佇列

佇列 ≡ 排隊

```
isEmpty(): boolean {query}
enqueue (in newItem : QueueItemType)
dequeue () throw QueueException
getfront (out queueFront : QueueItemType)
          {query} throw QueueException
dequeue (out queueFront : QueueItemType) throw QueueException
```

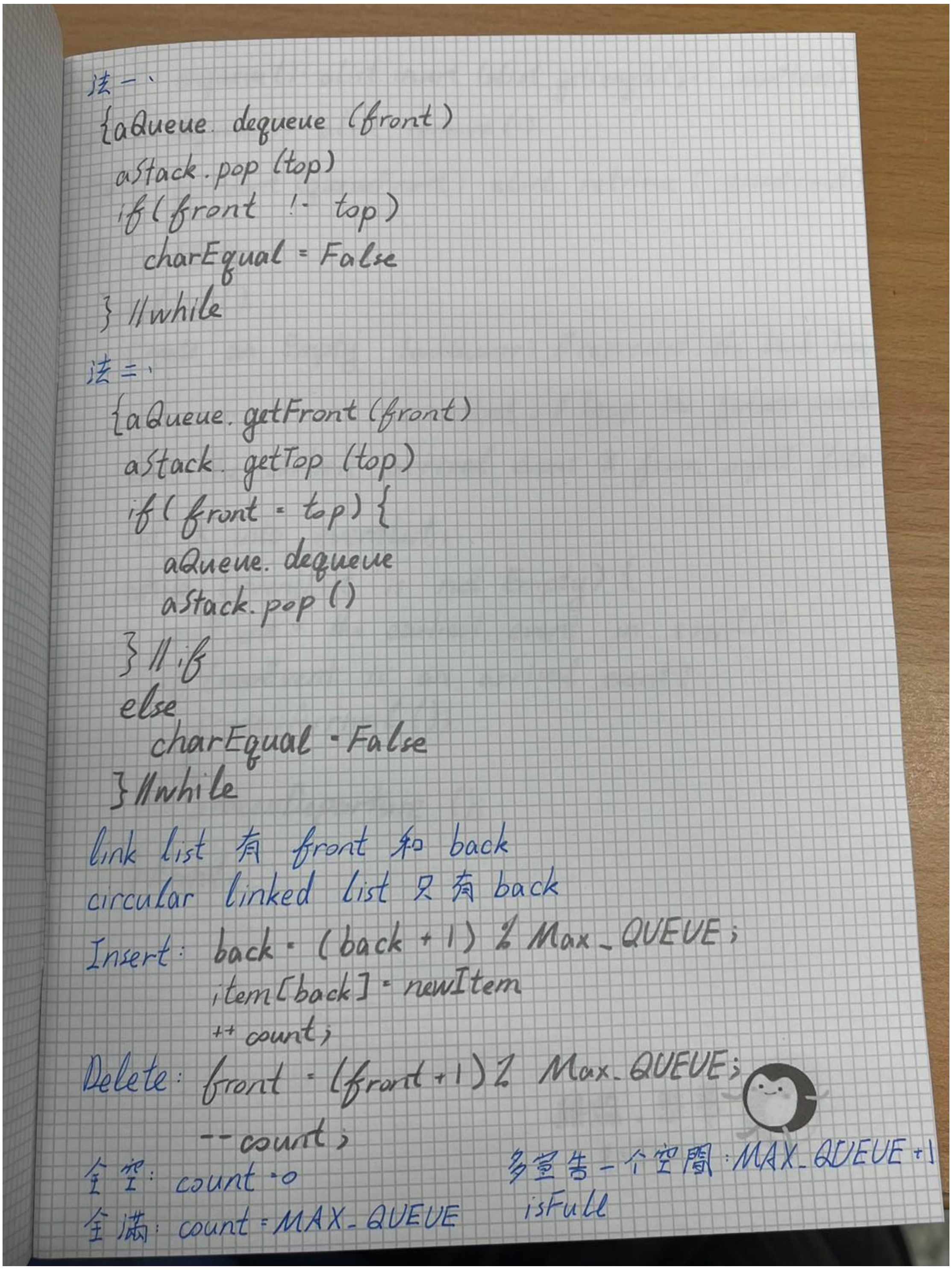
字串轉數字

```
do { aQueue.dequeue (ch)
  }// while (ch is blank)
n=0
done = False
while (!done and ch is digit) {
  n = nx 10 + integer represented by ch
  if (aQueue. isEmpty ())
    done = True;
  else
    aQueue.dequeue (ch)
}// while
```

有小數奌

```
if(!done and ch=','){
 aQueue.dequeue(ch)
 p=0
 while(!done and ch is digit){
   n=n×10+integer of ch
   p++
   if(aQueue.isEmpty())
     done=True;
   else
     aQueue.dequeue(ch)
 }//while
 $n=n \times (0.1)^p$
}//if

isPal(in str: string): boolean
 aQueue.createQueue()
 aStack.createStack()
 for(the next character ch in str){
   aQueue.enqueue(ch)
   aStack.push(ch)
 }//for
 charEqual = True
 while(!aQueue.isEmpty() && charEqual)
```



法一、

```
{aQueue.dequeue (front)
 aStack.pop (top)
 if(front != top)
   charEqual = False
}//while
```

法二、

```
{aQueue.getFront (front)
 aStack.getTop (top)
 if(front = top){
   aQueue.dequeue
   aStack.pop ()
 }//if
 else
   charEqual = False
}//while
```

link list 有 front 和 back

circular linked list 只有 back

```
Insert: back = (back + 1) % Max_QUEUE;
        item[back] = newItem
        ++count;
Delete: front = (front+1) % Max_QUEUE;
        --count;
```

全空: count = 0

全滿: count = MAX_QUEUE

多宣告一个空間: MAX_QUEUE + 1

isFull

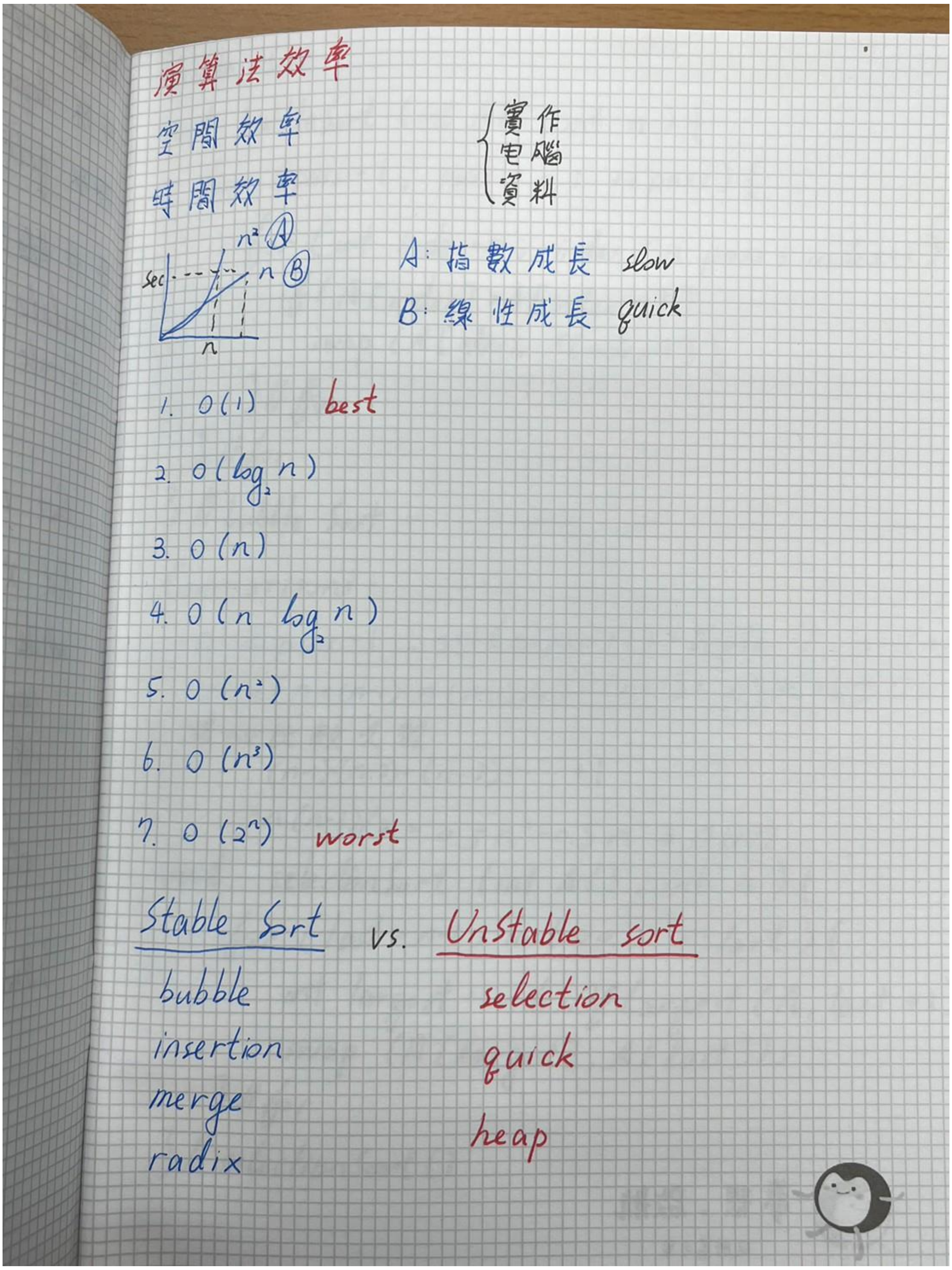
新增: enqueue(): aList.insert(aList.getLength()+1, newItem)

移除: dequeue(): aList.remove(1)

擷取: getFront(queueFront): aList.retrieve(1, queueFront)

事件驅動

```
Simulate()
  Create an Empty bankQueue; //represent the bank line
       "          eventList; //keep the future events
  Get the earliest arrival event X from input file
  Put X into eventList;
  while (eventList is not Empty) {
    newEvent = the earliest event in evenList
    if (newEvent is an arrival event)
      processArrival();
    else
      processDeparture();
  }//while
```



演算法效率

空間效率

時間效率

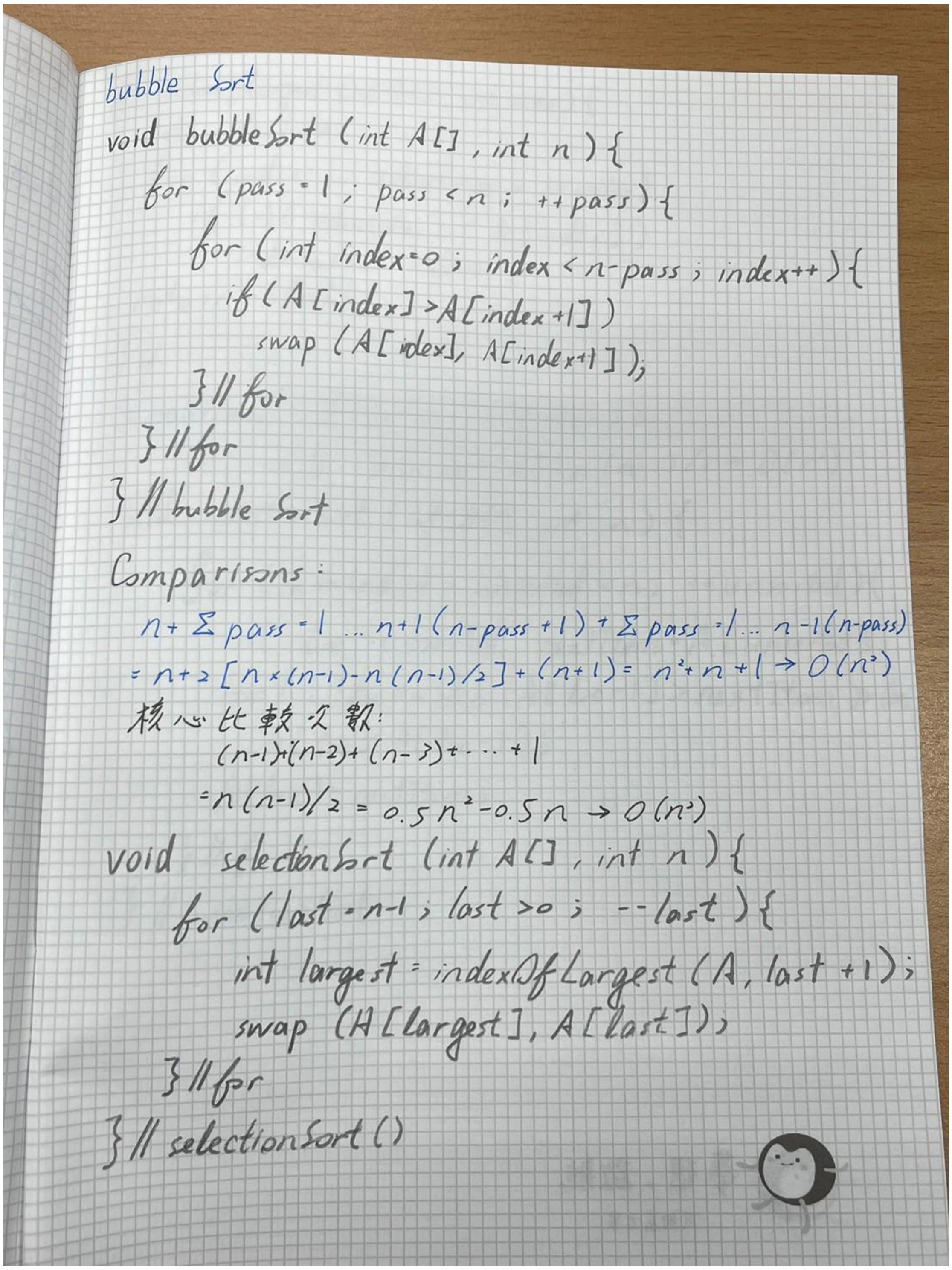
- 實作
- 电腦
- 資料

A: 指數成長 slow

B: 線性成長 quick

1. $O(1)$ best
2. $O(\log_2 n)$
3. $O(n)$
4. $O(n \log_2 n)$
5. $O(n^2)$
6. $O(n^3)$
7. $O(2^n)$ worst

Stable Sort	vs.	Unstable sort
bubble		selection
insertion		quick
merge		heap
radix		



bubble Sort

```
void bubbleSort (int A[], int n){
  for (pass = 1; pass < n; ++pass){
     for (int index=0; index < n-pass; index++){
        if (A[index] > A[index+1])
           swap (A[index], A[index+1]);
     }// for
  }// for
}// bubble Sort
```

Comparisons:

$n + \Sigma\, pass = 1 \ldots n+1 (n-pass+1) + \Sigma\, pass = 1 \ldots n-1 (n-pass)$

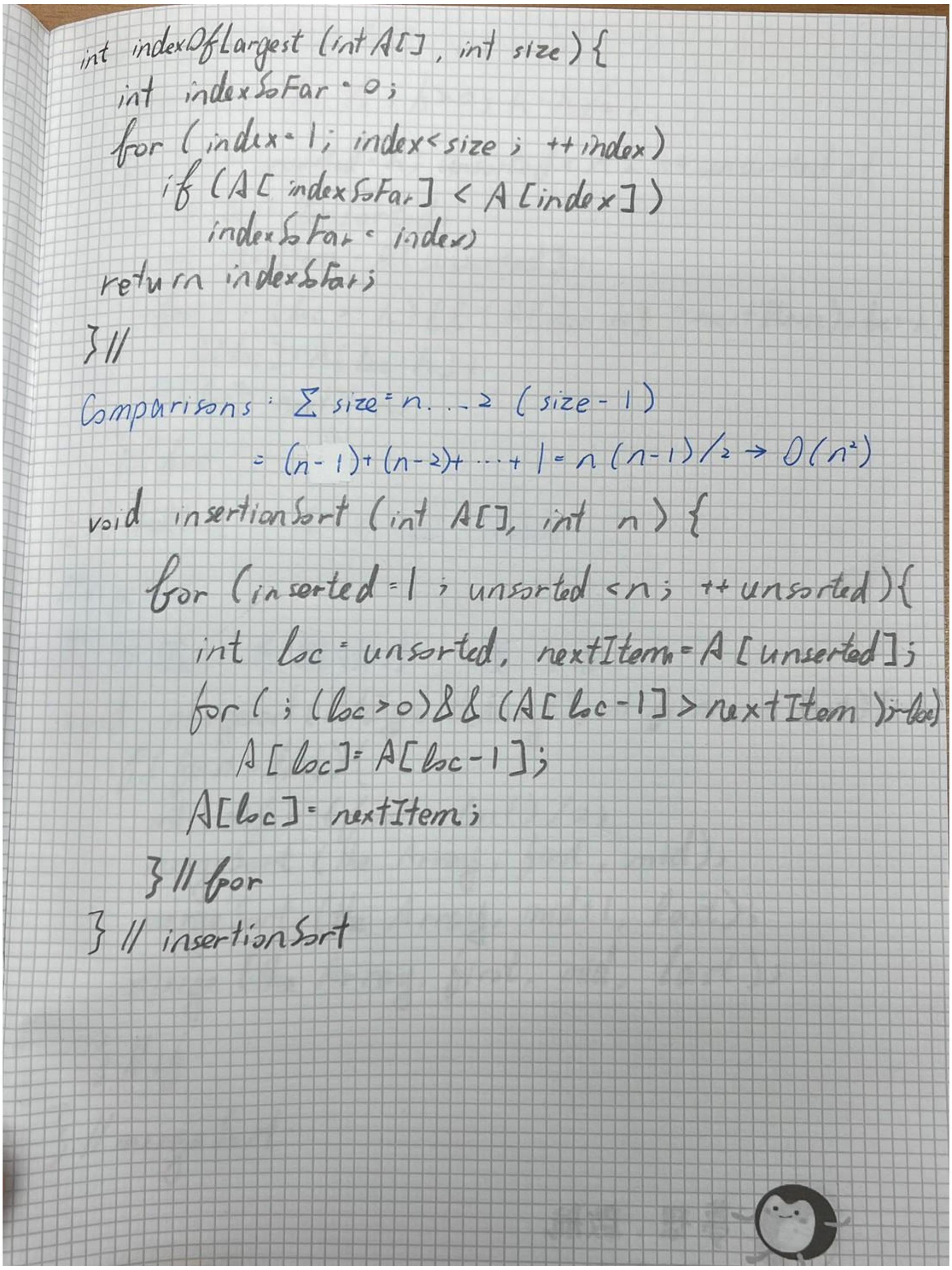
$= n + 2[n \times (n-1) - n(n-1)/2] + (n+1) = n^2 + n + 1 \rightarrow O(n^2)$

核心比較次數:

$(n-1)+(n-2)+(n-3)+\cdots+1$

$= n(n-1)/2 = 0.5n^2 - 0.5n \rightarrow O(n^2)$

```
void selectionSort (int A[], int n){
   for (last = n-1; last > 0; --last){
      int largest = indexOfLargest (A, last+1);
      swap (A[largest], A[last]);
   }// for
}// selectionSort ()
```



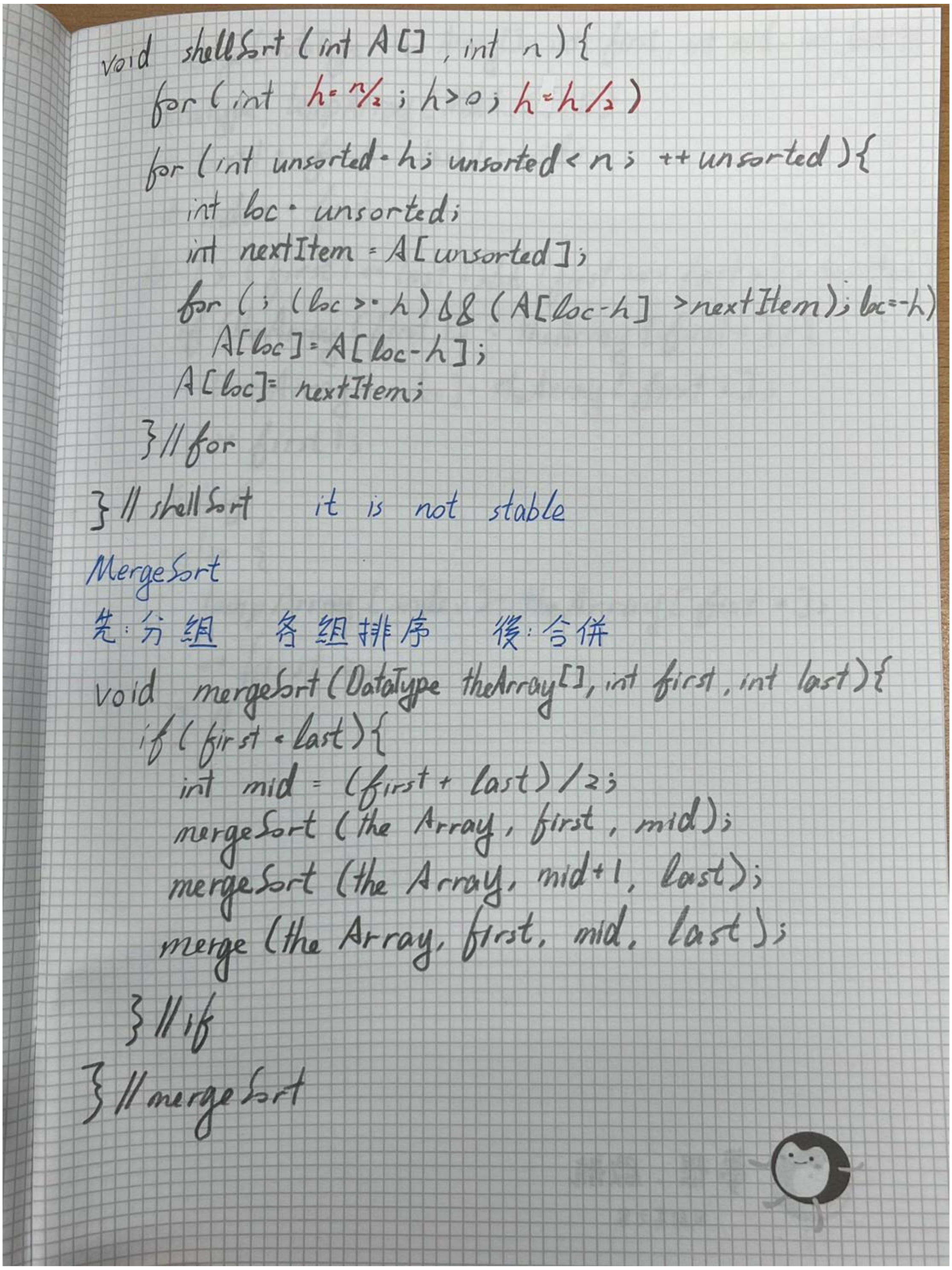
```
int indexOfLargest (int A[], int size){
  int indexSoFar = 0;
  for (index = 1; index < size ; ++index)
      if (A[ indexSoFar] < A[index])
         indexSoFar = index)
 return indexSoFar;

}//
```

Comparisons : $\sum_{size=n \ldots 2} (size-1)$

$= (n-1)+(n-2)+\cdots+1 = n(n-1)/2 \rightarrow O(n^2)$

```
void insertionSort (int A[], int n) {
    for (inserted = 1 ; unsorted < n; ++unsorted){
       int loc = unsorted, nextItem = A [unserted];
       for ( ; (loc > 0) && (A[ loc-1] > nextItem );--loc)
          A[loc]= A[loc-1];
       A[loc] = nextItem;
    }// for
}// insertionSort
```

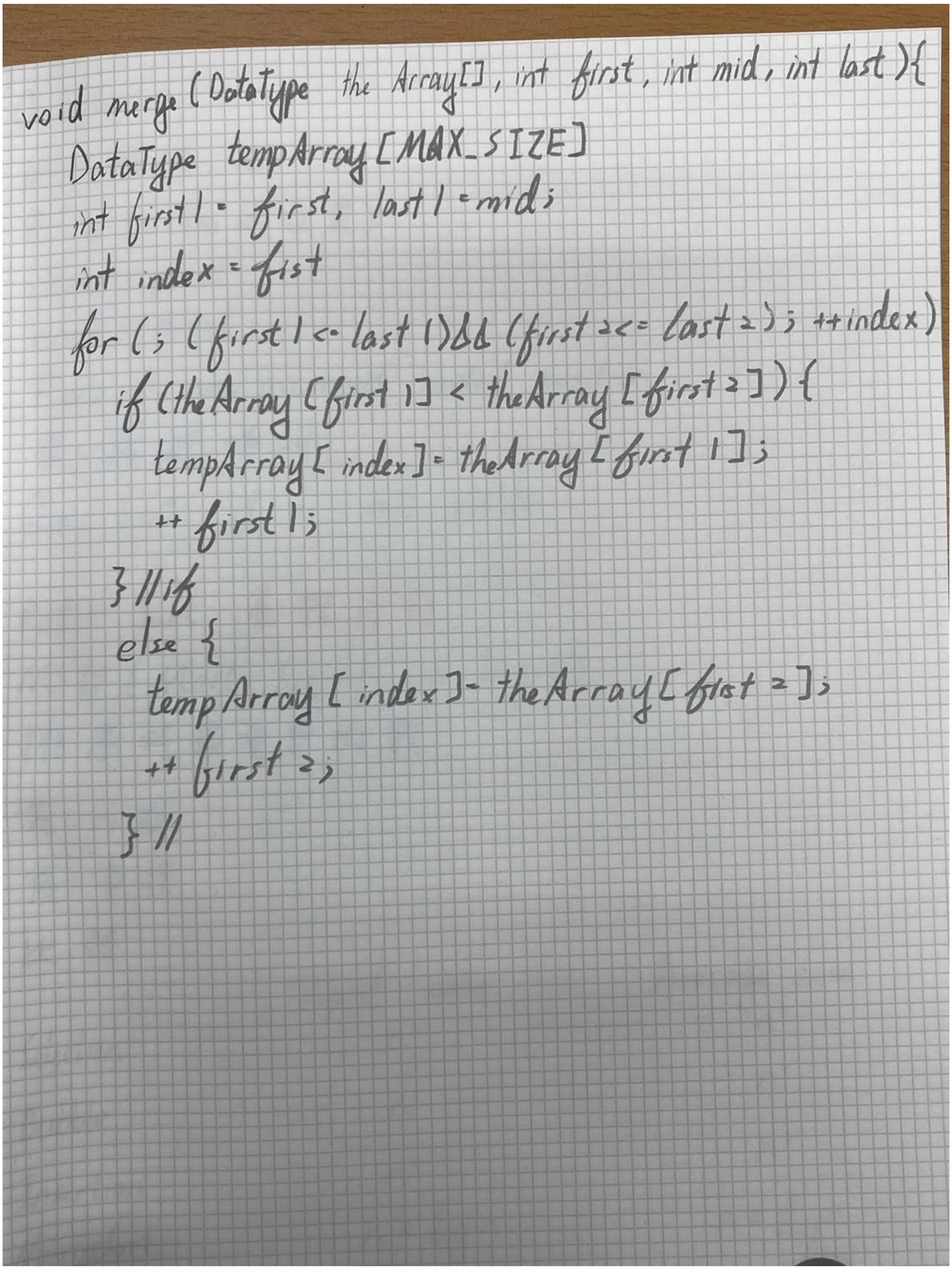


```
void shellSort (int A[] , int n){
  for (int h= n/2 ; h>0; h=h/2)
  for (int unsorted=h; unsorted<n; ++unsorted){
    int loc= unsorted;
    int nextItem = A[unsorted];
    for ( ; (loc >= h) && (A[loc-h] >nextItem); loc-=h)
      A[loc]=A[loc-h];
    A[loc]= nextItem;
  }//for
}//shellSort   it is not stable
```

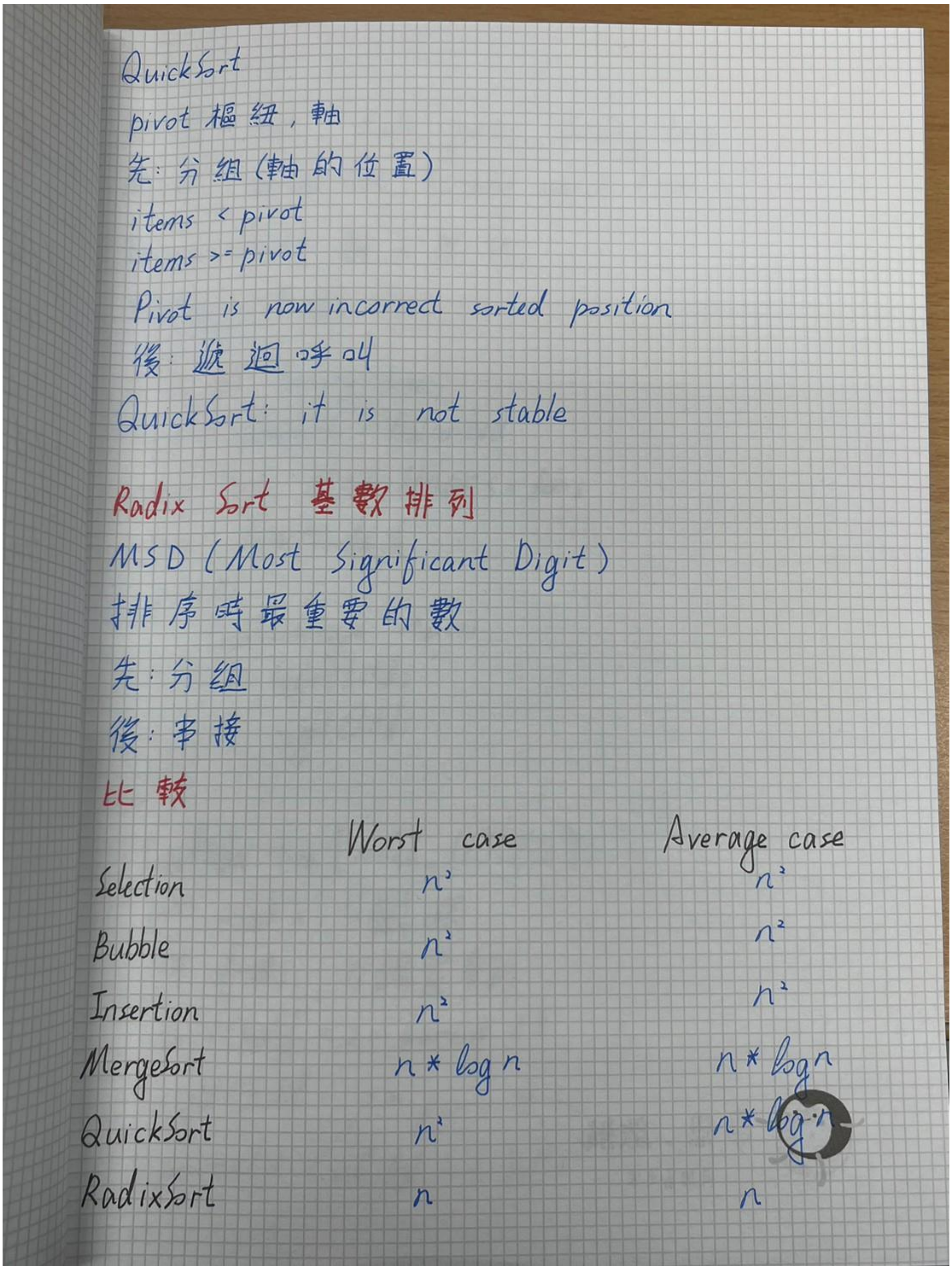
MergeSort

先:分组　各组排序　後:合併

```
void mergeSort(DataType theArray[], int first, int last){
  if (first < last){
    int mid = (first + last)/2;
    mergeSort (the Array, first, mid);
    mergeSort (the Array, mid+1, last);
    merge (the Array, first, mid, last);
  }//if
}//mergeSort
```



```
void merge (DataType the Array[], int first, int mid, int last ){
  DataType tempArray [MAX_SIZE]
  int first1 = first, last1 = mid;
  int index = fist
  for (; (first1 <= last1)&&(first2<= last2); ++index)
    if (theArray [first1] < theArray [first2]) {
      tempArray[ index]= theArray [first1];
      ++ first1;
    } //if
    else {
      tempArray [ index]= theArray[fist 2];
      ++ first 2;
    } //
```



QuickSort

pivot 樞紐，軸

先：分组（軸的位置）

items < pivot

items >= pivot

Pivot is now incorrect sorted position

後：遞迴呼叫

QuickSort: it is not stable

Radix Sort 基數排列

MSD (Most Significant Digit)

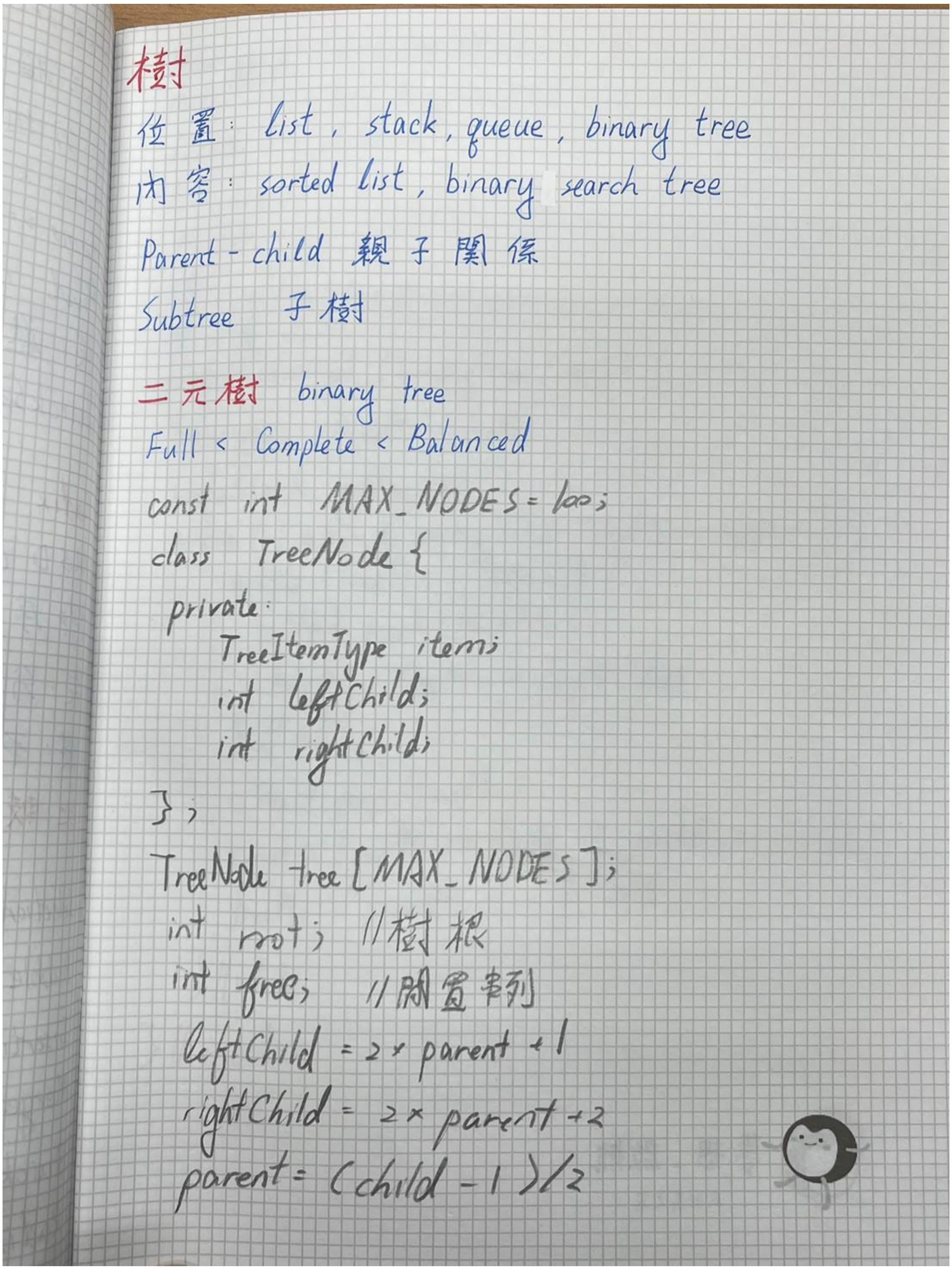
排序時最重要的數

先：分组

後：串接

比較

	Worst case	Average case
Selection	n^2	n^2
Bubble	n^2	n^2
Insertion	n^2	n^2
MergeSort	$n * \log n$	$n * \log n$
QuickSort	n^2	$n * \log n$
RadixSort	n	n



樹

位置: list, stack, queue, binary tree

內容: sorted list, binary search tree

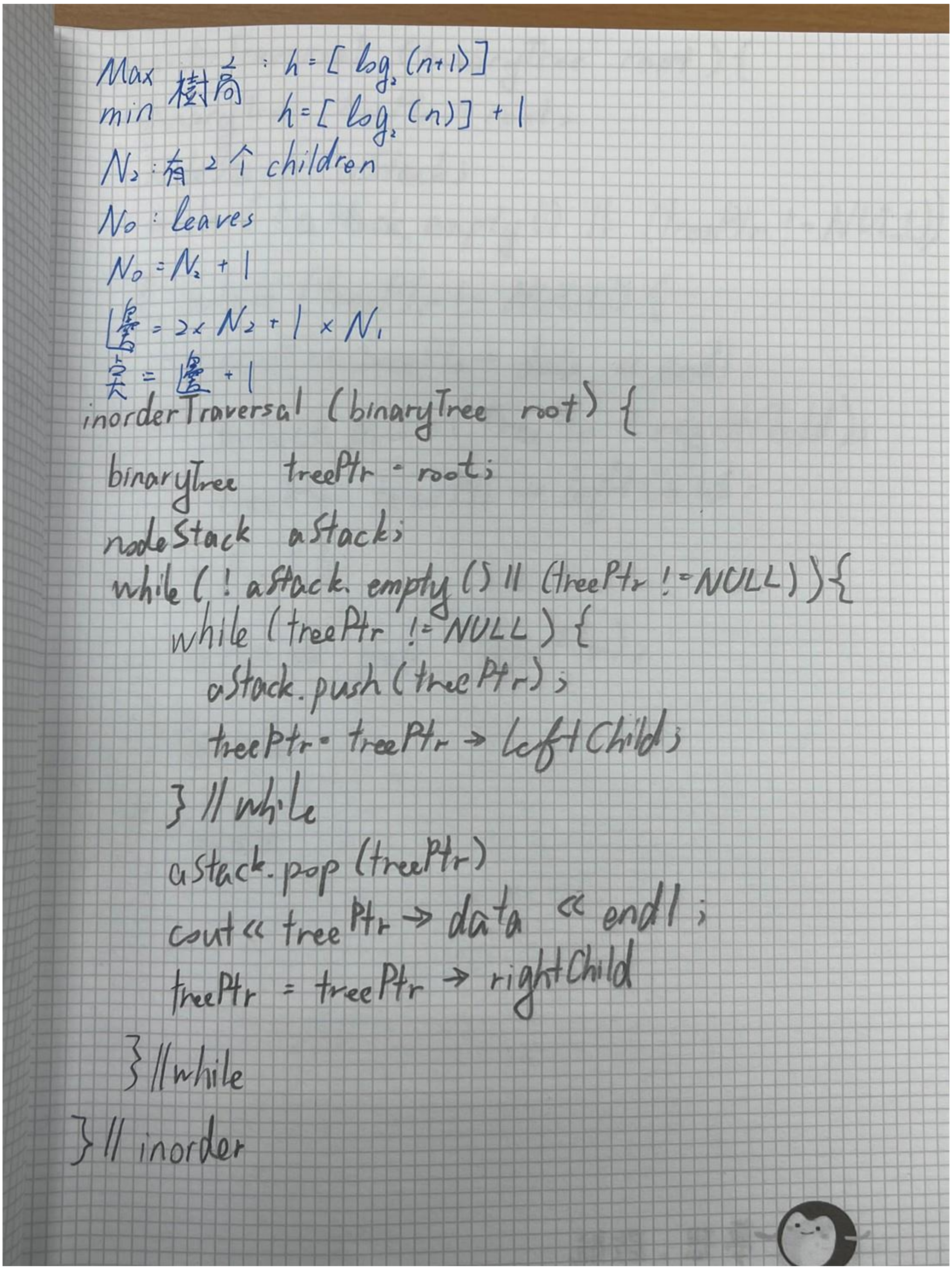
Parent-child 親子關係

Subtree 子樹

二元樹 binary tree

Full < Complete < Balanced

```
const int MAX_NODES = 100;
class TreeNode {
 private:
    TreeItemType item;
    int leftChild;
    int rightChild;
};
TreeNode tree[MAX_NODES];
 int root; //樹根
 int free; //閒置串列
  leftChild = 2 * parent + 1
  rightChild = 2 * parent + 2
  parent = (child - 1)/2
```



Max 樹高 : $h = [\log_2(n+1)]$

min $h = [\log_2(n)] + 1$

N_2 : 有 2 个 children

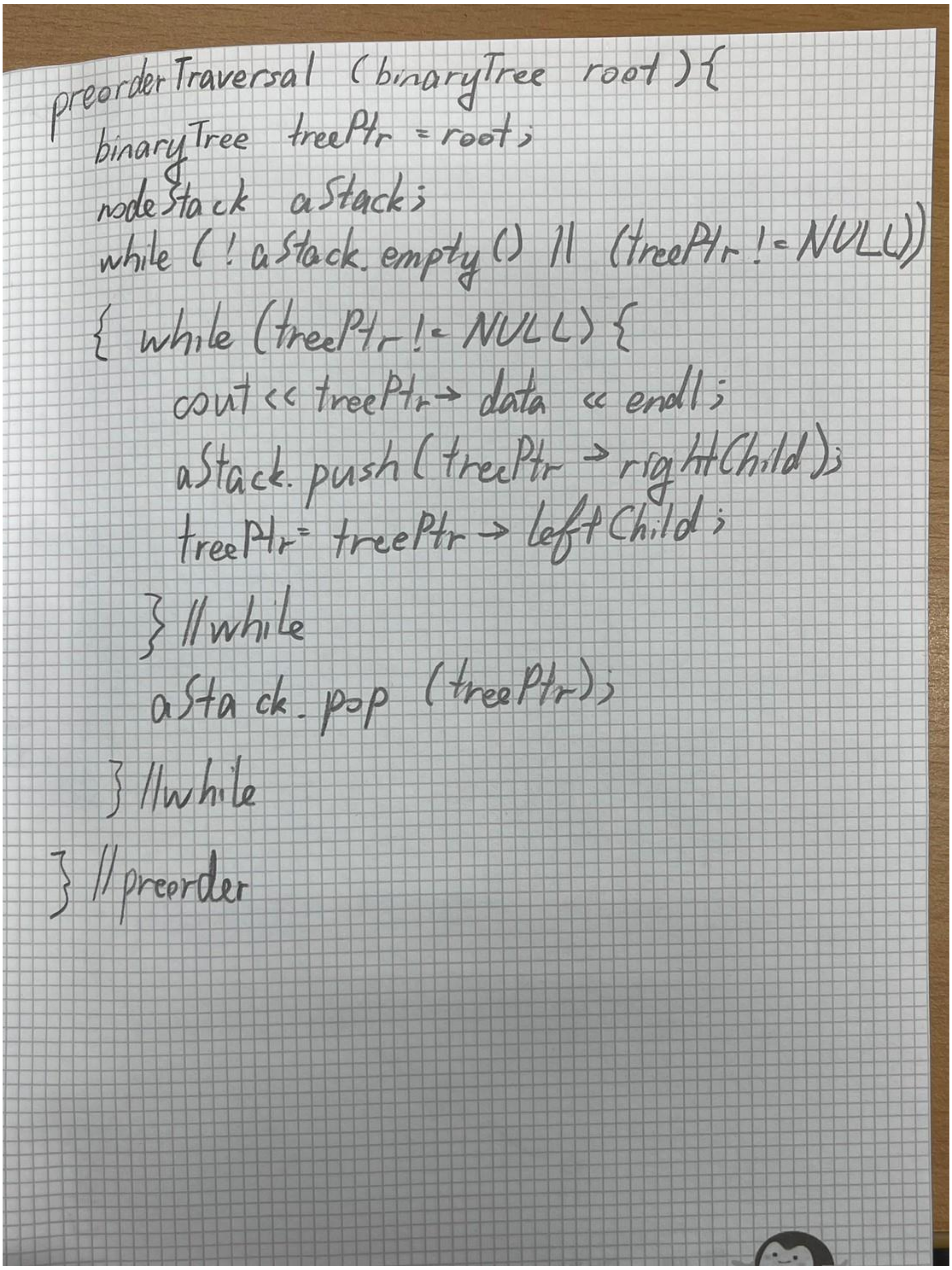
N_0 : leaves

$N_0 = N_2 + 1$

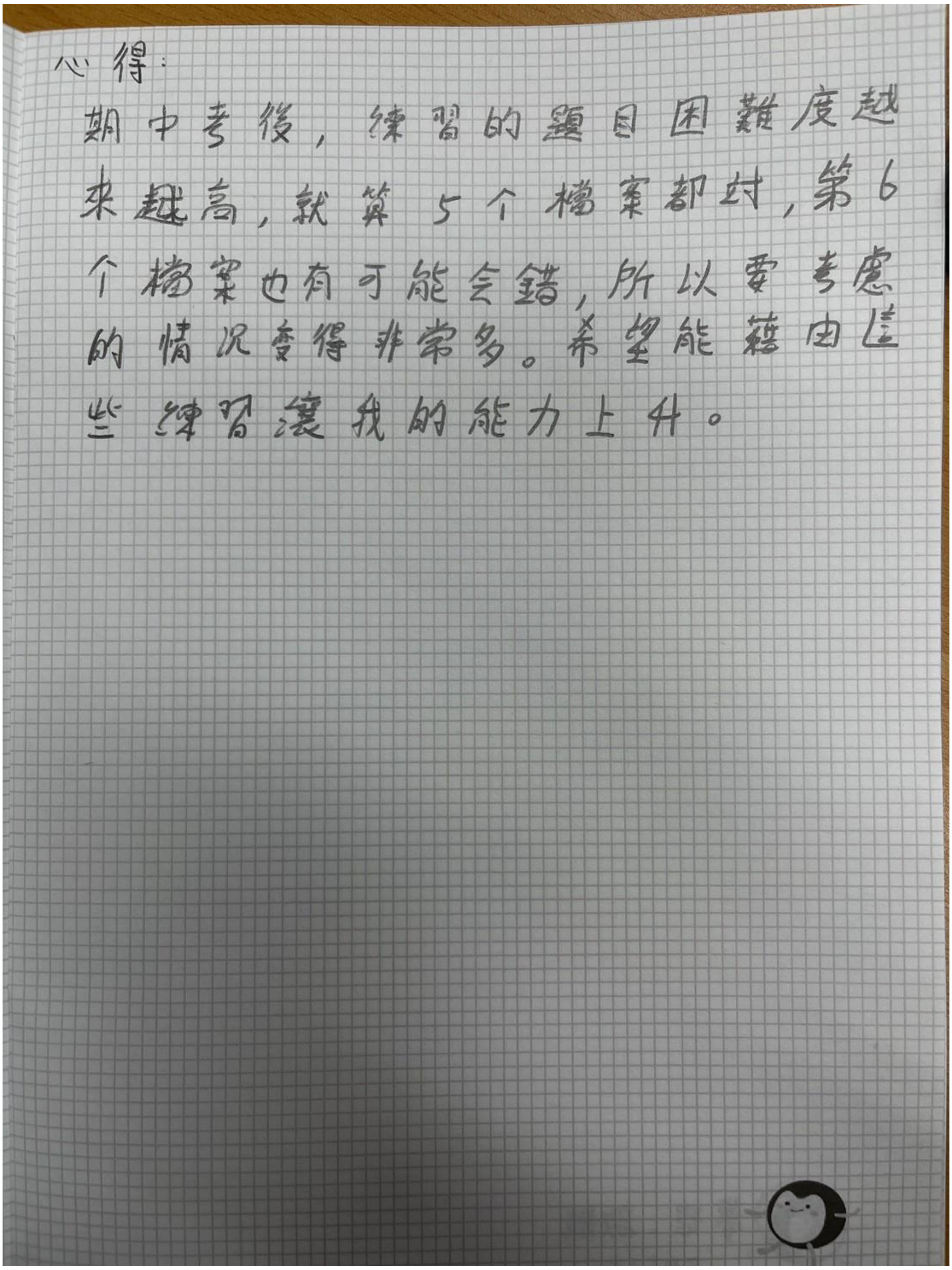
邊 $= 2 \times N_2 + 1 \times N_1$

點 = 邊 + 1

```
inorderTraversal (binaryTree root) {
 binaryTree  treePtr = root;
 nodeStack  aStack;
 while ( ! aStack. empty () || (treePtr != NULL) ) {
     while (treePtr != NULL) {
        aStack.push (treePtr);
        treePtr = treePtr -> leftChild;
     } // while
     aStack.pop (treePtr)
     cout << treePtr -> data << endl;
     treePtr = treePtr -> rightChild
   } //while
} // inorder
```



```
preorderTraversal (binaryTree root){
  binaryTree treePtr = root;
  nodeStack aStack;
  while (!aStack.empty() || (treePtr != NULL))
  { while (treePtr != NULL) {
        cout << treePtr → data << endl;
        aStack.push(treePtr → rightChild);
        treePtr = treePtr → leftChild;
     } //while
     aStack.pop (treePtr);
   } //while
} //preorder
```



心得：

期中考後，練習的題目困難度越来越高，就算5个檔案都对，第6个檔案也有可能会錯，所以要考慮的情況变得非常多。希望能藉由这些練習讓我的能力上升。