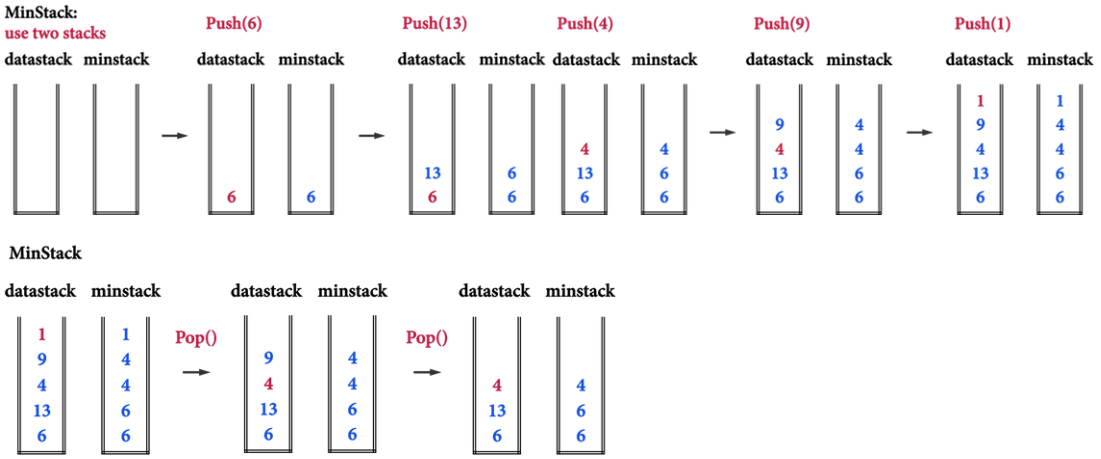
B05611047 化學三 鍾宜樺

Problem 1.-1 implement min stack

圖示:(取自 reference 1)



C_code:

```c
/*
Reference :
1. http://alrightchiu.github.io/SecondRound/stack-neng-gou-zai-o1qu-de-zui-xiao-zhi-de-minstack.html
2. https://www.geeksforgeeks.org/design-a-stack-that-supports-getmin-in-o1-time-and-o1-extra-space/
*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <stdbool.h>
struct Stack{
    int top;
    int Max_sz;
    int* data;
};
struct Stack *create_stack( int size ){
    struct Stack* stack = (struct Stack*) malloc(sizeof(struct Stack));
    stack->Max_sz = size -1;
    stack->top = -1;
    stack->data = (int*) malloc(stack->Max_sz * sizeof(int));
    return stack;
};
```

```
bool isFull( struct Stack * stack ){
    return stack->top == stack->Max_sz-1;
}

// Stack is empty when top is equal to -1
bool isEmpty(struct Stack* stack) {
    return stack->top == -1;
}
// Function to add an item to stack.  It increases top by 1
void push(struct Stack* data_stk, struct Stack* min_stk, int num){
    //因為題目說 stack 大小無限大，因此不用檢查 stk 是否滿了
    data_stk->data[++data_stk->top] = num;
    //一旦遇到比較小就加進來新的數值進 min stack，若沒有則加進原本的數值
    if( isEmpty(min_stk) == true || top(min_stk) > num ){
        min_stk->data[++min_stk->top] = num;
    }
    else{
        min_stk->data[++min_stk->top] = top(min_stk);
    }
}

// Function to remove an item from stack.  It decreases top by 1
void pop(struct Stack* data_stk, struct Stack* min_stk){
    if ( isEmpty(data_stk) == true){
        printf("stack is empty");
        return -1 ;
    }
    else{
        data_stk->data[data_stk->top--];
        min_stk->data[min_stk->top--];
    }
}

int top(struct Stack *stack) {
    if (isEmpty(stack) == true){
        printf("the stack is empty\n");
        return -1;
    }
```

```
    return stack->data[stack->top];
}

int getMin(struct Stack* min_stk){
    if(isEmpty(min_stk) == true){
        printf("the mini stack is empty");
        return -1;
    }
    return min_stk->data[min_stk->top];
}


int main(){
    //using 2 stacks and O(1) extra space
    int x = INT_MAX;// 題目假設 stack 大小無限大
    struct Stack *data_stk = create_stack(x);
    struct Stack *minstk = create_stack(x);
    //z_min_num 用來存放最小值
    int z_min_num = INT_MAX;
    /*
    程式想法：
    1.建立兩個 stack 分別是 data 跟 min，以及一個用來表示最小值的 z_min_num
    2.最一開始因為兩個 stack 都是空的，因此不用經過比大小就可以將第一個讀到的值填入兩個 stack 中
    3.接下來的讀值，data_stack 跟 min_stack 惠同時新增東西，data_stk 便是新增新讀入的值；
      而 min_stack 則需比較，若 min_stack 的 top，也就是 min_stk 最上面的數值比新讀入的值還要大，則新讀入的值便加進去 min_stack，
      若反之較小或一樣大，則 min_stack 新增他原本的 top 的讀值，因此 minstack 的 top 依舊是整個讀進列的最小值。
    4.會需要兩個 stack 都同時新增東西進來、同時丟東西出去的原因，是為了防止可能發生 data_stack 已經將最小值(eg = 1)丟出去的時候，
      min_stack 可能還沒把 1 丟出去，造成數列其實已經沒有 1 了，但 min_stack 所存放的的最小值依舊是 1，但若同步進行丟東西跟放東西的話，
      就不用對此事再有另外的處理。
    5.而欲取得當時的最小值，把 min stack 丟進 getMin()的同時，函數便會回傳，當時 min_stack 最上面那個數，及 input 數列的最小值。
    */
    return 0;
```

```
}
```

Problem 1.-2

(a) push order (1,2,3,4,5) pop order (3,5,2,1,4)		(b) push order (1,2,3,4,5) pop order (5,2,3,4,1)		(a) push order (1,2,3,4,5) pop order (2,4,3,5,1)	
operation	Queue	operation	Queue	operation	Queue
pushFront(1)	[1]	WA 1.因為 12 前後進入且 1 為第一個進去的數字，因此此二數相連。 2.5 需最先 pop out，因此 1234 必須先被 push in。 在這兩個條件下會產生下列幾種條件(視 54312 和 12345 為同種狀快討論) 54312 不合 pop out 順序 53412 不合 push in 順序 52134 不合 pop out 順序 52143 不合 push in 順序		pushFront(1)	[1]
pushBack(2)	[1,2]			pushBack(2)	[1,2]
pushFront(3)	[3,1,2]			2=popBack()	[1]
3=popFront()	[1,2]			pushFront(3)	[3,1]
pushFront(4)	[4,1,2]			pushBack(4)	[3,1,4]
pushBack(5)	[4,1,2,5]			4=popBack()	[3,1]
5=popBack()	[4,1,2]			3=popFront()	[1]
2=popBack()	[4,1]			pushBack(5)	[1,5]
1=popBack()	[4]			5=popBack()	[1]
4=popBack()	[ ]			1=popFront()	[ ]

Problem 2.-1 Complexity

以下皆設總共進入 for loop $\boldsymbol{x}$ *次：*

(a)可得到此式：$1 + 2 + \cdots + (x - 1) + x = sum \geq n$，一旦此式成立後，則跳出迴圈結束 function。

原式 $= \frac{(1+x)x}{2} \geq n \Rightarrow x = \frac{-1 \pm \sqrt{1^2 + 4*1*2n}}{2*1}$，因為 n→∞，省略相加的 const 得 x $= \sqrt{2}n^{1/2}$ ，又 by define，$C_1 * n^{\frac{1}{2}} \leq x \leq C_2 * n^{\frac{1}{2}}$ as $n \geq n_0$。舉例：取 $C_1 = 1;\ C_2 = 2; n_0 = 2$，因此得$\boldsymbol{\theta(n^{1/2})}$。

(b)分兩部分討論。Situation A : i<n 先成立因此跳出 while loop ,Situation B : j^ 3<n 先成立因此跳出 while loop。

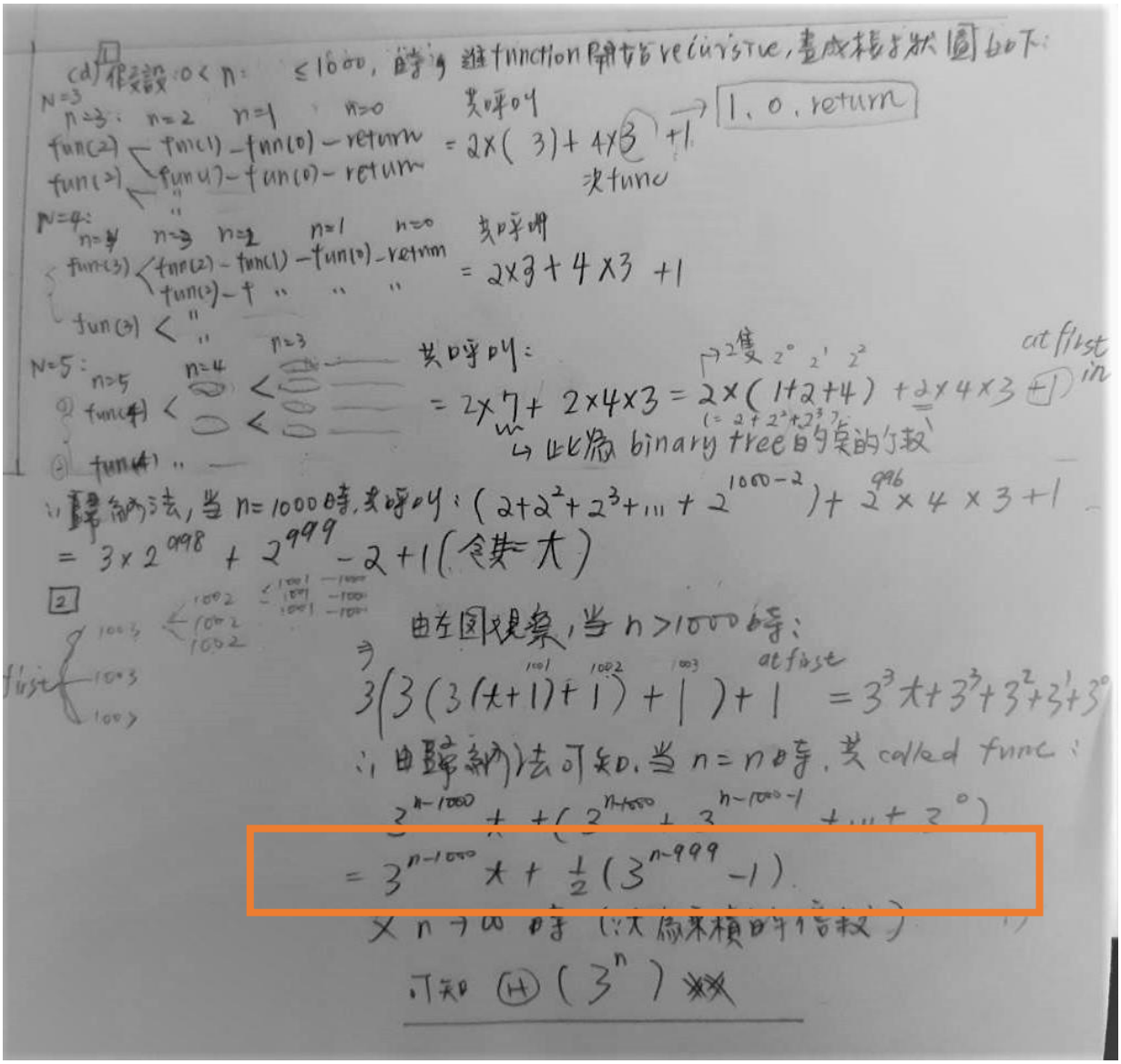
Situation A : $2^x \geq n \Rightarrow x \geq \log_2 n = \frac{\log n}{\log 2} \approx 3.32 * \log n$ ，因此$\theta(\log_2 n)$，欲符合定義，舉例：則取$C_1 = 2; C_2 = 4; n_0 = 2$;即成立。

Situation B :$x^3 \geq n \Rightarrow x \geq n^{1/3}$，因此$\theta(n^{1/3})$，且取$C_1 = 1; C_2 = 2; n_0 = 2;$

又 Situation A 會比 B 先達成 if n 很大，因此取$\boldsymbol{\theta(\log n)}$。

(c)經由觀察歸納：$2^{2^x} \geq n$，會跳過 while loop。整理得：$x \geq 11.04 * \log n$，因此取$\boldsymbol{\theta(\log n)}$，欲符合定義，舉例：取$C_1 = 5; C_2 \leq 15; n_0 = 2;$即成立。

(d)因為不好打所以用寫的：



延續上圖框框中的部分：$\cong (c_1 + c_2) * 3^n \ \ (as \ \ n \to \infty) = \ \ c_3 * 3^n$，得 $\boldsymbol{\theta(3^n)}$，欲符合定義，舉例：取$C_1 = 5; C_2 \leq C_3 \ ; n_0 = 2;$ 即成立。

Problem 2.-2

(a) If f(n) = O(i(n)), g(n) = O(j(n)), then f(n)−g(n) = O(i(n)−j(n))

令 f(n) = 2n，g(n) = 3n^3；則 i(n) = n，j(n) = n^3。

By 教授的 ppt：

Definition [Theta]:$\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2, n_0 such that $\mathbf{0 \le c_1 g(n) \le f(n) \le c_2 g(n)}$ for all $n \ge n_0\}$

然而，如粗體字所示部分，f(n)需大於 0，然此時的 f(n) = 2n − 3n^3，當 n>1 時，其值必小於 0，因此無法討論 bigO，故此題假設**不正確**。

(b) f(n) = O(i(n)), g(n) = O(j(n)), then f(n)*g(n) = O(i(n)*j(n))

對，證明如下：

By define, f(n) = c1*i(n); g(n) =c2*j(n)

f(n)*g(n) = c1*i(n)* c2*j(n) = c3* i(n)*j(n)，符合 big O’s define.

(c) If f(n) = O(g(n)), then 2^f(n) = O(2^g(n))

反例：

令 f(n) = 2n, g(n) = n;

2^(f(n)) = 2^(2n) = 4^n，by define，2^(f(n)) = O(4^n) != O(2^n) = O(2^(g(n)))

(d) f(n) + g(n) = Θ(max(f(n),g(n)))

對，證明：max(f(n),g(n)) <= f(n) + g(n) <=2* max(f(n),g(n))

(e) log(n!) = Θ(nlogn)

對，證明如下：

$\because (n/2)^{n/2} \le n! \le n^n,$

$\therefore n/4 \log(n) = n/2 \log(n^{1/2}) \le n/2 \log(n/2) \le \log(n!) \le n \log(n)$ (Ref：https://www.zhihu.com/question/27300145)

Problem 2.-3 求 T(n)

(ref: http://ind.ntou.edu.tw/~litsnow/al98/pdf/Algorithm-Ch4-Recurrences.pdf)

Guess : T(n)=O(nlgn)，並且由此推得下界，得 theta

Upper bound:

claim : that there exist positive constants c>0 and n0 such that T(n)≤cnlgn for all n≥n0

when $n=1$, $T(1) \le c_1 1 \lg 1 = 0$, which is odds with $T(1)=1$

since the recurrence does not depend directly on $T(1)$, we can replace $T(1)$ by $T(2)=4$ and $T(3)=5$ as the base cases

$T(2) \le c_1 2 \lg 2$ and $T(3) \le c_1 3 \lg 3$ for any choice of $c_1 \ge 2$

thus, we can choose $c_1=2$ and $n_0=2$

assume $T(\lfloor n/2 \rfloor) \le c_2 \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$ for $\lfloor n/2 \rfloor$

then, $T(n) \le 2(c_2 \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n$

$\le c_2 n \lg(n/2) + n$

$= c_2 n \lg n - c_2 n \lg 2 + n$

$= c_2 n \lg n - c_2 n + n$

$\le c_2 n \lg n$

the last step holds as long as $c_2 \ge 1$

There exist positive constants $c = \max\{2,1\}$ and $n_0=2$ such that $T(n) \le cn \lg n$ for all $n \ge n_0$

又由上述證明可知 c = max{2,1}，因此欲得下界，Omega，即將 c 取小於 1 但大於 0 之常數即可，因此得證，T(n) = nlgn。

Problem 3

1. C-style pseudo code：(Insert 和 delete 的概念)

Insert(newElement)

```
Find node in linked list e
If e.numElements < e.data.size
    e.data.push(newElement)
    e.numElements ++
else
    create new Node e1
    move final half of e.data into e1.data
    e.numElements = e.numElements / 2
    e1.data.push(newElement)
    e1.numElements = e1.data.size / 2 + 1
```

Delete(element)

```
Find element in node e
e.data.remove(element)
e.numElements --
while e.numElements < e.data.size / 2
    put element from e.next.data in e.data
    e.next.numElements --
    e.numElements ++
if e.next.numElements < e.next.data.size / 2
    merge nodes e and e.next
    delete node e.next
```

(Ref: https://brilliant.org/wiki/unrolled-linked-list/)

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

typedef struct Node{
    int size; int array[100]; //K max
    struct Node *next; } node;
node *head;
int K ;


void init(int n, int A[]){
    //constructs the unrolled linked list by the input list.
    K = 300;
    head = (node*)malloc(sizeof(node)); head->size = 0;
     head->next = NULL;
     node *now = head; for(int i = 0; i < n; i ++){
         if(now->size == K){ node *tmp = (node*)malloc(sizeof(node));
         tmp->size = 0;
         tmp->next = NULL;
         now->next = tmp;
         now = tmp; }
         now->array[now->size ++] = A[i];
          }
         }

void locate(int *pos, node **now){
    //locates which node pos is at and its array index.
    while(*pos > (*now)->size){
        *pos -= (*now)->size;
        *now = (*now)->next;
        }
    }

int query(int pos){
    node *now = head;
     locate(&pos, &now);
     return now->array[pos - 1];
```

```
}

//Create a new node
node *CreateNode(void)
{
    // 宣告指標 ptr，並指向 NODE 大小的空間
    node *newNode = (node*)malloc(sizeof(node));
    if(newNode == NULL)
        exit(1);
    return (newNode);//回傳這個空間的指標
};


//insert Node
node *InsertNode(node *tmp){
    node *NewNode = CreateNode();
    NewNode->next = NULL;
    if (tmp->next == NULL){
        tmp->next = NewNode;
    }
    else{
        NewNode->next = tmp->next;
        tmp->next = NewNode;
    }
    return(head);
};

void Insert(int pos, char element){
    //index the site
    node *tmp = head;
    int sigma_sz = head->size;
    int count = 0;
    while(sigma_sz - pos < 0 && tmp->next != NULL){
        tmp = tmp->next;
        sigma_sz += tmp->size;
        count ++;
    }
    int put_place = pos - sigma_sz + tmp->size;
    //max size 未滿，可以直接加
```

```
    if(tmp->size < K){
        //往後平移
        for(int i = tmp->size -1 ; i > put_place - 1; i--){
            tmp->array[i+1] = tmp->array[i];
            tmp->array[i] = NULL;
        }
        tmp->array[put_place] = element;
        tmp->size++;
    }
    else if (tmp->size == K ){ // tmp->size == K 已經滿了的時候
        //initial
        head = InsertNode(tmp);
        node *Next = tmp->next;
        Next->size = 0;

        for(int i = 0 ; i < K; i++){
            Next->array[i] = NULL;
        }
        //減少 tmp 的 size//////////////////////////////////////////////
        tmp->size = tmp->size / 2;
        if(put_place <= K/2){//欲新增資料的位置仍在 tmp 裡面
            //移動位置到 next 裡面
            for (int i = 0 ; i < K/2; i++){
                Next->array[i] = tmp->array[i+K/2];
                tmp->array[i+K/2] = NULL;
                Next->size ++;
            }
            //空出要被插入的那格，後面資料向後移動\n
            for (int i = tmp->size - 1; i > (put_place - 1) ; i--){
                tmp->array[i+1] = tmp->array[i];
                tmp->array[i] = NULL;
            }
            //"放入資料
            tmp->array[put_place] = element;
            tmp->size++;
        }
        else{
            //先一 put_place 前面的
```

```
            for(int i = K/2; i < put_place; i++){
                //移過去後清空
                Next->array[i-K/2] = tmp->array[i];
                tmp->array[i] = NULL;
            }
            //放入資料
            Next->array[put_place - K/2] = element;
            //再移 put_place 後面的
            for(int i = put_place; i < K; i++){
                //移動過去後要清空 QQ
                Next->array[ i - K/2 + 1] = tmp->array[i];
                tmp->array[i] = NULL;
            }
            Next->size = K/2+1;
        }
    }
}

node *Free(node *head){
    free(head);
};

//delete Node
node *DeleteNode ( node *tmp){
    //3 situations, before head, tail, middle
    node *help = head;
    if(tmp == head){
        head = head->next;
    }
    else{
        while ( help->next != tmp){
            help = help->next;
        }
        // delete the last
        if(tmp->next == NULL)
            help->next = NULL;
        else//delete middle
        {
```

```
            help->next = tmp->next;
//            printf("ok\n");
        }

    }
    free(tmp);
    return(head);
};

void Delete (int pos){
    //index the site
    node *tmp = head;
    int sigma_sz = head->size;
    while(sigma_sz - pos <= 0 && tmp->next != NULL){
        tmp = tmp->next;
        sigma_sz += tmp->size;
    }
    int del_plc = pos - sigma_sz + tmp->size;
    tmp->array[del_plc] = NULL;
    for(int i = del_plc + 1; i < tmp->size; i++){
        tmp->array[i-1] = tmp->array[i];//往前推一格
        tmp->array[i] = NULL;
    }
    tmp->size--;

    //一次移動 1 個，且小心 tmp->next != NULL
    while(tmp->size < K/2 && tmp->next != NULL ){
        tmp->array[tmp->size] = tmp->next->array[0];
        tmp->next->array[0] = NULL;
        tmp->size ++;
        tmp->next->size --;
        for(int i = 0; i < tmp->next->size; i++){
            tmp->next->array[i] = tmp->next->array[i+1];
            tmp->next->array[i+1] = NULL;
        }
        if( tmp->next->size < K/2 ){
            //合併
            for(int i = 0; i < tmp->next->size; i++){
```

```
                tmp->array[ i + tmp->size] = tmp->next->array[i];
                tmp->next->array[i] = NULL;
                tmp->size ++;
            }
             head = DeleteNode(tmp->next);
        }
    }
}
//使用第五題的 main 測試
int main (){
    int read_lines,count = 0;
    int to_do; // 1, 2, 3
    int pos;
    int stp;
    char element;
    scanf("%d", &read_lines);
    //initial
    head = (node *)malloc(sizeof(node));
    head->size = 0;
    head->next = NULL;
    for(int i = 0 ; i < K; i++){
        head->array[i] = NULL;
    }
    while(count < read_lines){
       scanf("%d %d", &to_do, &pos);
        if(to_do == 1 ){ // to_do == 1, 加入東西
            scanf(" %c", &element);
            Insert(pos,element);
        }
        else (to_do == 2){
            Delete(pos);
        }
        count ++;
    }
    return 0;
}
```

2. Unrolled Linked List

Insert :

insert 發生最糟糕的狀況是前面的測資，讓 node 的分布變成全部都只裝到 K 的一半的值，假設 K 開 300(開 300 是配合第五題的測資數目 QQ)，則每個 node 都含有 150 or 151 個測資，(就是分裂之後 Insert 一個值進去後就不再碰觸到這兩個 node)，這樣會讓 node 數非常多，導致 index 的時間增加。且最後一個 node->size 是 K，再加上要 insert 的地方剛好是最後一個 node 的 array[0]。(最後面提到 array[0]，只是其中一個的狀況，insert 在 array 的任何位置所花費的時間最後相加起來都一樣)。

接下來，假設要 insert 的位置發生在最後一個 node 的第一個位置上。Index 到 Node 需要將全部 node 跑一次，這時候會花上 $O(\frac{n}{\frac{K}{2}})$(const * n -> O(n)，但是必定小於 n)。接著判斷這個 Node 是不是小於 K 或等於 K 是 const time，判斷滿了後，因為滿了，所以此時這個 node 會分裂成兩個 node 且須將一半的 data 一到 node->next 中，需花 K/2 的時間(又 n 趨近於無限，則 K 視為 const)，最後一步就是 insert。須將整個 node 中的 data 往後移動一個位置空出 array[0]給 element，因此又會花 K/2 的時間。而尋找真正為坐落在哪裡因為是用寫成一條計算式(計算 Put place)，固也是花 O(1)的時間。

總花費：$O\left(\frac{n}{\frac{K}{2}}\right) + 2 * O(1)$+O(K/2)+O(K/2) = O(2n/K)+O(1)+O(K)必小於 o(n^2)

//因此 K 開在 sqrt(n)的時候會跑得比較快些總平均來講(?)QWQ

Delete：

Delete 最糟的狀況一開始和 insert 一樣，是前面的測資，讓 node 的分布變成全部都只裝到 K 的一半的值，假設 K 開 300(開 300 是配合第五題的測資數目 QQ)，則每個 node 都含有 150 or 151 個測資，(就是分裂之後 Insert 一個值進去後就不再碰觸到這兩個 node)，這樣會讓 node 數非常多，導致 index 的時間增加。而尋找真正為坐落在哪裡因為是用寫成一條計算式(計算 Put place)，固也是花 O(1)的時間。

欲刪除的資料在倒數第二個 node 的 array[0]的位置，且 node->size = K/2，node->next->size 也是 K/2。刪除 node->array[0]雖然只花 O(1)，但因為他在第一個位置上，接下來要把後面的 K/2-1 個測資往前移；又因為當 node->size 小於 K/2 時，會進 code 的 while，從 node->next 拿第一個 data 進來 node 補，這會導致 node 裡面需要整個被往前移動一次，花費 K/2 的時間。這時候悲劇又發生了，發現 node->next->size 此時也小於 K/2 了，因此要將這兩個 Node merge 起來才不會導致 node 數太多，因此這時候又要再花 K/2-1 的時間，將他們全部移動到 node 裡面。最後得到下面此式子：

總花費：$O(\frac{n}{\frac{K}{2}})$+2*O(1)+O(K/2-1)+O(K/2)+O(K/2-1) < o(n^2) // if n ->∞

3.作業太晚開始寫，快寫不完惹 QQ 下次一定早早開始寫作業 QQ