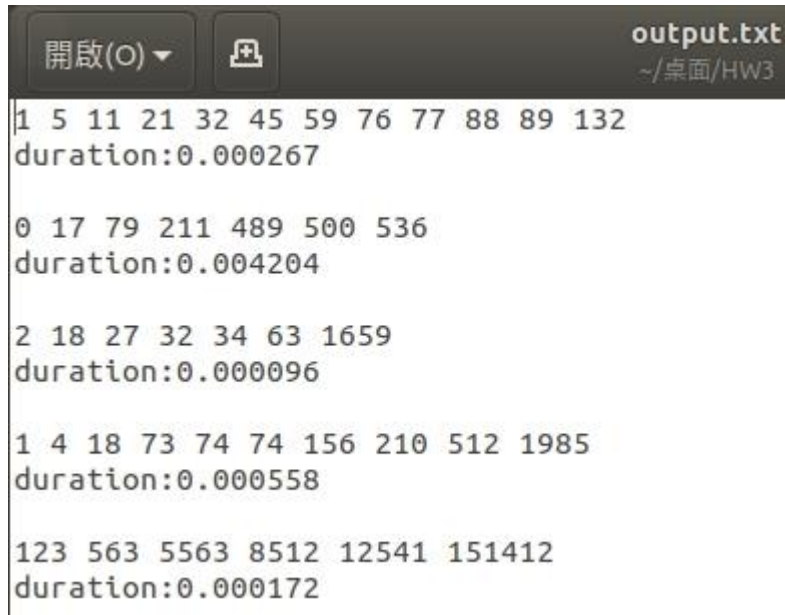


- Output



```
開啟(O) ▾  output.txt
~/桌面/HW3

1 5 11 21 32 45 59 76 77 88 89 132
duration:0.000267

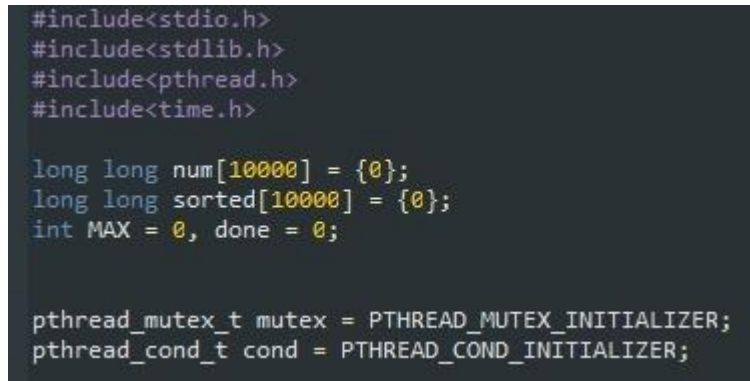
0 17 79 211 489 500 536
duration:0.004204

2 18 27 32 34 63 1659
duration:0.000096

1 4 18 73 74 74 156 210 512 1985
duration:0.000558

123 563 5563 8512 12541 151412
duration:0.000172
```

- Code



```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<time.h>

long long num[10000] = {0};
long long sorted[10000] = {0};
int MAX = 0, done = 0;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

首先 include 所有會用到的 header 檔，然後建立 global variable。num 用來存 testcase.txt 中的數字，sorted 用來存 merge sort 完的數字，MAX 用來存該列有幾個數字，done 表示有幾個 thread 已經 sort 完。接下來兩行則是 pthread_cond_wait 的 mutex 和 condition。

```

int main(int argc, char *argv[])
{
    char c;
    int i = 0, negative = 0;
    FILE* fin = fopen(argv[1], "r");
    FILE *fout = fopen(argv[2], "w");

    while ((c = fgetc(fin)) != EOF) {
        if(c==' '){
            if(negative) num[i] = -num[i];
            negative = 0;
            i++;
            num[i] = 0;
        }
        else if(c=='\n'){
            if(negative) num[i] = -num[i];
            MAX = i;
            clock_t start, end;

            start = clock();
            pthread_t sortA, sortB, merge;
            pthread_create(&sortA, NULL, merge_sort_A, (void*)NULL);
            pthread_create(&sortB, NULL, merge_sort_B, (void*)NULL);
            pthread_create(&merge, NULL, merge_sort_M, (void*)NULL);
            pthread_join(merge, NULL);
            end = clock();

            int j;
            for (j = 0; j < MAX; j++){
                fprintf(fout, "%lld ", sorted[j]);
            }
            fprintf(fout, "%lld", sorted[j]);
            fprintf(fout, "\nduration:%f\n\n", (double)(end-start) / CLOCKS_PER_SEC);
            negative = 0;
            done = 0;
            i = 0;
            num[i] = 0;
        }
        else if(c=='\r'){
        }
        else if(c=='-'){
            negative = 1;
        }
        else{
            long long b = c-'0';
            num[i] = num[i]*10 + b;
        }
    }
    fclose(fin);
    fclose(fout);

    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond);

    return 0;
}

```

c 用來存 input 的字元，i 表示讀到該列第幾個數字，negative 用來表示是否為負數。argv[1]為 testcase.txt，argv[2]為 output.txt。

讀到數字時，先把 c 減掉'0'，從字元 c 轉為數字 b，然後將前一個數字*10 再加上 b 存入 num[i]。

讀到負號時，把 negative 設為 1，表示為負數。

多處理一個讀到'\r'時，是因為在 linux 環境下讀取 testcase.txt 時，換行會有'\r'。(但我發現在 windows 環境下只會有'\n')

讀到空格時，表示這個數字已結束。若 negative 為 1，就把它加上負號，然後 negative 就可以歸零。i++到該列下一個數字，並先進行 num[i] = 0 的初始化。

讀到換行時，表示該列最後一個數字已結束。若 negative 為 1，一樣把它加上負號，negative 就可以歸零。然後把 MAX 設為目前的 i，就知道該行有幾個數字，便可以開始進行 merge sort。

進行 merge sort 前，先宣告三個 pthread 變數。sortA 用來 sort 數列的前半段，sortB 用來 sort 數列的後半段，merge 則用來做最後前半、後半數列的 merge。

然後用 pthread_create 來建立子執行緒，第一個參數為指向執行緒識別符的指標，第二個參數用來設定執行緒屬性，第三個參數是執行緒執行 function 的起始地址，最後一個參數是執行 function 的引數。

```
void* merge_sort_A(void *arg)
{
    merge_sort(0, MAX/2);

    pthread_mutex_lock(&mutex);
    done++;
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&mutex);
}

void* merge_sort_B(void *arg)
{
    merge_sort(MAX/2 + 1, MAX);

    pthread_mutex_lock(&mutex);
    done++;
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&mutex);
}
```

merge_sort_A 就是 pthread_t sortA 用來做 merge sort 的 function。因為它是負責前半段數列的，所以只會做 0 到 MAX/2 的 merge sort。並用 mutex 來避免同時修改 done 的問題，等它做完後，就會發 signal 給在等待的 pthread_t merge。merge_sort_B 基本上跟 merge_sort_A 做的事情一模一樣，只是換成數列的後半段，所以是 MAX/2+1 到 MAX。

```

void merge_sort(int start, int end)
{
    if (start < end){
        int mid = (end + start) / 2;
        merge_sort(start, mid);
        merge_sort(mid + 1, end);
        Merge(start, mid, end);
    }
}

```

這就是一個普通的 merge sort fuction，把大數列切一半成為兩個小數列，然後把切好的兩個小數列再各自切一半，不斷重複，直到每個小數列都只剩一個數字，再用 Merge 來合併。

```

void Merge(int start, int mid, int end)
{
    int lenA = mid - start + 1;
    int lenB = end - mid;

    int *A = (int*)malloc(sizeof(int)*lenA);
    int *B = (int*)malloc(sizeof(int)*lenB);

    for(int i = 0; i < lenA; i++){
        A[i] = num[start + i];
    }
    for(int i = 0; i < lenB; i++){
        B[i] = num[mid + 1 + i];
    }

    int i = 0, j = 0, index = start;

    while(i < lenA && j < lenB){
        if(A[i] < B[j])
            num[index++] = A[i++];
        else
            num[index++] = B[j++];
    }

    while(i < lenA){
        num[index++] = A[i++];
    }
    while(j < lenB){
        num[index++] = B[j++];
    }
}

```

這也是一個普通的 Merge fuction，它所做的事情就是，把兩個小數列合併排序成一個大數列，這邊就不多加說明。

```

void* merge_sort_M(void *arg)
{
    pthread_mutex_lock(&mutex);
    while (done < 2){
        pthread_cond_wait(&cond, &mutex);
    }
    pthread_mutex_unlock(&mutex);

    int i = 0, j = MAX/2 + 1, index = 0;
    while(i <= MAX/2 && j <= MAX){
        if(num[i] < num[j])
            sorted[index++] = num[i++];
        else
            sorted[index++] = num[j++];
    }
    while(i <= MAX/2){
        sorted[index++] = num[i++];
    }
    while(j <= MAX){
        sorted[index++] = num[j++];
    }
}

```

這是最後一個會進到的 function，先檢查 signal 若不是 done 等於 2 時（merge_sort_A 和 merge_sort_B 都完成），就會繼續在 while loop 中 wait。merge_sort_M 負責把 sortA 和 sortB 分別處理完的前半和後半數列，再合併排序成最終的完整數列 sorted，做法和 Merge function 一樣。

- Note

```

haiyin@haiyin-virtual-machine:~/桌面/HW3$ gcc hw3.c -pthread -o hw3.o
gcc: error: -pthread: 沒有此一檔案或目錄
gcc: error: -o: 沒有此一檔案或目錄
haiyin@haiyin-virtual-machine:~/桌面/HW3$ gcc hw3.c -o hw3.o -lpthread
haiyin@haiyin-virtual-machine:~/桌面/HW3$ ./hw3.o testcase.txt output.txt
haiyin@haiyin-virtual-machine:~/桌面/HW3$

```

我用作業講義中的指令 gcc hw3.c -pthread -o hw3.o，會出現 error，但改用 gcc hw3.c -o hw3.o -lpthread 就沒問題了。