1. AdjustHeap( StudentNumberNode \*array, int I, int size);

參數含義

StudentNumberNode \* ：元素為 StudentNumberNode 的數組首地址

Int i ：當前需要調整的節點序號

Int size ：當前Heap中節點數量，從 1 開始計數

Int max; 父節點，左右孩子節點三者中，所存儲的值最大的節點序號

該函數是建 MaximumHeap 最核心的函數。我的 MaximumHeap 是從 0 開始計數的，即 ：

樹根為序號 0，

左孩子序號 = 父節點序號 \* 2 + 1，

右孩子序號 = 父節點序號 \* 2 + 2，

父節點序號 = ( 孩子節點序號 – 1 ) / 2。

函數邏輯

1. 判斷當前需要調整的節點是否為非葉子節點，由於葉子節點沒有左右孩子，所以當前需要调整的節點是葉子節點的話，則不進行任何操作。函數直接完成
2. 判斷當前需要調整的節點是否有左孩子，即左孩子節點序號是否大於當前Heap 最後一個元素序號。同時判斷左孩子節點中存儲的值是否大於序號為 max 節點中存儲的值，若大於，則將左孩子節點的序號賦給 max，若不，則不進行操作。
3. 判斷當前需要調整的節點是否有右孩子，即右孩子節點序號是否大於當前Heap 最後一個元素序號。同時判斷右孩子節點中存儲的值是否大於序號為 max 節點中存儲的值，若大於，則將右孩子節點的序號賦給 max，若不，則不進行操作。
   1. 若父節點，左右孩子節點三者中，所存储的最大的節點序號 max 不等於父節點序號時，即左右孩子中至少有一個孩子節點中存储的值大於父節點中存儲的值，將左右孩子節點中最大的節點，與父節點進行交換。交換完成後，為了避免以新的孩子節點（之前為父節點）為父節點的子樹不滿足最大堆的需求，所以重複調用 AdjustHeap() 函數若父節點，使得以新的孩子節點（之前為父節點）為父節點的子樹同樣滿足最大堆的需求。
   2. 若以左右孩子節點三者中，所存储的最大的節點序號 max 等於父節點序號時，不進行任何操作。

1. BuildHeap(StudentNumberNode \*array, int size)

建堆的過程，從最後一個非葉子節點開始，直到根節點，迴圈調用 AdjustHeap( StudentNumberNode \*array, int I, int size) 調整堆，於是最大堆就可以建成。

1. InsertHeap(StudentNumberNode \*array, StudentNumberNode insertElement, int size)

向堆中插入新數據，按照堆的原則，新數據總是從最右下的空位置插入。數據插入後，從新數據所在子樹開始調用AdjustHeap（）函數進行調整。若新數據大於該子樹父節點中存放的值，則繼續以新數據所在的子樹調整，直到根節點為止。若新數據小於該子樹父節點中存放的值，則結束調整。

1. HeapSort(StudentNumberNode \*array, int size)

最大堆的樹根一定是堆中最大的數。

1. 將樹根與樹最後一個葉子節點交換，則此時堆中最大的數放在了堆中最後一個位置。此時忽略最後一個葉子節點，則少了最後一個葉子節點的子堆必然不滿足最大堆。對子堆調用BuildHeap(StudentNumberNode \*array, int size)函數建堆。
2. 重複步驟 1