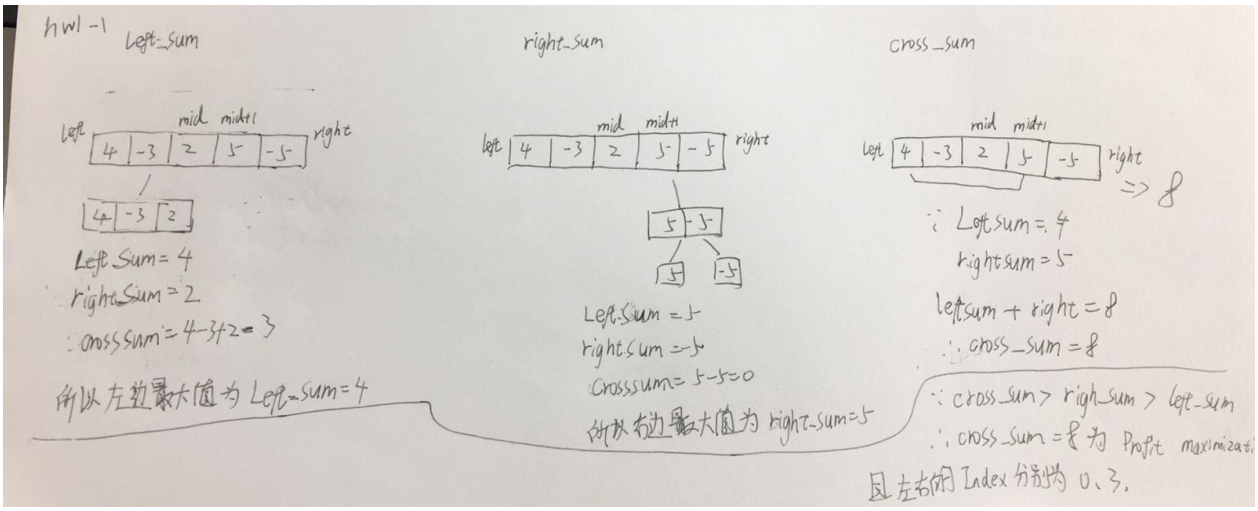
# 演算法作業一

## 作業 1-1 圖解說明



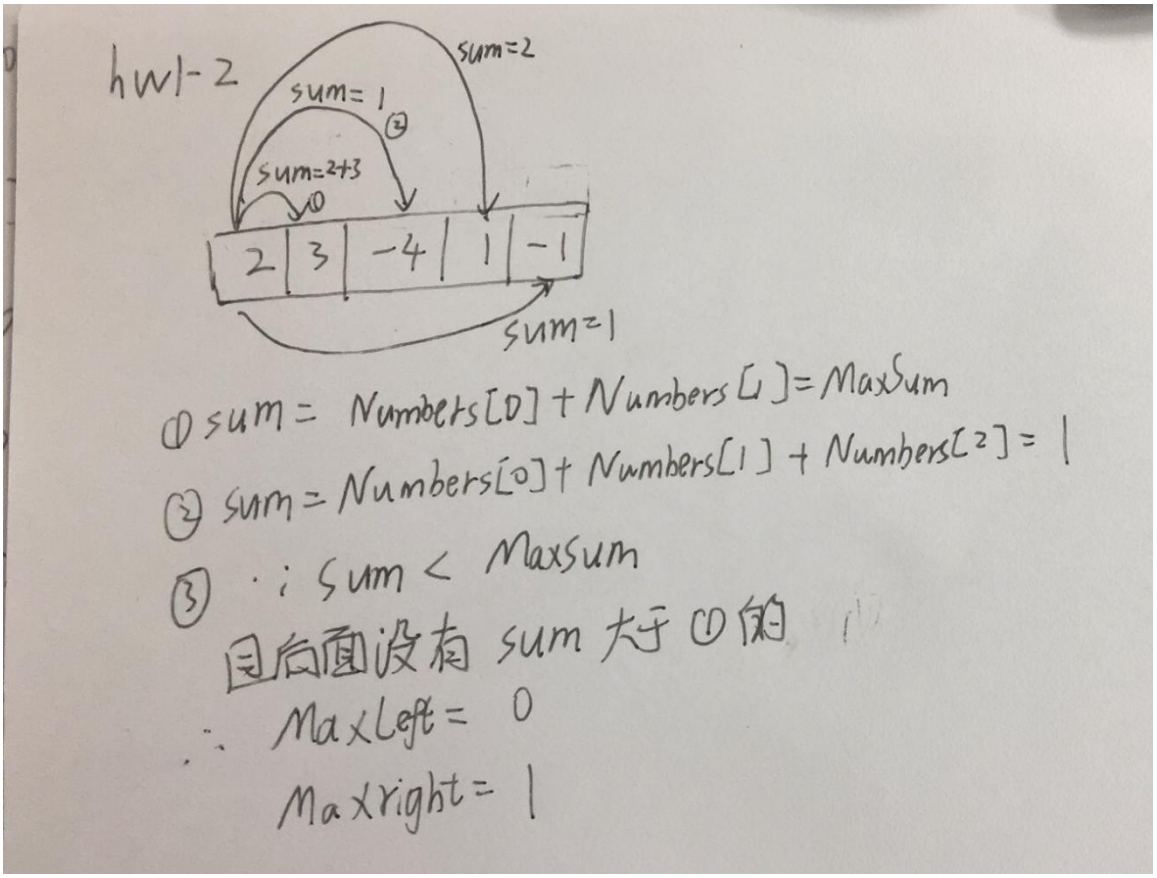

## 作業 1-1 虛擬碼

```
MAXSUM(valus,left,right,result)
if left==right
    result.left=result.right=left //base case]
    result.sum=Values[right]
else
    left_sum = -∞
    right_sum = -∞
    Sum = 0
    mid = (left+right)/2
    (left_sum.left,left_sum.right,left_sum.sum)=
        MAXSUM(values,left,mid,&left_Sum)
    (right_sum.right,right_sum.right,right_sum.sum)=
        MAXSUM(values,mid+1,right,&right_Sum)
    for i =mid downto left
        sum = sum + values[i]
        if sum greater than leftsum
            cross_sum.left = i
            leftsum = The largest of the (sum,leftsum)
    sum = 0
    for i =mid+1 upto right
        sum = sum + values[i]
        if sum greater than rightsum
            cross_sum.right = i
            rightsum = The largest of the (sum,rightsum)
    if (left_Sum.sum >= right_Sum.sum && left_Sum.sum >= cross_Sum.sum)
        Stroe left_Sum
    else if (left_Sum.sum >= left_Sum.sum && right_Sum.sum >= cross_Sum.sum)
        Stroe right_Sum;
    else
        Stroe cross_Sum;
```

## 作業 1-1 程式碼

```cpp
#include <iostream>
#include <limits.h>
#include <algorithm>
using namespace std;
//定義一個Struct儲存左右兩邊的索引值
struct RESULT{
    int left,right,sum;
};
/*
int* A是要輸入測試的數組、
int left是區間最左邊的Index、
int right是區間最右邊的Index
struct RESULT* result是儲存最大的Subarray的和
*/
void MaxSum(int* Values,int left,int right,struct RESULT* result)
{
    if(left == right)//遞歸終止條件，當左Index等於右Index遞歸終止。
    {
        result->left=result->right=left;//因爲分割到最後只剩一個，所以left=right
        result->sum=Values[right];//分割最大Subarray
    }else
    {
        int leftsum = INT_MIN;//leftsum用來暫時儲存左邊subarray的和，並賦值INT_MIN，這樣才可以賦值負數
        int rightsum = INT_MIN;//leftsum用來暫時儲存右邊subarray的和，並賦值INT_MIN，這樣才可以賦值負數
        int sum=0; //初始化sum=0，用來暫時儲存sum值
        int mid = (left+right)/2;//mid是區間的中間值
        /*cross_sum用來儲存跨中間的sum值
        left_sum用來儲存左邊的sum值
        right_sum用來儲存右邊的sum值
        */
        struct RESULT cross_Sum,left_Sum,right_Sum;
        MaxSum(Values,left,mid,&left_Sum);//分割中間值之前的前半段區間
        MaxSum(Values,mid+1,right,&right_Sum);//分割中間值之後的後半段區間
        //下面兩個for循環是分割中間區間的cross sum
        for(int i= mid;i >= left;i--)//計算左邊區間尋找最大的序列和
        {
            sum += Values[i];//把輸入的i值相加
            if( sum > leftsum ){//如果sum>leftsum就執行下面的if語句
                cross_Sum.left=i;//儲存最左邊的索引值
                leftsum = max(sum,leftsum);//比對sum和leftsum哪個更大，儲存比較大的一個值在leftsum中
            }
        }
        sum=0;
        for(int i=mid+1;i<=right;i++)
        {
            sum += Values[i];//計算右邊區間尋找最大的序列和
            if( sum > rightsum ){//如果sum>rightsum就執行下面的if語句
                cross_Sum.right=i;//儲存當前最右邊的right的索引值
                rightsum = max(sum,rightsum);//比對sum和lrightsum哪個更大，儲存比較大的一個值在rightsum中
            }
        }
        cross_Sum.sum = leftsum + rightsum;//將lefsum和rightsum相加儲存到cross_sum.sum的值
        if (left_Sum.sum >= right_Sum.sum && left_Sum.sum >= cross_Sum.sum)     // 左邊和最大
            *result=left_Sum;//如果left_sum比較大，就儲存left_sum到*result
        else if (left_Sum.sum >= left_Sum.sum && right_Sum.sum >= cross_Sum.sum)   // 右邊和最大
            *result=right_Sum;//如果right_sum比較大，就儲存right_sum到*result
        else                                                    // 中間和最大
            *result=cross_Sum;//如果cross_sum值比較大，就儲存cross_sum到*result中
    }
}
int main()
{
    int num = 0;
    while (scanf("%d", &num) != EOF){//輸入array裏面的數字總數
        int Values[num];//初始化Values array
        for(int i=0;i<num;i++)
            scanf("%d",&Values[i]);//依次輸入array裏面的值
        struct RESULT result;
        MaxSum(Values,0,num-1,&result);//執行MaxSum函數
        printf("%d %d %d\n",result.left,result.right,result.sum);//打印出最左的索引值，打印出最右的索引值和Sum
    }
    return 0;
}
```

## 作業 1-2 圖解說明




## 作業 1-2 虛擬碼

```
Mian()
    While(input number of array)
        Numbers(number)
        Left = 0
        sum = 0
        maxLeft = 0
        maxRight = 0
        maxSum = 0
        While(i less than number)
            Input Numbers[i]
            sum = sum + Numbers[i]
            While( sum greater than maxSum)
                maxRight = i
                maxSum = sum
                maxLeft = left
            While( sum less than 0)
                Left = i + 1
                Sum = 0
            i++
        Output maxLeft,maxRight,maxSum
```

## 作業 1-2 程式碼

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main(){
    int n;
    while (scanf("%d", &n) != EOF){//輸入array裏面有多少個數字
        vector<int> Numbers(n);//新定義一個Numbers(n)的vector容器來儲存數組
        int Left = 0; //定義一個臨時的儲存left的索引值
        int sum = 0;//定義一個臨時的儲存sum的索引值
        int maxLeft = 0;//定義一個儲存左邊最大的索引值maxLeft
        int maxRight = 0;//定義一個儲存右邊最大的索引值maxRight
        int maxSum = 0;//定義一個儲存最大的和的值maxSum
        int i = 0;
        while(i<n){//當i<n時，執行while循環
            scanf("%d",&Numbers[i]);//依次向Numbers容器輸入值
            sum = sum + Numbers[i];//將輸入Numbers容器的值相加
            while ( sum > maxSum )//當sum>maxSum時，
            {
                maxRight = i;  //把maxRight設成當前的i值
                maxSum = sum;  //把當前的sum值儲存進maxSum
                maxLeft = Left;  //把當前的Left索引值儲存進maxLeft
            }
            while (sum < 0){  //當sum<0時
                Left = i + 1; //把當前i值+1儲存進Left
                sum = 0;    //把當前的sum值置0
            }
            i++; //執行完后，i自加1
        }
        printf("%d %d %d\n", maxLeft, maxRight, maxSum);//依次輸出maxLeft、maxRight、maxSum
        Numbers.clear();//清空容器Numbers
    }
    return 0;//置0
}
```