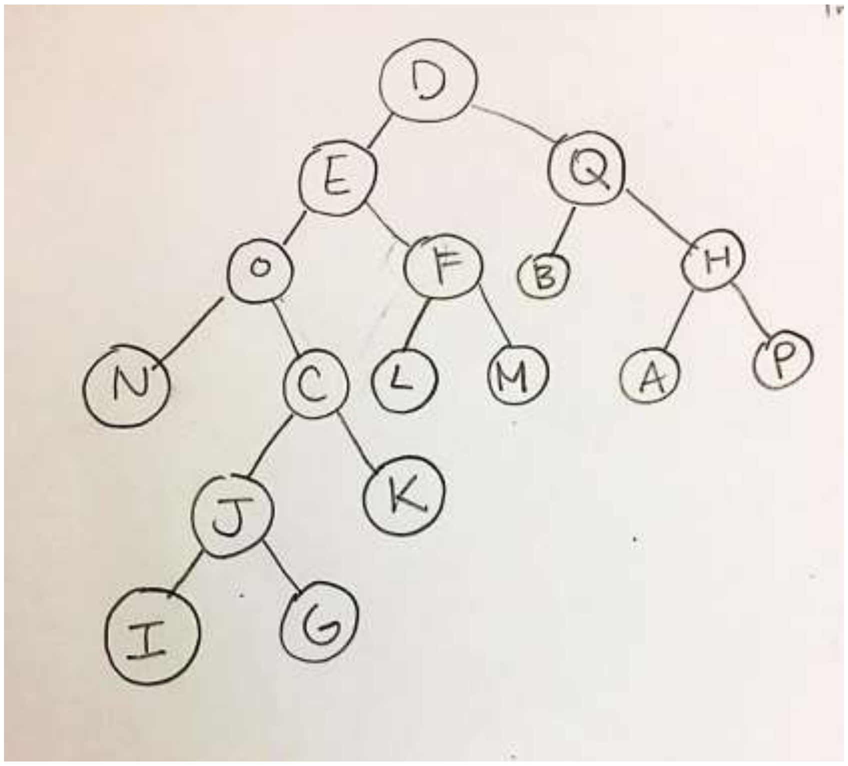
羅費南 – B06902102

Homework 3

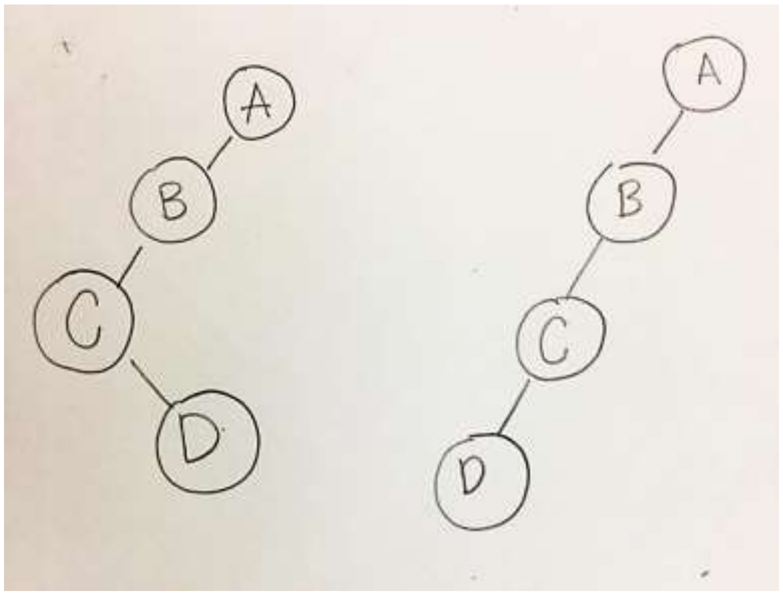
Problem 1:

1. Postorder: N I G J K C O L M F E B A P Q H D



2. Preorder: A B C D

   Postorder: D C B A



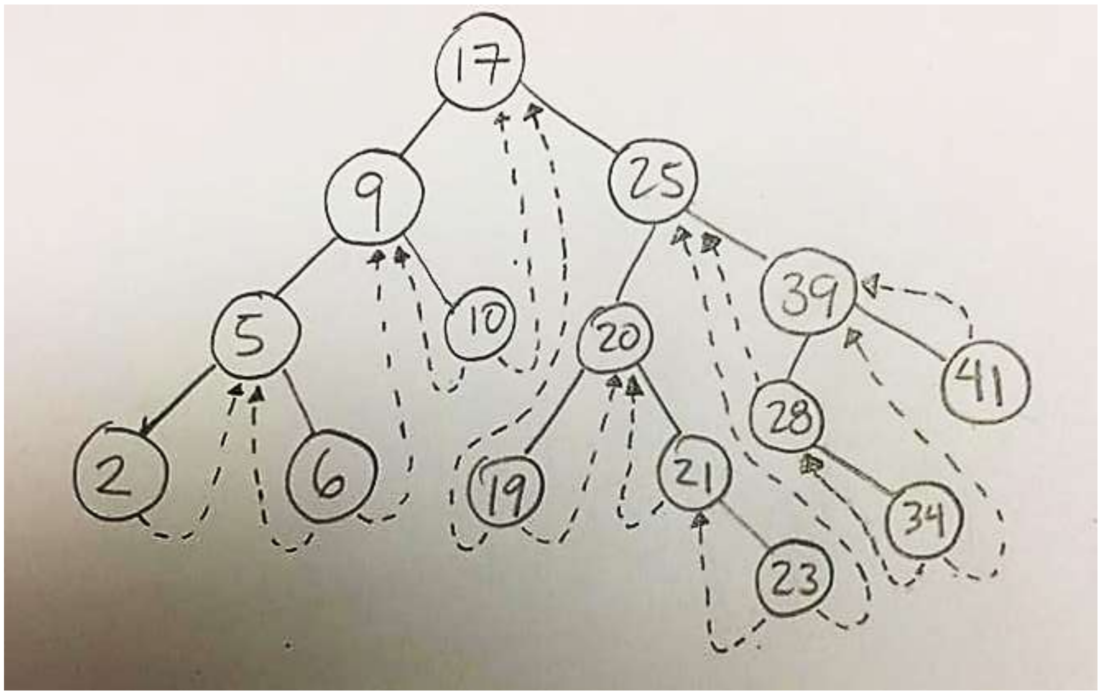
3. https://algorithms.tutorialhorizon.com/inorder-predecessor-and-successor-in-binary-search-tree/

   https://www.geeksforgeeks.org/threaded-binary-tree/

The successor of a node is another node in the right sub-tree that is at the most left place. The predecessor is the opposite; is the node at the left sub-tree which is at the most right place. Both are the elements next to the given node in inorder notation.

Threaded binary trees are trees that have the right nodes which point to NULL point to the inorder successor instead (if single threaded) and also have the left ones point to the predecessor (if double threaded).

4.



Problem 2:

1.

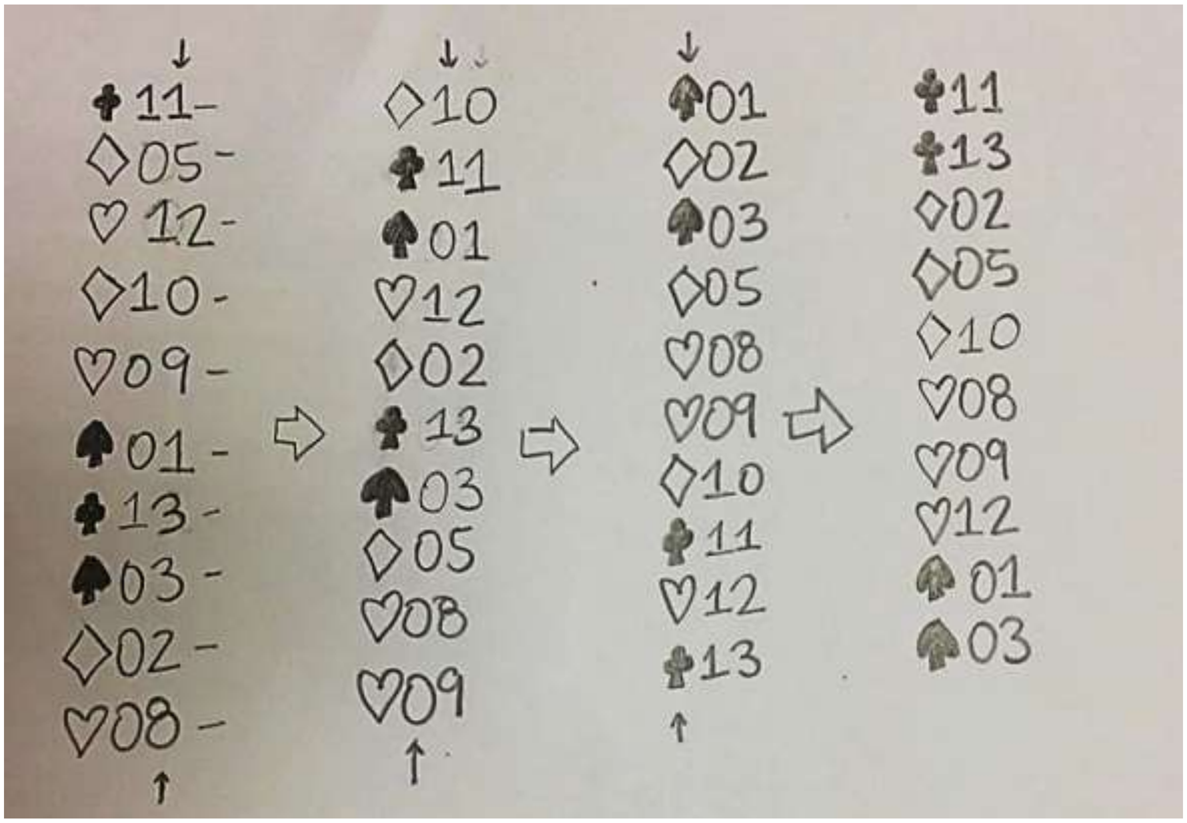
	0	1	2	3	4	5	6	7	8	9	10	11
♠7					♠7							
♥K				♥K	♠7							
♠10				♥K	♠7		♠10					
♦Q	♦Q			♥K	♠7		♠10					
♣6	♦Q			♥K	♠7		♠10	♣6				
♥J	♦Q			♥K	♠7		♠10	♣6		♥J		
♦5	♦Q			♥K	♠7		♠10	♣6		♥J	♦5	
♣3	♦Q	♣3		♥K	♠7		♠10	♣6		♥J	♦5	

2. Hash value: 22

   Card values:

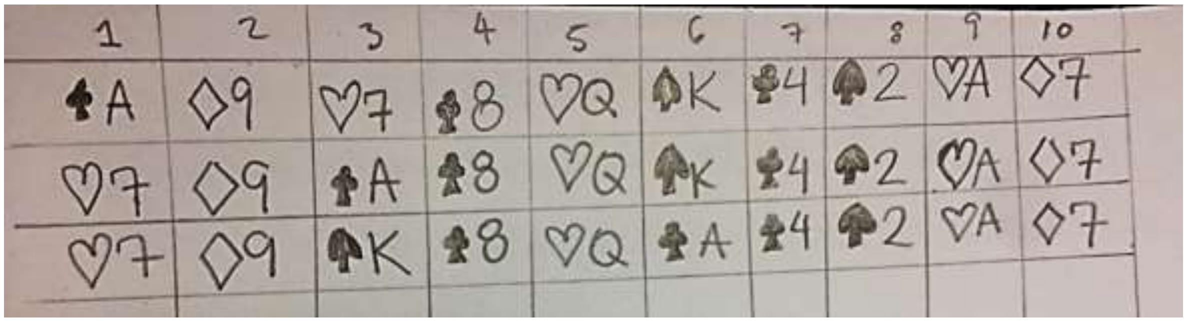
   a. ♠2
   b. ♦8
   c. ♣9

3. Radix: 10



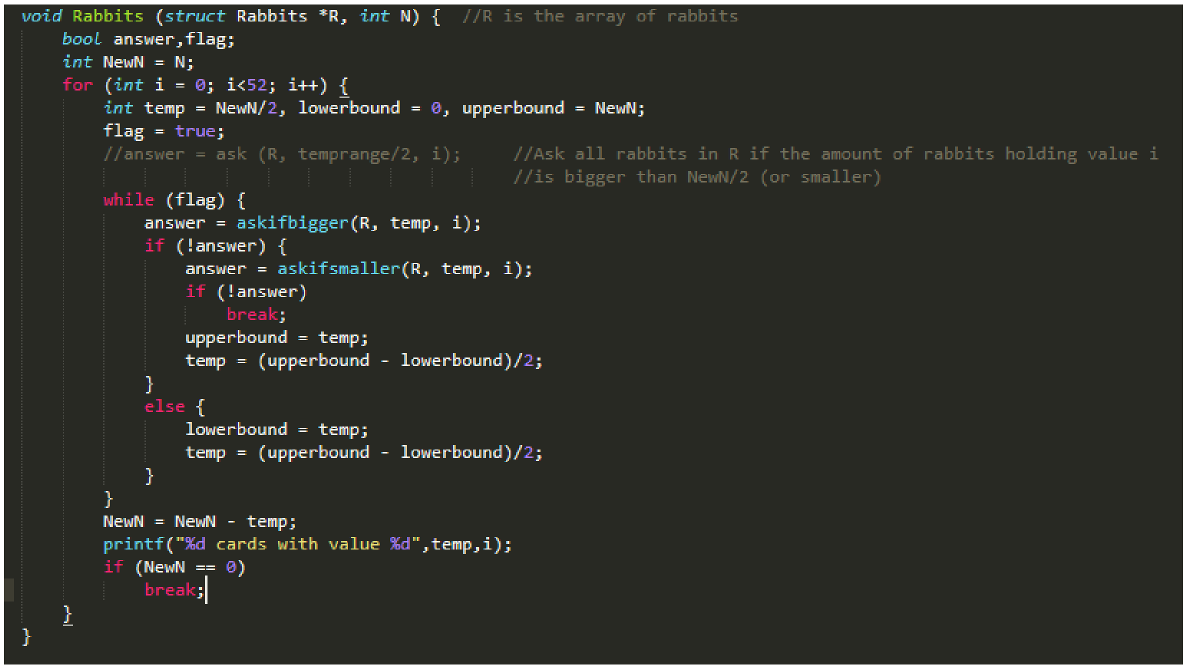
♣11	♢10	♠01	♣11
♢05	♣11	♢02	♣13
♡12	♠01	♠03	♢02
♢10	♡12	♢05	♢05
♡09	♢02	♡08	♢10
♠01	♣13	♡09	♡08
♣13	♠03	♢10	♡09
♠03	♢05	♣11	♡12
♢02	♡08	♡12	♠01
♡08	♡09	♣13	♠03

4. After calling MAX-HEAPIFY (Card_Array,1):



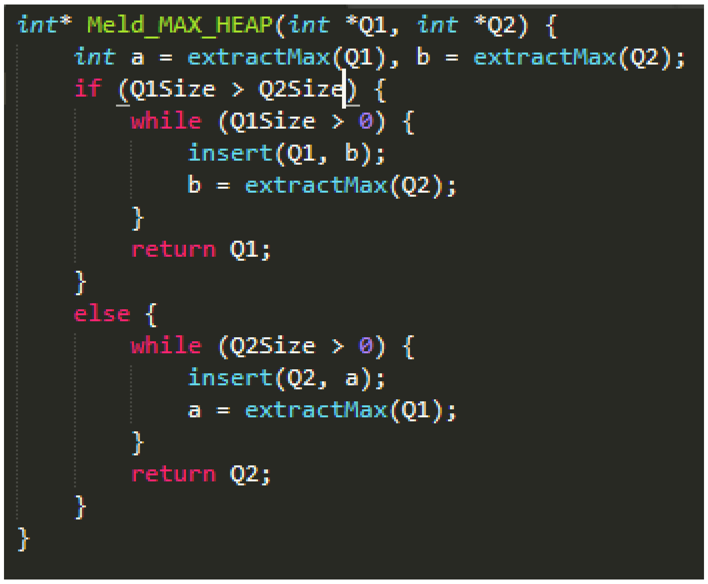
1	2	3	4	5	6	7	8	9	10
♣A	♢9	♡7	♣8	♡Q	♠K	♣4	♠2	♡A	♢7
♡7	♢9	♣A	♣8	♡Q	♠K	♣4	♠2	♡A	♢7
♡7	♢9	♠K	♣8	♡Q	♣A	♣4	♠2	♡A	♢7

5. In this problem, we need to find a way to say how many cards of each type exists amongst the rabbits. Since we know there are only 52 cards of poker, and the amount of cards total is N, we can reduce the number of questions if we use a method similar to counting sort. First, per each card in the cards available in poker, we ask all rabbits if the number of rabbits with that card is less than N/2. If the average answer is yes, we ask for N/4 cards, and so on and so on. If the answer is no, we ask for 3N/4, and keep asking in this manner. When they answer yes and no for a question, we've reached the amount, and discount this amount from the total number of rabbits available. Thus, we are making N questions per answer we want, and for each value in the 52 available we want O(log N) answers, since it's the amount of questions you ask for a binary search; thus the entire operation can be made in O(N log N) time. Extra space is O(1), since we print how many rabbits there are for each type and some extra variables to calculate the range we need to use for asking the questions for the binary search.



```c
void Rabbits (struct Rabbits *R, int N) {  //R is the array of rabbits
    bool answer,flag;
    int NewN = N;
    for (int i = 0; i<52; i++) {
        int temp = NewN/2, lowerbound = 0, upperbound = NewN;
        flag = true;
        //answer = ask (R, temprange/2, i);     //Ask all rabbits in R if the amount of rabbits holding value i
                                                //is bigger than NewN/2 (or smaller)
        while (flag) {
            answer = askifbigger(R, temp, i);
            if (!answer) {
                answer = askifsmaller(R, temp, i);
                if (!answer)
                    break;
                upperbound = temp;
                temp = (upperbound - lowerbound)/2;
            }
            else {
                lowerbound = temp;
                temp = (upperbound - lowerbound)/2;
            }
        }
        NewN = NewN - temp;
        printf("%d cards with value %d",temp,i);
        if (NewN == 0)
            break;
    }
}
```

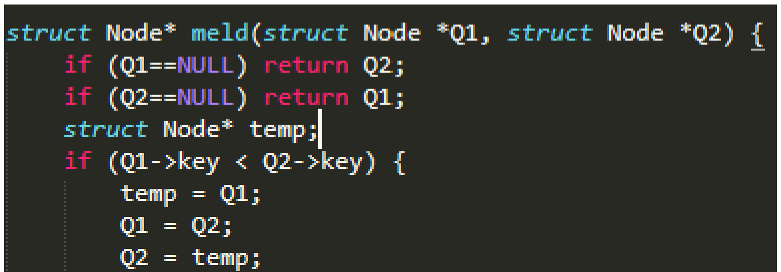
Problem 3:

1. 

```
int* Meld_MAX_HEAP(int *Q1, int *Q2) {
    int a = extractMax(Q1), b = extractMax(Q2);
    if (Q1Size > Q2Size) {
        while (Q1Size > 0) {
            insert(Q1, b);
            b = extractMax(Q2);
        }
        return Q1;
    }
    else {
        while (Q2Size > 0) {
            insert(Q2, a);
            a = extractMax(Q1);
        }
        return Q2;
    }
}
```

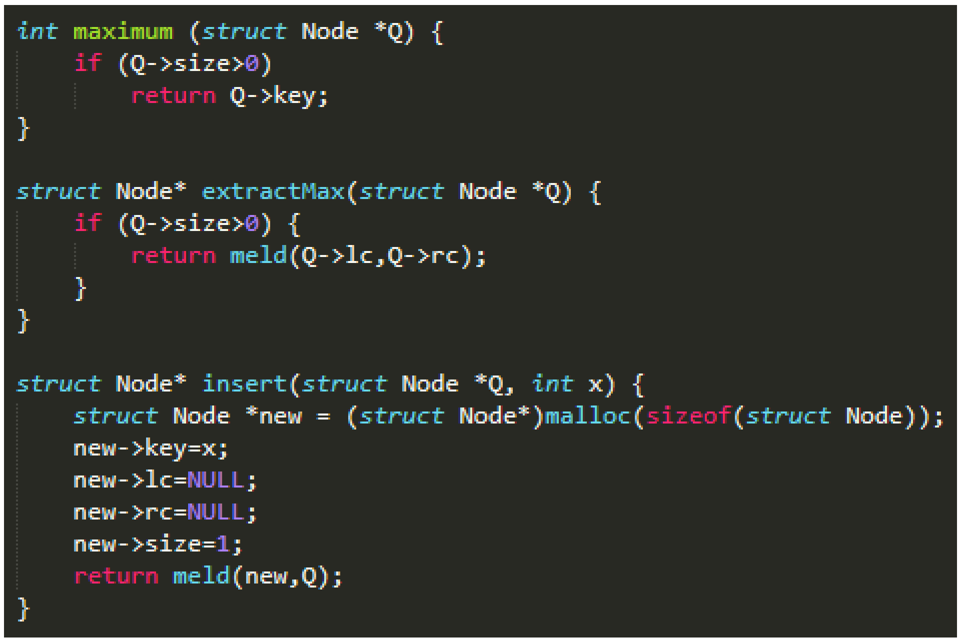
Let Q1Size and Q2Size be global variables which store the size of both heaps. Since both Q1 and Q2 are max heaps, we can extract elements from Q2 in decreasing order and insert them into Q1 or from Q1 into Q2, depending on which one is larger. Since insertions take log n time, and each value will have a smaller value than its predecessors, we can say that this takes amortized O(log^2 N) time.

2. https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/heaps.pdf



```
struct Node* meld(struct Node *Q1, struct Node *Q2) {
    if (Q1==NULL) return Q2;
    if (Q2==NULL) return Q1;
    struct Node* temp;
    if (Q1->key < Q2->key) {
        temp = Q1;
        Q1 = Q2;
        Q2 = temp;
```

It’s only O(log n), since each recursive call for meld between two heaps calls for half of the heap of the one with larger max value and the entirety of the second heap. Each comparison compares only values that can be smaller than the previous values, and since the height of the heap is log N, we will do at most 2*log N comparisons, which is still O(log N).

3. 

```c
int maximum (struct Node *Q) {
    if (Q->size>0)
        return Q->key;
}

struct Node* extractMax(struct Node *Q) {
    if (Q->size>0) {
        return meld(Q->lc,Q->rc);
    }
}

struct Node* insert(struct Node *Q, int x) {
    struct Node *new = (struct Node*)malloc(sizeof(struct Node));
    new->key=x;
    new->lc=NULL;
    new->rc=NULL;
    new->size=1;
    return meld(new,Q);
}
```