為資料處理中記憶類型考題，題目沒有額外變化，因此僅需將教科書內的雜湊、數位簽章定義寫出即可，本題可以搭配作圖回答，以適當擴充篇幅。
考點命中	《高點・高上資料處理講義》第二回，JAMES編撰，頁39、150。

答：

1.雜湊法(Hasing)：利用雜湊函數(hash function)將每筆紀錄之鍵值(key value)或雜湊欄位(hash field)對應至相對磁碟儲存位址，插入資料時將雜湊欄位(鍵值)帶入雜湊函數h(x)，轉換成儲存位址，再將資料插入於此位址；若此位址先前已有資料，則採用雜湊碰撞解決方法。

2.數位簽章(Digital signature)：數位簽章則是利用雜湊函數先將傳送資料轉成摘要，再用發送者的「私密金鑰」加密該摘要並附在傳送資料中，使接收者能以傳送者之「公開金鑰」驗證來源身分與內容完整性的技術。



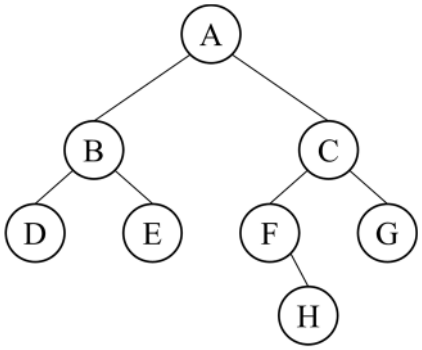
(圖：數位簽章示意圖，圖中E與D代表加密與解密，收送兩端將使用相同的雜湊函數進行)

二、請完成下列各小題，內容包含運算式轉換、樹狀結構走訪與最小堆積樹（Min Heap），請寫出詳細步驟或畫出結果。（25分）

⑴將下列運算式由中序式（Infix）轉換為前序式（Prefix）：

(A-B)*(C+D)/F

⑵根據下列二元樹，寫出其後序（Postfix）走訪結果：



⑶依序將數字12, 8, 20, 4, 15, 7, 3, 10插入一個空的最小堆積樹，畫出最後的堆積樹。
⑷承上題，刪除最小數字3後的最小堆積樹，畫出其最後結果。

試題評析	本題分四小題，皆為實作類型考題，與多數實作類型考題不同，本題並未出現過多變化，主要都集中於傳統計算機概論、資料結構常出現的考點，例如：二元樹、算術運算式轉換、堆積等，只要記得方法，實際作答難度應不高。
考點命中	《高點・高上資料處理講義》第二回，JAMES編撰，頁7~17。

答：

(1)中序轉換前序

快速演算步驟說明：

步驟1：依照運算子優先順序加上括號

步驟2：從最內層括號開始，把表示法轉成前序

獲得輸出結果

本題結果：

Step 1: $((A-B)*(C+D))/F$

=Step 2: $((-AB)*(+CD))/F$

=Step 3: $(*-AB+CD)/F$

=Step 4: $/*-AB+CDF$ (End)

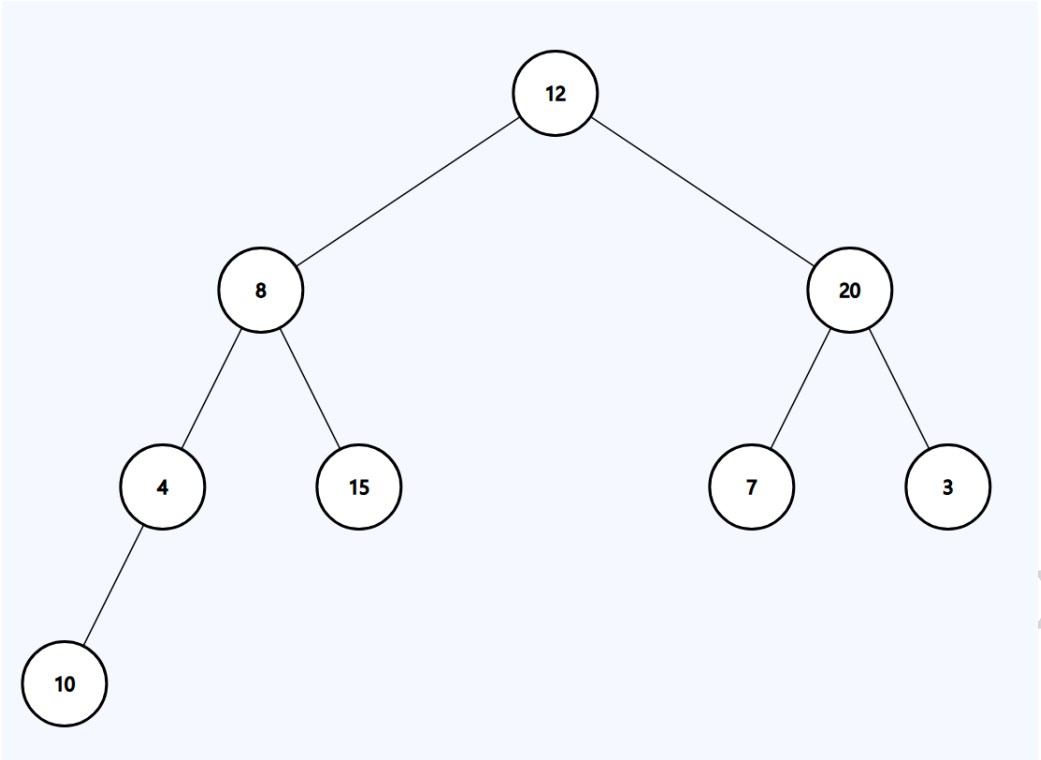
(2)二元樹

後序走訪口訣：Left → Right → Root (先左再右，樹根最後)

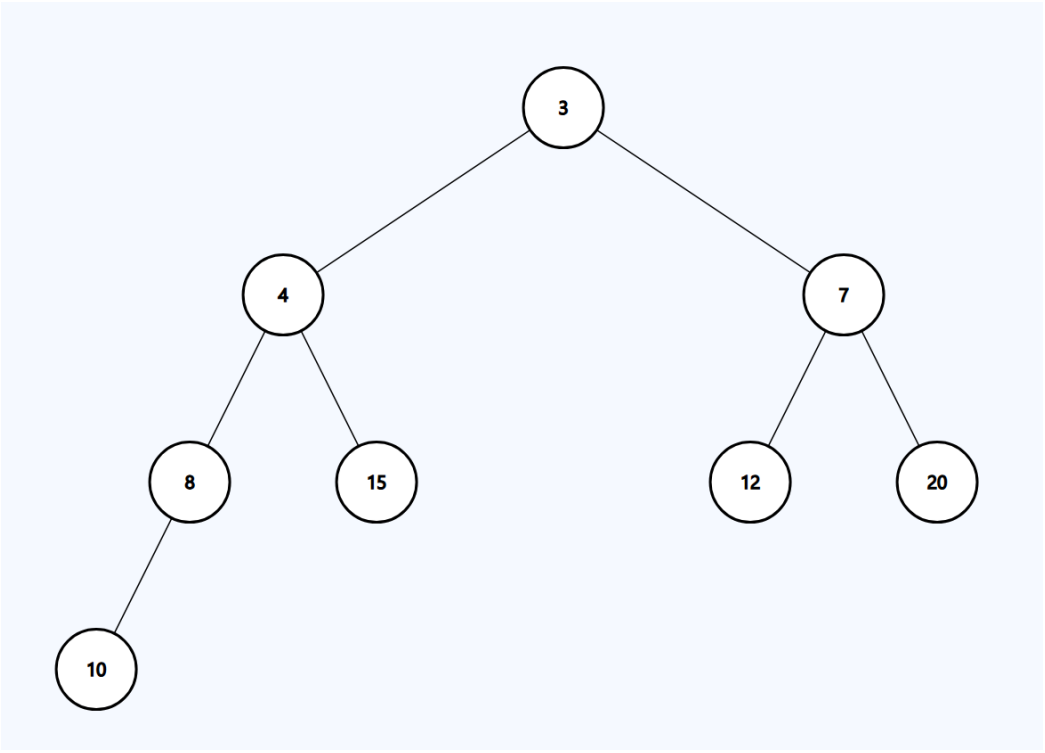
Postorder Traversal結果：[D,E,B,H,F,G,C,A]

(3)最小堆積

轉換為堆積前之Complete Binay Tree：

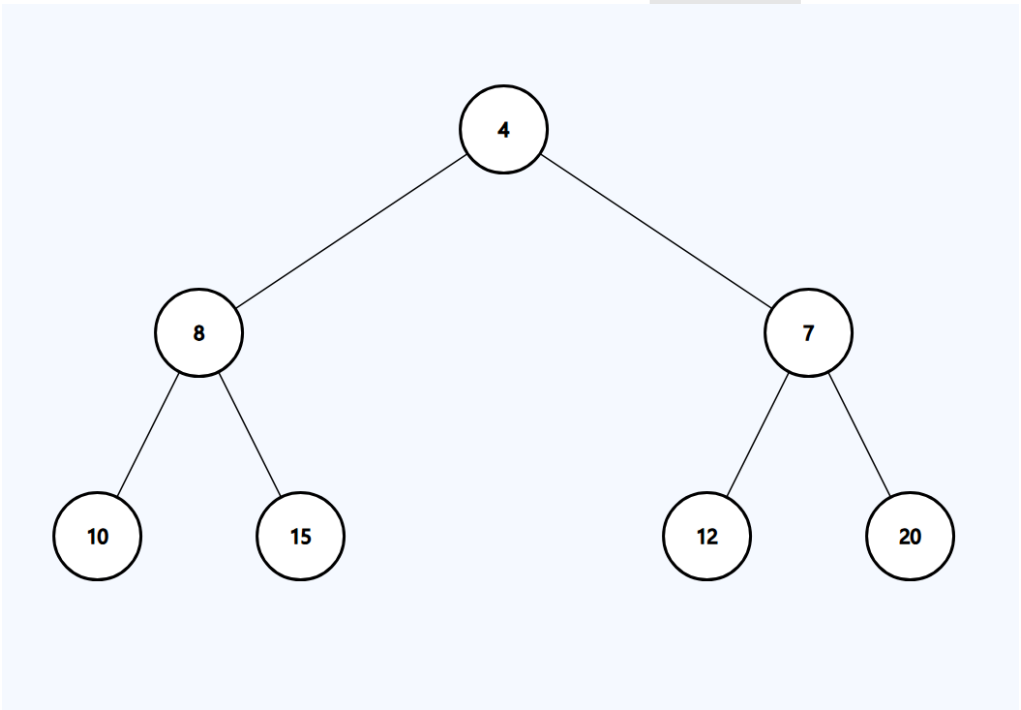


由節點[4]開始執行Heapify，最小堆積結果如下：



(4)最小堆積-刪除節點[3]

刪除樹根節點[3]後，依照堆積規則，以樹葉節點[10]取代樹根，並由樹根往下進行Heapify，時間複雜度為O(log$_2$n)：



三、正規化是為資料表的優化，而資料庫正規化有一些規則，每條規則都稱為「正規形式」（Normal Form），請說明各階段正規化的規則（包含第一正規化、第二正規化、第三正規化和BCNF等）。（25分）

試題評析	本題為幾乎每年必考的資料庫理論的試題，但題目並沒有實作或額外申論的部分，純粹為資料庫正規化的記憶型考題，對於考生來說算是非常容易作答，只需照定義復述即可。
考點命中	《高點・高上資料處理講義》第三回，JAMES編撰，頁34 ~40。

答：

1.第一正規化(1NF)：在資料表中的所有記錄之屬性內含值都是基元值Atomic(Atomic Value)，亦即無重覆項目群，不存在多值屬性(multi-valued attributes)、複合屬性(composite attributes)。
2.第二正規化(2NF)：關聯屬於2NF， 屬於1NF，且對於所有非鍵值屬性，必須「完全相依」於主鍵，即不可「部分功能相依」於主鍵。換言之，也就是不允許當某些欄位只與「主鍵中的部分欄位」有「相依性」，而與另一部分的欄位沒有相依性。
3.第三正規化(3NF)：關聯為3NF，若且唯若此關聯屬於2NF，且關聯中所有屬性皆非遞移相依(non-transitive dependent)於主鍵，不存在遞移相依於主鍵之屬性。
4.Boyce-Codd正規化(BCNF)：一關聯為BCNF，若且唯若此關聯中，所有non-trivial的功能相依之決定因素皆為此關聯之候選鍵(candidate key)，即「主鍵」中的各欄位不可以相依於其他非主鍵的欄位。
5.第四正規化(4NF)：關聯表R屬於第四正規化型式(4NF)，若且唯若它所有的多重值相依性都是功能相依性。
6.第五正規化(5NF)：關聯表R屬於第五正規化型式(5NF)，若且唯若R中所有的合併相依性(X,Y)中的X與Y兩關聯表所含有的共同屬性(換言之為作自然合併的屬性)是R的候選鍵，意即5NF中不可以存在合併相依性。

四、請依照下列程式碼，當執行函數呼叫Test(3)時，最後輸出結果為何？並請寫出詳細過程。（25分）

```
String Test(int n) {
    String s = n + Test(n - 1) + n + Test(n - 2);
    if (n <= 0) return "";
     return s;
}
```

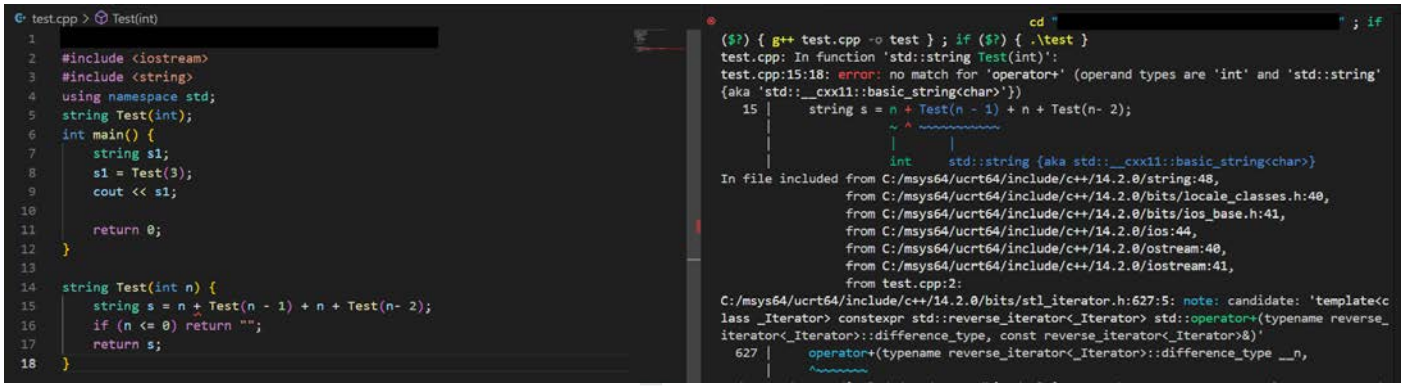
試題評析	本題為程式實作題，題目選擇為遞迴式的程式碼解讀，本題以考點角度而論式議題，為相當有趣的試題，難度適中且能驗證考生對於程式碼的追蹤能力，但可惜的是本題並未說明程式語言為何，且存在嚴重的type casting問題，實際執行結果應很有爭議，我認為出題者可能沒有仔細思考過型態轉換問題，此題需要使用to_string來確保執行結果一致。
考點命中	《高點・高上資料處理講義》第三回，JAMES編撰，頁34~40。

答：

理論上來說，題目是希望利用 {+} operator 來表達字串 Concatenate 的意思，依此題意則執行結果如下：

```
Test(3) = "3" + Test(2) + "3" + Test(1)
        = "3" + "2" + Test(1) + "2" + Test(0) + "3" + "11"
        = "3" + "2" + "1" + Test(0) + "1" + Test(-1) + "2" + Test(0) + "3"          + "11"
       = "3" + "2112" + "3" + "11"
       = "32112311"
```

然而本題存在 Type casting 問題，題目中並沒有聲明該語言種類為何，因此實際執行結果取決於該程式語言是否能在此情況進 implicit type casting，例如以 C++語言為例，題目中 int n 與 string {+}產生 type mismatch，此題即便更改 Test 引數為 string，又反而會造成遞迴式中 Test(n - 1)產生 type casting 問題，因此實際執行結果如下(Compiler Error)：



```
test.cpp > Test(int)
#include <iostream>
#include <string>
using namespace std;
string Test(int);
int main() {
    string s1;
    s1 = Test(3);
    cout << s1;

    return 0;
}

string Test(int n) {
    string s = n + Test(n - 1) + n + Test(n- 2);
    if (n <= 0) return "";
    return s;
}
```

```
cd "                                                    " ; if
($?) { g++ test.cpp -o test } ; if ($?) { .\test }
test.cpp: In function 'std::string Test(int)':
test.cpp:15:18: error: no match for 'operator+' (operand types are 'int' and 'std::string'
{aka 'std::__cxx11::basic_string<char>'})
   15 |     string s = n + Test(n - 1) + n + Test(n- 2);
      |                ~ ^ ~~~~~~~~~~~
      |                |   |
      |                int     std::string {aka std::__cxx11::basic_string<char>}
In file included from C:/msys64/ucrt64/include/c++/14.2.0/string:48,
                 from C:/msys64/ucrt64/include/c++/14.2.0/bits/locale_classes.h:40,
                 from C:/msys64/ucrt64/include/c++/14.2.0/bits/ios_base.h:41,
                 from C:/msys64/ucrt64/include/c++/14.2.0/ios:44,
                 from C:/msys64/ucrt64/include/c++/14.2.0/ostream:40,
                 from C:/msys64/ucrt64/include/c++/14.2.0/iostream:41,
                 from test.cpp:2:
C:/msys64/ucrt64/include/c++/14.2.0/bits/stl_iterator.h:627:5: note: candidate: 'template<c
lass _Iterator> constexpr std::reverse_iterator<_Iterator> std::operator+(typename reverse_
iterator<_Iterator>::difference_type, const reverse_iterator<_Iterator>&)'
  627 |     operator+(typename reverse_iterator<_Iterator>::difference_type __n,
      |     ^~~~~~~~
```

Test 應修改為：

```
string Test(int n) {
    if (n <= 0)
        return "";
    string s = to_string(n) + Test(n - 1) + to_string(n) + Test(n - 2);
    return s;
}
```

執行結果：

```
[redacted]\Desktop\TestCode> cd "[redacted]" ; if ($?) { g++ tes
t.cpp -o test } ; if ($?) { .\test }
32112311
```