HW6 Readme

輸入處理函式

- 主要是藉由讀入字串，再用**atoi**的function將字元轉成數字放進整數型態的陣列，最後再用題目的方法去做排序。

```
void processinput( char* input )
{
    int lenth = strlen( input ) ;
    int flag = 0 ;
    char num[ 30 ] ;
    int idxnum = 0 ;
    for( int i = 0 ; i < lenth ; i++ )
    {
        char tmp = input[ i ] ;
        if( isdigit( tmp ) )
        {
            num[ idxnum ] = tmp ;
            idxnum++ ;
            flag = 1 ;
        }
        else
        {
            if( flag == 1 )
            {
                flag = 0 ;
                num[ idxnum ] = '\0' ;
                idxnum = 0 ;
                int val = atoi( num ) ;
                array[ idxa ] = val ;
                idxa++ ;
            }
        }
    }
    int val = atoi( num ) ;
    array[ idxa ] = val ;
    idxa++ ;
}
```

6-1 思路

- quick sort 主要是利用**選擇pivot**的方式去左右分堆，在不斷的左右分堆中去排序list
- 找取PIVOT方式：選擇那堆中最左邊的，並有兩個index：left&right，分別找左堆中大於pivot和右堆中小於pivot，並交換。
- **recursive** soring：利用遞迴的方式左右分堆

CODE 分析

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int array[ 2000 ] ;
int n ;
int idxa ;
void swap( int a , int b )
{
    int t = array[ a ] ;
    array[ a ] = array[ b ] ;
    array[ b ] = t ;
}
int partition( int le , int rt , int key )
{
    int i = le , j = rt ;
    while( i < j )
    {
        while( array[ j ] > key && i <= j ) j-- ;
        while( array[ i ] <= key && i < j ) i++ ;
        if( i < j ) swap( i , j ) ;
    }
    swap( le , i ) ;
    for( int i = 0 ; i< n ; i++ )
    {
        if( i == n - 1 )
        {
            printf("%d",array[ i ]) ;
        }
        else
        printf("%d, ",array[ i ]) ;
    }
    printf("\n") ;
    return i ;
}
void quicksort( int le , int rt , int key )
{
    if( le >= rt ) return ;
    int pos = partition( le , rt , key ) ;
    quicksort( le , pos - 1 , array[ le ] ) ;
    quicksort( pos +1 , rt , array[ pos + 1 ] ) ;
}
void processinput( char* input )
{
    int lenth = strlen( input ) ;
    int flag = 0 ;
    char num[ 30 ] ;
    int idxnum = 0 ;
    for( int i = 0 ; i < lenth ; i++ )
    {
        char tmp = input[ i ] ;
        if( isdigit( tmp ) )
        {
            num[ idxnum ] = tmp ;
            idxnum++ ;
            flag = 1 ;
        }
        else
        {
            if( flag == 1 )
            {
                flag = 0 ;
                num[ idxnum ] = '\0' ;
                idxnum = 0 ;
                int val = atoi( num ) ;
                array[ idxa ] = val ;
                idxa++ ;
            }
        }
    }
    int val = atoi( num ) ;
    array[ idxa ] = val ;
    idxa++ ;
}
int main()
{
    char input[ 100000 ] ;
    fgets( input , 100000 , stdin ) ;
    processinput( input ) ;
    n = idxa ;
    quicksort( 0 , n -1 , array[ 0 ] ) ;
    return 0;
}
```

6-2 思路

- selection sort 是從尚未排序的list中一個一個找最小的，並和現在第n個元素做交換
- select the minimun element from unsorted as the n-th element

CODE 分析

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int array[ 2000 ] ;
int n ;
int idxa ;
void selection_sort( void  )
{
    for (int i=0; i<n-1; i++)
    {
        int min_idx = i;
        for (int j=i+1; j<n; j++)
        {
            if (array[  j  ] < array[  min_idx  ] )
            {
                min_idx = j;
            }
        }
        // swap
        int temp = array[ min_idx ];
        array[ min_idx ] = array[ i ];
        array[ i ] = temp;
        for( int i = 0 ; i< n ; i++ )
        {
            if( i == n - 1 )
            {
                printf("%d",array[ i ]) ;
            }
            else
            printf("%d, ",array[ i ]) ;
        }
        printf("\n") ;
    }
}
// select the minimun element from unsorted list
void processinput( char* input )
{
    int lenth = strlen( input ) ;
    int flag = 0 ;
    char num[ 30 ] ;
    int idxnum = 0 ;
    for( int i = 0 ; i < lenth ; i++ )
    {
        char tmp = input[ i ] ;
        if( isdigit( tmp ) )
        {
            num[ idxnum ] = tmp ;
            idxnum++ ;
            flag = 1 ;
        }
        else
        {
            if( flag == 1 )
            {
                flag = 0 ;
                num[ idxnum ] = '\0' ;
                idxnum = 0 ;
                int val = atoi( num ) ;
                array[ idxa ] = val ;
                idxa++ ;
            }
        }
    }
    int val = atoi( num ) ;
    array[ idxa ] = val ;
    idxa++ ;
}
int main()
{
    char input[ 10000 ] ;
    fgets( input , 10000 , stdin ) ;
    processinput( input ) ;
    n = idxa ;
    selection_sort( ) ;
    return 0;
}
```

6-3 思路

- inserion sort 在 sort第 n 個element時，會去檢查**前面n-1已經sort的list**，找到適合的位置並插入。
- put n-th element in its correct place by scanning the n-1 sorted element

CODE 分析

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int array[ 2000 ] ;
int n ;
int idxa ;
void insertion_sort( )
{
    int i, key, j;
    for ( i = 1 ; i < n; i++)
    {
        key = array[i];
        j = i - 1;
        /**Move elements of arr[0..i-1], that are
          greater than key, to one position ahead
          of their current position **/
        while ( j >= 0 && array[j] > key )
        {
            array[j + 1] = array[j];
            j = j - 1;
        }
        array[j + 1] = key;
        for( int i = 0 ; i < n ; i++ )
        {
            if( i == n - 1 ) printf("%d",array[ i ] ) ;
            else printf("%d, ",array[ i ] ) ;
        }
        printf("\n") ;
    }
}
void processinput( char* input )
{
    int lenth = strlen( input ) ;
    int flag = 0 ;
    char num[ 30 ] ;
    int idxnum = 0 ;
    for( int i = 0 ; i < lenth ; i++ )
    {
        char tmp = input[ i ] ;
        if( isdigit( tmp ) )
        {
            num[ idxnum ] = tmp ;
            idxnum++ ;
            flag = 1 ;
        }
        else
        {
            if( flag == 1 )
            {
                flag = 0 ;
                num[ idxnum ] = '\0' ;
                idxnum = 0 ;
                int val = atoi( num ) ;
                array[ idxa ] = val ;
                idxa++ ;
            }
        }
    }
    int val = atoi( num ) ;
    array[ idxa ] = val ;
    idxa++ ;
}
int main()
{
    char input[ 10000 ] ;
    fgets( input , 10000 , stdin ) ;
    fgets( input , 10000 , stdin ) ;
    processinput( input ) ;
    n = idxa ;
    for( int i = 0 ; i < n ; i++ )
    {
        if( i == n - 1 ) printf("%d",array[ i ] ) ;
        else printf("%d, ",array[ i ] ) ;
    }
    printf("\n") ;
    insertion_sort( ) ;
    return 0;
}
```

6-4 思路

- 這題是mergesort，並且印出步驟。
  我利用了迴圈的方式，而不是遞迴，去做mergesort所需要的 **divide&conquer**
- 接著題目需要輸出minimun gap ， 因此我再跑一次迴圈去計算。

CODE 分析

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int array[ 2000 ] ;
int n ; // for array lenth
int idxa ;
int FindMin(int x, int y)
{
    if( x < y ) return x ;
    else return y ;
}
void MyMergeSort( void )
{
    int *list = array;
    int *sorted= (int *) malloc( n * sizeof(int));
    // use pointer for swapping
    int  segment ;
    int  start ;
    for (segment = 1; segment < n ; segment += segment)
    {
        for ( start = 0 ; start < n ; start += segment * 2)
        {
            int low = start ;
            int mid = FindMin(start +  segment , n) ;
            int high = FindMin(start +  segment * 2 , n);
            int k = low ;
            int start1 = low ;
            int end1 = mid ;
            int start2 = mid ;
            int end2 = high ;
            while ( start1 < end1 && start2 < end2 )
            {
                 if( list[ start1 ] < list[ start2 ] )
                 {
                     sorted[ k++ ] = list[ start1++ ] ;
                 }
                 else
                 {
                     sorted[ k++ ] = list[ start2++ ] ;
                 }
             }
            while (start1 < end1)
            {
                sorted[k++] = list[start1++];
            }
            while (start2 < end2)
            {
                sorted[k++] = list[start2++];
            }
        }
        for( int i = 0 ; i< n ; i++ )
        {
            if( i == n - 1 )
            {
                printf("%d",sorted[ i ]) ;
            }
            else
            printf("%d, ",sorted[ i ]) ;
        }
        printf("\n") ;

        int *temp = list;
        list = sorted ;
        sorted = temp ;
        // swap sorted list and origin list
    }
    // level
}

void processinput( char* input )
{
    int lenth = strlen( input ) ;
    int flag = 0 ;
    char num[ 30 ] ;
    int idxnum = 0 ;
    for( int i = 0 ; i < lenth ; i++ )
    {
        char tmp = input[ i ] ;
        if( isdigit( tmp ) )
        {
            num[ idxnum ] = tmp ;
            idxnum++ ;
            flag = 1 ;
        }
        else
        {
            if( flag == 1 )
            {
                flag = 0 ;
                num[ idxnum ] = '\0' ;
                idxnum = 0 ;
                int val = atoi( num ) ;
                array[ idxa ] = val ;
                idxa++ ;
            }
        }
    }
    int val = atoi( num ) ;
    array[ idxa ] = val ;
    idxa++ ;

}
int main()
{
    char input[ 10000 ] ;
    fgets( input , 10000 , stdin ) ;
    processinput( input ) ;
    n = idxa ;
    MyMergeSort(  ) ;
    int min = 0 ;
    for( int i = 1 ; i < n ; i++ )
    {
        if( i == 1 )
        {
            min = array[  1  ] - array[  0  ] ;
        }
        else
        {
            if( array[ i  ] - array[ i - 1 ] < min )
            {
                min = array[ i ] - array[ i - 1 ] ;
            }
        }
    }
    // find minimum gap
    printf("Minimum gap: %d\n",min) ;
    return 0;
}
# MERGE SORT 的部分我有另外開一個空間去處理，
  分成已經merge 的 資料( sorted ) 和尚未merge的資料( list )
```