

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。

```
1 void processinput( char* input )
2 {
3     int lenth = strlen( input ) ;
4     int flag = 0 ;
5     char num[ 30 ] ;
6     int idxnum = 0 ;
7     for( int i = 0 ; i < lenth ; i++ )
8     {
9         char tmp = input[ i ] ;
10        if( isdigit( tmp ) )
11        {
12            num[ idxnum ] = tmp ;
13            idxnum++ ;
14            flag = 1 ;
15        }
16        else
17        {
18            if( flag == 1 )
19            {
20                flag = 0 ;
21                num[ idxnum ] = '\0' ;
22                idxnum = 0 ;
23                int val = atoi( num ) ;
24                array[ idxa ] = val ;
25                idxa++ ;
26            }
27        }
28    }
29    int val = atoi( num ) ;
30    array[ idxa ] = val ;
31    idxa++ ;
32 }
```

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

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <string.h>
4  int array[ 2000 ] ;
5  int n ;
6  int idxa ;
7  void swap( int a , int b )
8  {
9      int t = array[ a ] ;
10     array[ a ] = array[ b ] ;
11     array[ b ] = t ;
12 }
13 int partition( int le , int rt , int key )
14 {
15     int i = le , j = rt ;
16     while( i < j )
17     {
18         while( array[ j ] > key && i <= j ) j-- ;
19         while( array[ i ] <= key && i < j ) i++ ;
20         if( i < j ) swap( i , j ) ;
21     }
22     swap( le , i ) ;
23     for( int i = 0 ; i < n ; i++ )
24     {
25         if( i == n - 1 )
26         {
27             printf("%d",array[ i ]) ;
28         }
29         else
30             printf("%d, ",array[ i ]) ;
31     }
32     printf("\n") ;
33     return i ;
34 }
35 void quicksort( int le , int rt , int key )
36 {
37     if( le >= rt ) return ;
38     int pos = partition( le , rt , key ) ;
39     quicksort( le , pos - 1 , array[ le ] ) ;
40     quicksort( pos + 1 , rt , array[ pos + 1 ] ) ;
41 }
42 void processinput( char* input )
43 {
44     int length = strlen( input ) ;
45     int flag = 0 ;
46     char num[ 30 ] ;
47     int idxnum = 0 ;
48     for( int i = 0 ; i < length ; i++ )
49     {
50         char tmp = input[ i ] ;
51         if( isdigit( tmp ) )
52         {
53             num[ idxnum ] = tmp ;
54             idxnum++ ;
55             flag = 1 ;
56         }
57         else
58         {
59             if( flag == 1 )
60             {
61                 flag = 0 ;
62                 num[ idxnum ] = '\0' ;
63                 idxnum = 0 ;
64                 int val = atoi( num ) ;
65                 array[ idxa ] = val ;
66                 idxa++ ;

```

```
66         idxa++ ;
67     }
68 }
69 }
70 int val = atoi( num ) ;
71 array[ idxa ] = val ;
72 idxa++ ;
73 }
74 int main()
75 {
76     char input[ 100000 ] ;
77     fgets( input , 100000 , stdin ) ;
78     processinput( input ) ;
79     n = idxa ;
80     quicksort( 0 , n -1 , array[ 0 ] ) ;
81     return 0;
82 }
```

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

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  int array[ 2000 ] ;
5  int n ;
6  int idxa ;
7  void selection_sort( void )
8  {
9      for (int i=0; i<n-1; i++)
10     {
11         int min_idx = i;
12         for (int j=i+1; j<n; j++)
13         {
14             if (array[ j ] < array[ min_idx ] )
15             {
16                 min_idx = j;
17             }
18         }
19         // swap
20         int temp = array[ min_idx ];
21         array[ min_idx ] = array[ i ];
22         array[ i ] = temp;
23         for( int i = 0 ; i < n ; i++ )
24         {
25             if( i == n - 1 )
26             {
27                 printf("%d",array[ i ] ) ;
28             }
29             else
30                 printf("%d, ",array[ i ] ) ;
31         }
32         printf("\n") ;
33     }
34 }
35 // select the minimum element from unsorted list
36 void processinput( char* input )
37 {
38     int length = strlen( input ) ;
39     int flag = 0 ;
40     char num[ 30 ] ;
41     int idxnum = 0 ;
42     for( int i = 0 ; i < length ; i++ )
43     {
44         char tmp = input[ i ] ;
45         if( isdigit( tmp ) )
46         {
47             num[ idxnum ] = tmp ;
48             idxnum++ ;
49             flag = 1 ;
50         }
51         else
52         {
53             if( flag == 1 )
54             {
55                 flag = 0 ;
56                 num[ idxnum ] = '\0' ;
57                 idxnum = 0 ;
58                 int val = atoi( num ) ;
59                 array[ idxa ] = val ;
60                 idxa++ ;
61             }
62         }
63     }
64     int val = atoi( num ) ;
65     array[ idxa ] = val ;
66     idxa++ ;

```

```
66         idx++;  
67     }  
68     int main()  
69     {  
70         char input[ 10000 ] ;  
71         fgets( input , 10000 , stdin ) ;  
72         processinput( input ) ;  
73         n = idxa ;  
74         selection_sort( ) ;  
75         return 0;  
76     }
```

6-3 思路

- insertion 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

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  int array[ 2000 ] ;
5  int n ;
6  int idxa ;
7  void insertion_sort( )
8  {
9      int i, key, j;
10     for ( i = 1 ; i < n; i++)
11     {
12         key = array[i];
13         j = i - 1;
14         /**Move elements of arr[0..i-1], that are
15          greater than key, to one position ahead
16          of their current position **/
17         while ( j >= 0 && array[j] > key )
18         {
19             array[j + 1] = array[j];
20             j = j - 1;
21         }
22         array[j + 1] = key;
23         for( int i = 0 ; i < n ; i++ )
24         {
25             if( i == n - 1 ) printf("%d",array[ i ] ) ;
26             else printf("%d, ",array[ i ] ) ;
27         }
28         printf("\n") ;
29     }
30 }
31 void processinput( char* input )
32 {
33     int length = strlen( input ) ;
34     int flag = 0 ;
35     char num[ 30 ] ;
36     int idxnum = 0 ;
37     for( int i = 0 ; i < length ; i++ )
38     {
39         char tmp = input[ i ] ;
40         if( isdigit( tmp ) )
41         {
42             num[ idxnum ] = tmp ;
43             idxnum++ ;
44             flag = 1 ;
45         }
46         else
47         {
48             if( flag == 1 )
49             {
50                 flag = 0 ;
51                 num[ idxnum ] = '\0' ;
52                 idxnum = 0 ;
53                 int val = atoi( num ) ;
54                 array[ idxa ] = val ;
55                 idxa++ ;
56             }
57         }
58     }
59     int val = atoi( num ) ;
60     array[ idxa ] = val ;
61     idxa++ ;
62 }
63 int main()
64 {
65     char input[ 10000 ] ;
66     fgets( input, 10000, stdin ) ;

```

```
66     fgets( input , 10000 , stdin ) ;
67     processinput( input ) ;
68     n = idxa ;
69     for( int i = 0 ; i < n ; i++ )
70     {
71         if( i == n - 1 ) printf("%d",array[ i ] ) ;
72         else printf("%d, ",array[ i ] ) ;
73     }
74     printf("\n") ;
75     insertion_sort( ) ;
76     return 0;
77 }
```

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

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  int array[ 2000 ] ;
5  int n ; // for array lenth
6  int idxa ;
7  int FindMin(int x, int y)
8  {
9      if( x < y ) return x ;
10     else return y ;
11 }
12 void MyMergeSort( void )
13 {
14     int *list = array;
15     int *sorted= (int *) malloc( n * sizeof(int));
16     // use pointer for swapping
17     int segment ;
18     int start ;
19     for (segment = 1; segment < n ; segment += segment)
20     {
21         for ( start = 0 ; start < n ; start += segment * 2)
22         {
23             int low = start ;
24             int mid = FindMin(start + segment , n) ;
25             int high = FindMin(start + segment * 2 , n);
26             int k = low ;
27             int start1 = low ;
28             int end1 = mid ;
29             int start2 = mid ;
30             int end2 = high ;
31             while ( start1 < end1 && start2 < end2 )
32             {
33                 if( list[ start1 ] < list[ start2 ] )
34                 {
35                     sorted[ k++ ] = list[ start1++ ] ;
36                 }
37                 else
38                 {
39                     sorted[ k++ ] = list[ start2++ ] ;
40                 }
41             }
42             while (start1 < end1)
43             {
44                 sorted[k++] = list[start1++];
45             }
46             while (start2 < end2)
47             {
48                 sorted[k++] = list[start2++];
49             }
50         }
51         for( int i = 0 ; i < n ; i++ )
52         {
53             if( i == n - 1 )
54             {
55                 printf("%d",sorted[ i ] ) ;
56             }
57             else
58                 printf("%d, ",sorted[ i ] ) ;
59         }
60         printf("\n") ;
61
62         int *temp = list;
63         list = sorted ;
64         sorted = temp ;
65         // swap sorted list and origin list
66     }

```

```
66     }
67     // level
68 }
69
70 void processinput( char* input )
71 {
72     int lenth = strlen( input ) ;
73     int flag = 0 ;
74     char num[ 30 ] ;
75     int idxnum = 0 ;
76     for( int i = 0 ; i < lenth ; i++ )
77     {
78         char tmp = input[ i ] ;
79         if( isdigit( tmp ) )
80         {
81             num[ idxnum ] = tmp ;
82             idxnum++ ;
83             flag = 1 ;
84         }
85         else
86         {
87             if( flag == 1 )
88             {
89                 flag = 0 ;
90                 num[ idxnum ] = '\0' ;
91                 idxnum = 0 ;
92                 int val = atoi( num ) ;
93                 array[ idxa ] = val ;
94                 idxa++ ;
95             }
96         }
97     }
98     int val = atoi( num ) ;
99     array[ idxa ] = val ;
100     idxa++ ;
101 }
102
103 int main()
104 {
105     char input[ 10000 ] ;
106     fgets( input , 10000 , stdin ) ;
107     processinput( input ) ;
108     n = idxa ;
109     MyMergeSort( ) ;
110     int min = 0 ;
111     for( int i = 1 ; i < n ; i++ )
112     {
113         if( i == 1 )
114         {
115             min = array[ 1 ] - array[ 0 ] ;
116         }
117         else
118         {
119             if( array[ i ] - array[ i - 1 ] < min )
120             {
121                 min = array[ i ] - array[ i - 1 ] ;
122             }
123         }
124     }
125     // find minimum gap
126     printf("Minimum gap: %d\n",min) ;
127     return 0;
128 }
129 # MERGE SORT 的部分我有另外開一個空間去處理，
130 分成已經merge 的 資料( sorted ) 和尚未merge的資料( list )
```