第八週作業補充說明

1. 前言

在今天的助教課裡，我們學到了怎麼樣排序，為了讓各位為同學更好地掌握排序演算法，助教打上這份說明文件給各位同學。

2. 作業規格變更

首先，今天的作業總共有五題：

- 第一題：找出不成對的數字
- 第二題：quicksort
- 第三題：mergesort
- 第四題：inplace mergesort
- 第五題：eight queens

其中，第二題、第三題、第四題是排序問題，也是這份補充文件主要會講解的部分。

3. 排序

3.1. 什麼是排序？

給定一串數字，將其由小到大排好，就是排序。

3.2. 如何將一串數字由小到大排好？

如果我們只能透過兩兩比較來進行排序，這樣的排序方法稱為comparison-based的方法，例如：11月3日星期二上課所教的選擇排序法(selection sort)就是comparison-based的排序方法。在這第八次作業，我們要實作兩個著名的比較排序演算法：快速排序法(quicksort)和合併排序法(mergesort)。這兩個方法解決問題的思路，都是divide-and-conquer。什麼是divide-and-conquer呢？請參閱下一小節。

3.3. Divide-and-conquer

Divide-and-conquer解決問題的方法是先把大問題劃分成許多的小問題，並分別擊破小問題。分成三個步驟：

- 第一步：Divide：也就是把大問題拆分成小問題。
- 第二步：Conquer：當問題被化簡得夠小之後，就可以直接解決了。
- 第三步：Merge：將解決小問題後得到的局部解，合併成大問題的全局解。

在快速排序法和合併排序法中我們都可以清楚看到這三個步驟。

4. 前置作業

在開始撰寫排序演算法之前，我們先寫好四個基本函式，分別是：

1. `int* createArray(int* A, int size)`
2. `void swap(int* A, int i, int j)`
3. `void shuffle(int* A, int size)`
4. `void printArray(int* A, int size)`

關於這四個函式的定義請務必詳細參閱主要作業文檔，若有任何疑問請寄信詢問助教，或於助教們的office hour(星期三晚上7-9和星期五晚上7-9)來詢問，以下就如何使用做示範。

4.1. 函式範例

4.1.1. createArray

4.1.1.1. 代碼範例

範例代碼片段：

```
int main(){
    int size = 5;
    int* A = new int[size];
    A = createArray(A, size);
    printArray(A, 5);
    delete [] A;
    return 0;
}
```

輸出：

```
0 1 2 3 4
```

說明：

1. `int size = 5;`：size是我們要宣告陣列的大小。
2. `int* A = new int[size];`：和記憶體預借一個大小為`size`的整數空間，並且用指標變數`A`指著，每一格抽屜大小都是一個整數的大小，也就是4B。關於指標同學們將在期中考後學習到，在這裡完全不用擔心，這不是這次作業的重點！這裡同學們只需要實作`createArray`的內容即可。
3. `A = createArray(A, size);`：創建一個大小為5的陣列，叫作A。
4. `printArray(A, 5);`：把陣列A印出來，詳見4.4。
5. `delete [] A;`：釋放記憶體空間。

4.1.1.2. 代碼格式

```
int* createArray(int* A, int size){
    /*
    your code here
    */
    return A;
}
```

注意，這裡宣告函式用到的`int*`代表回傳的是一個指標(記憶體位址)。

4.1.2. swap

4.1.2.1. 代碼範例

範例代碼片段：

```
int main(){
    int size = 10;
    int* A = new int[size];
    A = createArray(A, size);
    printArray(A, 10);
    cout << endl;
    swap(A[3], A[7]);
    printArray(A, 10);
    cout << endl;
    delete [] A;
    return 0;
}
```

輸出：

```
0 1 2 3 4 5 6 7 8 9
0 1 2 7 4 5 6 3 8 9
```

說明：

將`A[3]`和`A[7]`的值交換！

4.1.2.2. 代碼格式

```
void swap(int* A, int i, int j){
    /*
    your code here
    */
}
```

4.1.3. shuffle

4.1.3.1. 代碼範例

範例代碼片段：

```
int main(){
    int size = 10;
    int* A = new int[size];
    A = createArray(A, size);
    printArray(A, 10);
    cout << endl;
    shuffle(A, 10);
    printArray(A, 10);
    cout << endl;
    delete [] A;
    return 0;
}
```

輸出：

```
0 1 2 3 4 5 6 7 8 9
5 8 7 4 9 1 2 3 6 0
```

說明：

將創建的陣列亂序後輸出。在上述代碼範例中，我們並沒有設定random seed，同學們在自行測試時可以自己使用random seed。但是！請同學上繳作業後，不要把`srand()`寫在函數裡面，助教們會有自己的`srand()`，會測試同學們寫出來的亂序和助教們的結果是否相同，這個亂序函式的實作，請務必按照作業主文檔給的演算法實作。

4.1.3.2. 代碼格式

```
void shuffle(int* A, int size){
    /*
    your code here
    */
}
```

4.1.4. printArray

4.1.4.1. 代碼範例

範例代碼片段：

```
int main(){
    int size = 10;
    int* A = new int[size];
    A = createArray(A, size);
    printArray(A, 10);
    delete [] A;
    return 0;
}
```

輸出：

```
0 1 2 3 4 5 6 7 8 9
```

說明：

印出陣列A。透過這個函式，可以將陣列的狀態完整印出。

4.1.4.2. 代碼格式

```
void printArray(int* A, int size){
    /*
    your code here
    */
}
```

4.2. 作業繳交規定

請同學將撰寫好的四個函式，放入`basic.cpp`內，並且建立對應個`basic.h`。提醒同學，`basic.cpp`裡面放的是函式的實作細節(implementation)，而`basic.h`裡面放的是函式的宣告(declaration)。格式如下：

4.2.1. basic.cpp

```
// include things you need

int* createArray(int* A, int size){
    /*
    your code here
    */
}

void swap(int* A, int i, int j){
    /*
    your code here
    */
}

void shuffle(int* A, int size){
    /*
    your code here
    */
}

void printArray(int* A, int size){
    /*
    your code here
    */
```

```
25 }
```

4.2.2. basic.h

```
1 // declaration of the four functions
```

5. 快速排序法 (quicksort)

5.1. Pivot

快速排序法的關鍵在於pivot的選擇，為了簡單起見，我們選擇陣列中第0個元素當作我們的pivot，例如給定如下數列：

```
1 6 9 7 5 8 4 3 0 1 2
```

我們的pivot `A[0]`，也就是 `6`。

5.2. Partition

我們第一步就是要將這個 `6` 移動的正確的位置，而且在 `6` 左邊的數字全部都小於 `6`，在 `6` 的右邊的數字全部都大於 `6`，如下：

```
1 {numbers smaller than 6} 6 {numbers greater than 6}
```

這個步驟叫做「partition」，是快速排序法的關鍵步驟。助教們給定代碼樣板如下：

```
1 int partition(int* A, int l, int r){
2     /*
3     your code here
4     */
5 }
```

這個 `partition` 函數有三個輸入：

- `int *A`：輸入的陣列
- `int l`：左足標
- `int r`：右足標

這個函數會將 `A[l]` 到 `A[r]` 形成的子陣列進行 partition，並返回pivot最終的足標。例如給定下列程式片段：

```
int main(){
    int A[10] = {6, 9, 7, 5, 8, 4, 3, 0, 1, 2};
    cout << partition(A, 0, 9) << endl;
    printArray(A, 10);
    return 0;
}
```

因為我們選擇了陣列第0個元素作為pivot，所以陣列的輸出為：

```
6
3 2 1 5 0 4 6 8 7 9
```

這裡的`6`是指pivot的正確足標，注意在`6`的左邊數字`3 2 1 5 0 4`都小於`6`，在`6`的右邊數字`8 7 9`都大於`6`，這樣就完成了partition。

5.3. 快速排序法演算原理

完成partition後，接下來就是遞迴地排序「`3 2 1 5 0 4`」和「`8 7 9`」。`3 2 1 5 0 4`和`8 7 9`排序完畢之時，整個陣列也排序完畢。再給另一個例子：

```
int main(){
    int A[10] = {6, 9, 7, 5, 8, 4, 3, 0, 1, 2};
    cout << partition(A, 4, 7);
    printArray(A, 10);
    return 0;
}
```

此時輸出為：

```
7
6 9 7 5 0 4 3 8 1 2
```

這是因為我們這是partiton的對象是`8 4 3 0`，而這個子序列經partition後為`0 4 3 8`，所以返回pivot position為`7`。

有了partition函數後，quicksort就幾乎完成了！一但我們partition後，只需要遞迴地對pivot左半和右半進行快速排序即可。代碼規範如下：

```
void quickSort(int*A, int l, int r){
    /*
    your code here
    */
}
```

寫得好的話，甚至在四行代碼就可以完成了呢！我們一開始說過，快速排序法是個divide-and-conquer排序法，這是因為：

- Divide：就是partition
- Conquer：遞迴地對pivot左半和右半進行快速排序
- Merge：不需要

5.4. 快速排序法的時間複雜度

快速排序法的平均時間複雜度為 $O(n\log(n))$，但是最壞的情況會達 $O(n^2)$，其中 n 是要排序的資料數量。

5.5. 作業繳交規格

請將：

1. `partition`函式
2. `quickSort`函式

封裝成：

1. `quickSort.cpp`
2. `quickSort.h`

6. 合併排序法 (mergesort)

6.1. 一般合併排序法 (mergesort)

6.1.1. 合併排序法的演算原理

合併排序法一樣是採divide-and-conquer的策略，但是和快速排序法不同的是，合併排序法的divide非常容易，反倒是merge困難一些。

6.1.1.1. Divide

將一個陣列分成兩個子序列，例如：

```
{6, 9, 7, 5, 8, 4, 3, 0, 1, 2}
```

可以分成：

```
{6, 9, 7, 5}
```

```
{8, 4, 3, 0, 1, 2}
```

兩個子序列。

6.1.1.2. Conquer

一但分成兩個子序列後，遞迴地對兩個子序列進行排序。

6.1.1.3. Merge

這個函式會將兩個已經排序好的陣列，合併成一個更大排序好的陣列，例如將`1 3 5 7`和`2 4 6 8`合併成`1 2 3 4 5 6 7 8`。你會需要一個輔助陣列來完成合併的動作。同學們可以自行設計`merge`函式：

```
{?} merge(?){
    /*
    your code here
    */
}
```

助教對這個函式不做特別的規定！

6.1.2. 合併排序法的實作

一但實作好`merge`函式，合併排序法就幾乎完成了！請按照如下規格完成排序演算法：

```
void mergeSort(int* A, int size){
    /*
    your code here
    */
}
```

注意，`mergeSort`函式請務必遵循規格，並且使用已經寫好的`merge`函式。

6.1.3. 作業繳交規格

請將：

1. `merge`函式
2. `mergeSort`函式

封裝成：

1. `mergeSort.cpp`
2. `mergeSort.h`

6.2. Inplace合併排序法(inplace mergesort)

6.2.1. 合併排序法的優點和缺點

6.1的合併排序法的優點就是快！即使在最糟糕的情況，排序時間仍然可以維持在 $O(n\log(n))$，但是他有個缺點，就是十分浪費記憶體，一個長度為 n 的序列，會需要 $O(n)$ 的輔助記憶體空間來完成排序。

6.2.2. 如何更高效地使用記憶體空間

透過`swap`函式，我們可以將排序演算法變成inplace的排序。所謂inplace演算法就是使用輔助空間為$O(1)$的演算法。

6.2.3. Inplace merge

請將6.1.1.3的`merge`函式改為inplace，並重新命名為`mergeInplace`函式，同樣地助教並不對此函式做特別規格限制。另外，重新寫一個`mergeSortInplace`，並且使用到`mergeInplace`，規格如下：

```
void mergeSortInplace(int* A, int size){
    /*
    your code here
    */
}
```

6.2.4. 作業繳交規格

請將：

1. `mergeInplace`函式
2. `mergeSortInplace`函式

封裝成：

1. `mergeSortInplace.cpp`
2. `mergeSortInplace.h`

7. 作業繳交規格總結

針對問題二到問題四，同學一共要繳交8個檔案，分別是：

1. `basic.cpp`
2. `basic.h`
3. `quickSort.cpp`
4. `quickSort.h`
5. `mergeSort.cpp`
6. `mergeSort.h`
7. `mergeSortInplace.cpp`
8. `mergeSortInplace.h`

總共4個`.cpp`檔，和其對應的4個`.h`檔。助教會用下面三個檔案依序測試同學們的第二題到第四題。

7.1. 第二題(quickSort)

```
#include <iostream>
#include <ctime>
#include "basic.h"
```

```cpp
#include "quickSort.h"

using namespace std;

int main(int argc, char  ** argv){
    int size;
    if (argc == 3){
        srand(atoi(argv[1]));
        size = atoi(argv[2]);
    }
    else{
        srand(time(NULL));
        cin >> size;
    }

    int* A = new int[size];
    A = createArray(A, size);
    shuffle(A, size);
    printArray(A, size);
    cout << endl;

    quickSort(A, 0, size-1);
    printArray(A, size);
    cout << endl;
    delete [] A;
    return EXIT_SUCCESS;
}

#include "basic.cpp"
#include "quickSort.cpp"
```

以下將會使用到同學們寫的函式的行數標示出來：

- Line 20: `A = createArray(A, size);`
- Line 21: `shuffle(A, size);`（而此函式又會用到`swap`函式）
- Line 22: `printArray(A, size);`
- Line 23: `quickSort(A, 0, size-1);`（而此函式又會用到`partition`函式）
- Line 26: `printArray(A, size);`

7.2. 第三題(mergeSort)

```cpp
#include <iostream>
#include <ctime>
#include "basic.h"
#include "mergeSort.h"

using namespace std;

```

```cpp
int main(int argc, char  ** argv){
    int size;
    if (argc == 3){
        srand(atoi(argv[1]));
        size = atoi(argv[2]);
    }
    else{
        srand(time(NULL));
        cin >> size;
    }

    int* A = new int[size];
    A = createArray(A, size);
    shuffle(A, size);
    printArray(A, size);
    cout << endl;

    mergeSort(A, size);
    printArray(A, size);
    cout << endl;
    delete[] A;
    return EXIT_SUCCESS;
}

#include "basic.cpp"
#include "mergeSort.cpp"v
```

7.3. 第四題(mergeSortInplace)

```cpp
#include <iostream>
#include <ctime>
#include "basic.h"
#include "mergeSortInplace.h"

using namespace std;

int main(int argc, char  ** argv){
    int size;
    if (argc == 3){
        srand(atoi(argv[1]));
        size = atoi(argv[2]);
    }
    else{
        srand(time(NULL));
        cin >> size;
    }

    int* A = new int[size];
```

```
    A = createArray(A, size);
    shuffle(A, size);
    printArray(A, size);
    cout << endl;

    mergeSortInplace(A, size);
    printArray(A, size);
    cout << endl;
    delete[] A;
    return EXIT_SUCCESS;
}

#include "basic.cpp"
#include "mergeSortInplace.cpp"
```

8. 拆檔

8.1. 什麼是拆檔

將函式寫在不同的檔案，透過`include`的方式在不同的檔案間使用，謂之拆檔。

8.2. 為什麼要拆檔

為了重複利用代碼，我們將重複代碼包裝成函式；為了重複利用函式，我們將函式包裝稱檔案。

8.3. 怎麼拆檔

假設我們今天寫了一個函式：

```
void sayHello(int times){
    for (int i=0; i<times; i++){
        cout << "Hello!" << endl;
    }
}
```

我們要將其封裝成另外一個檔案`sayHello.cpp`，以供主程式`main.cpp`使用，首先我們開啟一個`.cpp`檔並命名為`sayHello.cpp`，並在裡面打上：

```
#ifndef _SAY_HELLO_
#define _SAY_HELLO_

#include <iostream>
using namespace std;

void sayHello(int times){
    for (int i=0; i<times; i++){
        cout << "Hello!" << endl;
    }
}

#endif
```

注意這裡必須要：

```
#include <iostream>
using namespace std;
```

因爲我們用到了`cout`函式。另外前置處理敘述：

```
#ifndef _SAY_HELLO_
#define _SAY_HELLO_

// define your function here

#endif
```

可以避免重複宣告的問題。接著，我們要撰寫標頭檔，開啟一個檔案命名為`sayHello.h`，並寫上函式`sayHello()`的宣告：

```
void sayHello(int times);
```

當然，你也可以寫成：

```
void sayHello(int);
```

這並不會影響函式的宣告。最後我們要在主函式使用剛剛定義好的函式，我們必須：

```cpp
#include <iostream>
#include "sayHello.cpp"
using namespace std;

int main(void)
{
    sayHello(5);
    return 0;
}
```

這樣就可以使用`sayhello()`函式了！上述代碼的輸出為：

```
Hello!
Hello!
Hello!
Hello!
Hello!
```