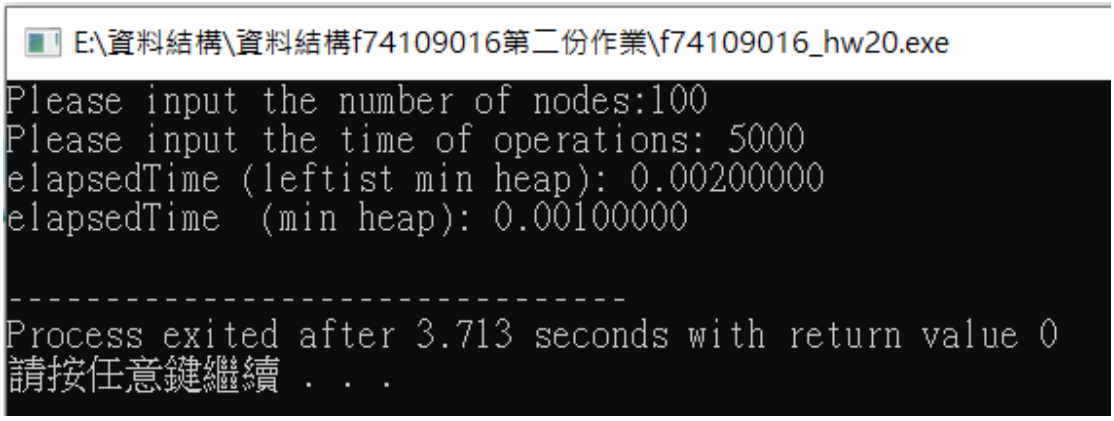
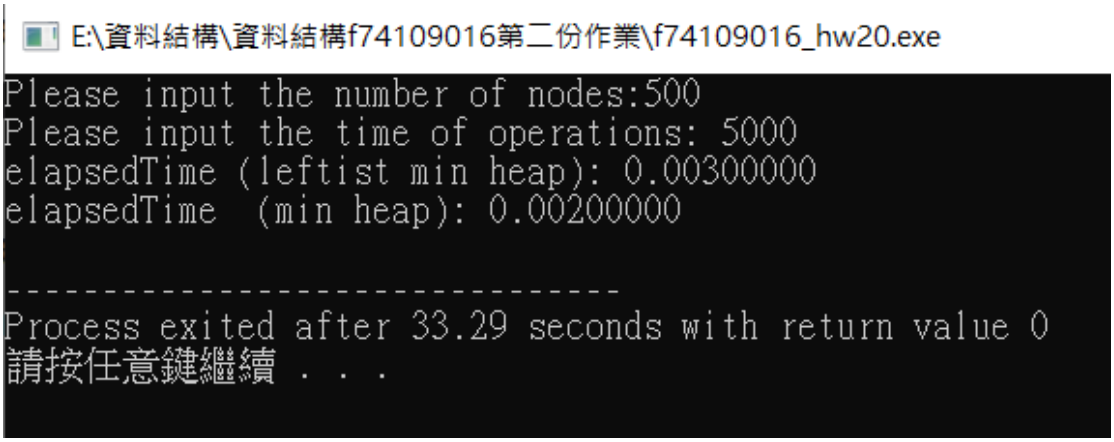
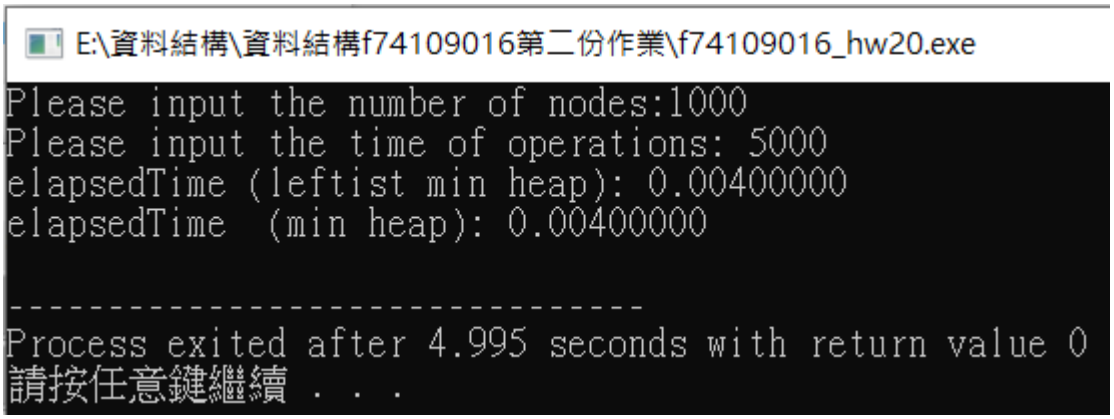
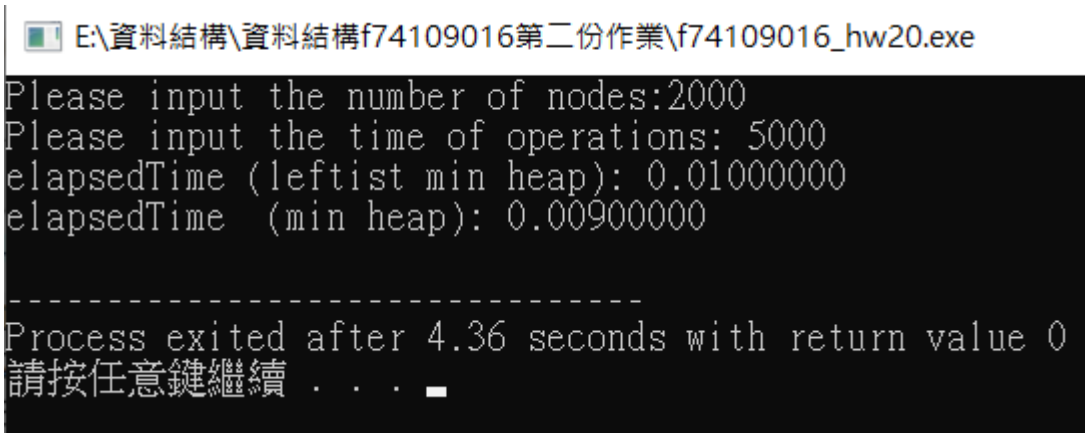
F74109016_hw20 葉惟欣

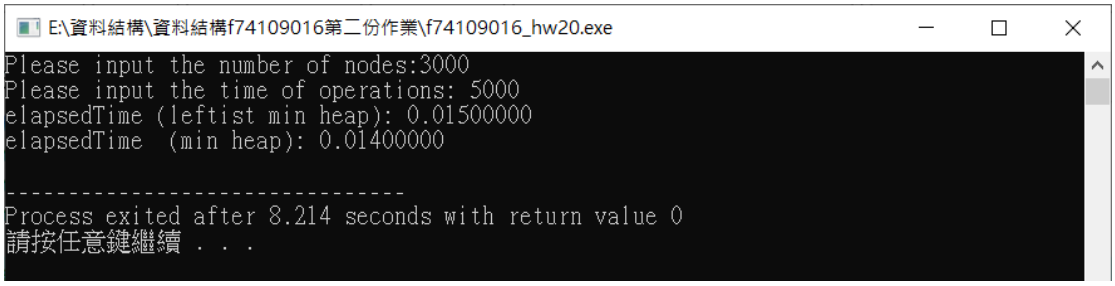
實驗:

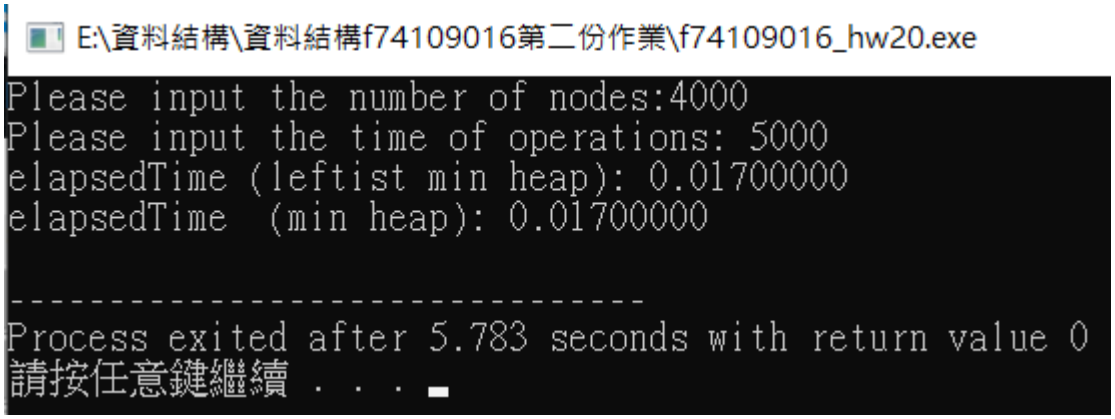


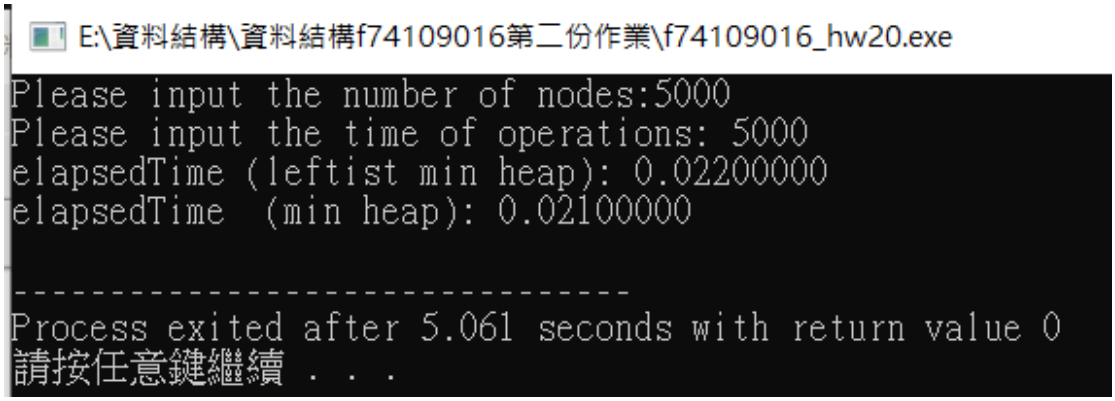












同樣的操作下，很多情況都是 leftist min heap 的時間比 min heap 長。

```
/********** leftist************/
typedef struct{
    int key;
}element;
typedef struct leftist Tree;

struct leftist{
    Tree * leftChild;
    element data;
    Tree * rightChild;
    int shortest;
    int id;
};
typedef struct node NODE;

typedef int bool;
enum { false, true };
struct node{
    int id;
    int data;
    NODE * next;
};
void SWAP(Tree *a,Tree *b,Tree* temp){
    *temp = *a;
    *a = *b;
    *b = *temp;
}
```

```
Tree* minUnion(Tree *a,Tree *b){
    /*recursivelu combine two nonempty min leftist trees */
    Tree* temp =(Tree *) malloc(sizeof(Tree));
    /* set a to be the tree with smaller root*/

    if(a->data.key > b->data.key){
        SWAP(a,b,temp); //右子樹的數自比較多，先調整
    }
    if(a->rightChild==NULL){            //如果沒有又子樹的話，就將b直接當作a的左子樹
        a->rightChild = b;
    }
    else{

        minUnion(a -> rightChild,b); //遞迴呼叫
    }
    //滿足leftchild的特性
    if(a->leftChild==NULL){          //沒有左小孩 ，將右小孩作為左小孩。
        a->leftChild = a->rightChild;
        a->rightChild = NULL;
    }
    else if(a->leftChild->shortest < a->rightChild->shortest){ //左小孩的數量比右小孩小
        SWAP(a->leftChild,a->rightChild,temp);
    }
    a->shortest = (!a->rightChild) ? 1: a->rightChild->shortest +1;
    return a;
}
```

```
65 void minMeld(Tree *a,Tree *b){
66     if(a==NULL) *a = *b;
67     else if(b!=NULL) minUnion(a,b);
68     b = NULL;
69 }
70 int delete_min(Tree *a){
71     Tree * b  = (Tree *) malloc(sizeof(Tree));
72     int _id = a->id;
73     if(a->rightChild!=NULL){
74         *b = *(a->rightChild);
75         *a = *(a->leftChild);
76         minMeld(a,b);
77     }
78     else{
79     *a = *(a->leftChild);
80     }
81     return _id;
82 }
```

做刪除動作

statement	frequency	Total steps
Tree * b = (Tree *) malloc(sizeof(Tree));		
int _id = a->id;	1(因為有 assign)	1
if(a->rightChild!=NULL){	1(要刪除的有兩個小孩)因為是左傾，所以如果有右小孩一定有左小孩	1
*b = *(a->rightChild);	1	1
*a = *(a->leftChild);	1	1
minMeld(a,b);	1	@@
}		
else{	1(要刪除的節點有可能只有一個小孩或沒有小孩)	1 這個 else 都是 O(1)所以影響不大
*a = *(a->leftChild);	1	1
}		
return _id;		

插入的話 就是先做合併動作

做合併動作 minMeld

statement	frquency	Total steps
if(a==NULL)	1	1
*a = *b;	1 如果只有一個節點直接給 assign 給 a	1
else if(b!=NULL)	1 如果 a 跟 b 都非空	1
minUnion(a,b);	1 就呼叫另一個函數	@@
b = NULL;	1	1

minUnion

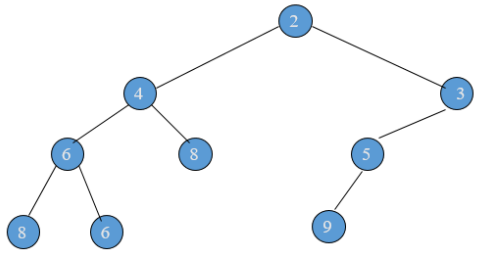
statement	frequency	totalsteps
Tree* temp =(Tree *) malloc(sizeof(Tree));	1	1
if(a->data.key > b->data.key){	1 //右子樹的的大小比較大，先調整	1
SWAP(a,b,temp);	1	3
}		
if(a->rightChild==NULL){	1 //如果沒有右子樹的話，就將 b 直接當作 a 的左子樹	1
a->rightChild = b;	1	1
}		
else{	有右子樹會做 O(logn)為 rightmost path 的長度	
minUnion(a -> rightChild,b);	遞迴呼叫	
}		
if(a->leftChild==NULL){	1 如果沒有左小孩，則左右小孩交換	1
a->leftChild = a->rightChild;	1	1
a->rightChild = NULL;	1	1
}		
else if(a->leftChild->shortest < a->rightChild->shortest){	1	1
SWAP(a->leftChild,a->rightChild,temp);	1	3
}		
a->shortest = (!a->rightChild) ? 1: a->rightChild->shortest +1;	1	1
return a;		

灰色的 statement 是看狀況才會執行，但除了遞迴呼叫外，其餘的時間複雜度都為 O(1)，倘落有進到遞迴呼叫，會進行 O(log n) 次 ，n 為 number of nodes in the a leftist tree.

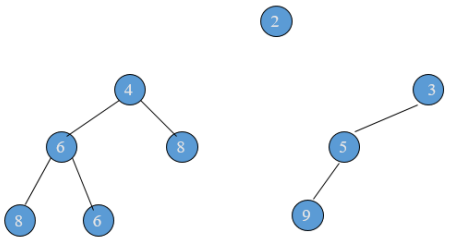
每次 minUnion 的時間複雜度為因為其他都是 O(1)，所以可以說 Union 複雜度是 O(log n)。

因此刪除動作的複雜度 O (log n)。(圖解)

刪除最小的節點



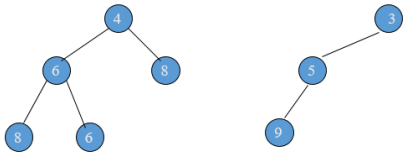
刪除最小的節點



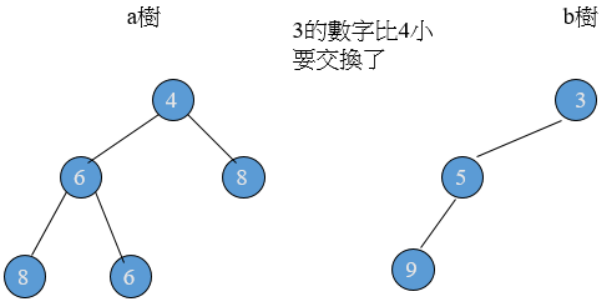
相當於把左子樹與右各自分開。形成兩個leftist tree

最右節點到 root 的長度為 log(n)。

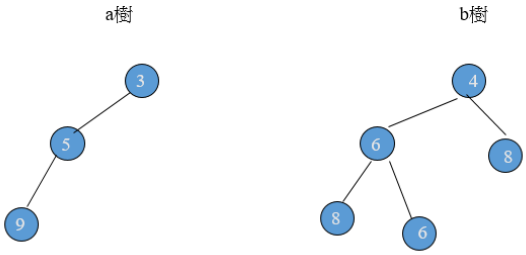
再將兩個子樹合併。

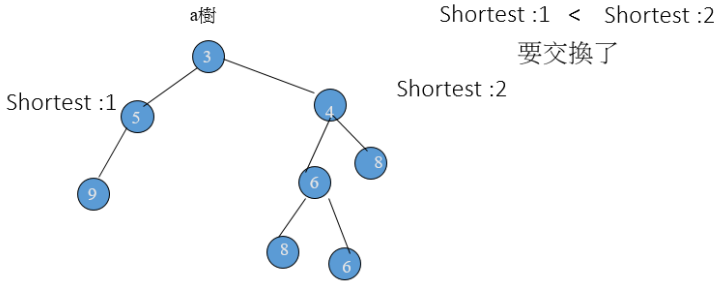


而合併相當於採訪最右邊的節點到root的長度。

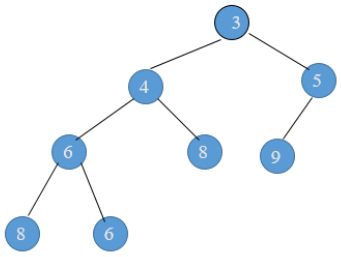


Swap(a,b,temp)

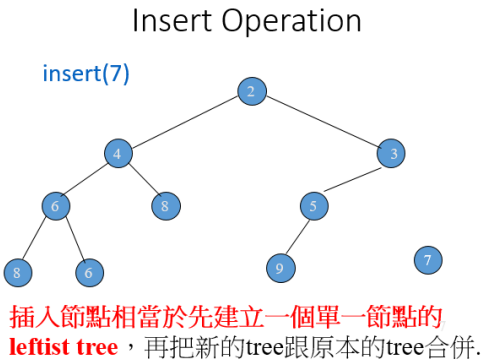




要算左右小孩的 shortest 所以需要 l log(n) ➔ 樹的高度的時間。



而插入動作其實就是合併動作 O(log n)。但差別差在要先建立一個 node。所以時間複雜度為 O(1)



插入節點相當於先建立一個單一節點的 **leftist tree**，再把新的tree跟原本的tree合併.

插入的頻率與刪除的頻率各一半，O(n/2)*O(log n) + O(log n)
插入時包含建立一個 min leftist tree

而 heap 的複雜度:
插入到最後一個節點再往上浮(如果比 parent 小的話)。浮的高度為 log (n) 此為樹的高度。 n 為樹的節點數量。
刪除 root 節點，並把最後一個節點補上去，補上去後開始往下沉。沉的高度為 log(n) 此為樹的高度。 n 為樹的節點數量。
所以可以說 heap 的複雜度為 O(log n)+ O(log n)。

因為 heap 是 complete binary tree 所以用陣列。因為 min heap 不需要建立新的節點。而 min leftist tree 不是 binary tree，所以每次插入都要建立節點所以時間複雜度還需加上創建節點的時間複雜度。

也因此我們從起初的實驗都可以看到 min heap 的用時都較 min leftist tree 少。