

一、code 截圖

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <pthread.h>
5
6  typedef struct node {
7      int low;
8      int high;
9  } NODE;
10
11 void* merge_sort(void *);
12 void* merge_final(void *);
13 void merge(void *);
14
15 int input[10005];
16
17 int main(int argc, char *argv[]) {
18     FILE *fptr1, *fptr2;
19     fptr1 = fopen(argv[1], "r");
20     fptr2 = fopen(argv[2], "w");
21
22
23     int temp;
24
25     while(fscanf(fptr1, "%d", &temp) != EOF) {
26         clock_t start, terminate;
27         int num, mid;
28         NODE m, n, p;
29         pthread_t left, right, merge_thread;
30
31         char garbage;
32
33         num = 0;
34         input[num++] = temp;
35         fscanf(fptr1, "%c", &garbage);
36         while(garbage != '\n') {
37             fscanf(fptr1, "%d", &temp);
```

```

38         input[num++] = temp;
39         fscanf(fptr1, "%c", &garbage);
40     }
41     start = clock();
42
43     mid = (num-1) / 2;
44     m.low = 0;
45     m.high = mid;
46     n.low = mid+1;
47     n.high = num-1;
48     p.low = 0;
49     p.high = num-1;
50
51     pthread_create(&left, NULL, merge_sort, &m);
52     pthread_create(&right, NULL, merge_sort, &n);
53
54     pthread_join(left, NULL);
55     pthread_join(right, NULL);
56
57     pthread_create(&merge_thread, NULL, merge_final, &p);
58     pthread_join(merge_thread, NULL);
59
60     terminate = clock();
61
62     for (int i = 0; i < num; i++)
63         fprintf(fptr2, "%d ", input[i]);
64     fprintf(fptr2, "\n");
65     fprintf(fptr2, "duration: %f\n\n", (terminate - start) / (double)CLOCKS_PER_SEC);
66
67     for (int i = 0; i < num; i++)
68         input[i] = 0;
69 }
70
71 fclose(fptr1);
72 fclose(fptr2);
73

```

```

74     return 0;
75 }
76
77 void* merge_sort(void *a) {
78     NODE *range = (NODE *)a;
79     int mid = range->low + (range->high - range->low) / 2;
80     NODE left, right, combine;
81
82     left.low = range->low;
83     left.high = mid;
84     right.low = mid+1;
85     right.high = range->high;
86     combine.low = range->low;
87     combine.high = range->high;
88
89     if (range->low < range->high) {
90         merge_sort(&left);
91         merge_sort(&right);
92         merge(&combine);
93     }
94 }
95
96 void merge(void *a) {
97     NODE *range = (NODE *)a;
98     int mid = range->low + (range->high - range->low) / 2;
99     int left_size = mid-range->low + 1;
100    int right_size = range->high - mid;
101    int *left, *right;
102    int count_left = 0, count_right = 0;
103    int pos = range->low;
104
105    left = (int *)malloc(left_size*sizeof(int));
106    right = (int *)malloc(right_size*sizeof(int));
107
108    for(int i = 0; i < left_size; i++) {
109        left[i] = input[range->low+i];

```

```

110     }
111
112     for(int i = 0; i < right_size; i++) {
113         right[i] = input[mid+1+i];
114     }
115
116     while(count_left < left_size && count_right < right_size)
117     {
118         if(left[count_left] <= right[count_right])
119             input[pos++] = left[count_left++];
120         else
121             input[pos++] = right[count_right++];
122     }
123
124     while(count_left < left_size)
125         input[pos++] = left[count_left++];
126     while(count_right < right_size)
127         input[pos++] = right[count_right++];
128 }
129
130 void* merge_final(void *a) {
131     NODE *range = (NODE *)a;
132     int mid = range->low + (range->high - range->low) / 2;
133     int left_size = mid-range->low + 1;
134     int right_size = range->high - mid;
135     int *left, *right;
136     int count_left = 0, count_right = 0;
137     int pos = range->low;
138
139     left = (int *)malloc(left_size*sizeof(int));
140     right = (int *)malloc(right_size*sizeof(int));
141
142     for(int i = 0; i < left_size; i++) {
143         left[i] = input[range->low+i];
144     }
145
146     for(int i = 0; i < right_size; i++) {

```

```

146         right[i] = input[mid+1+i];
147     }
148
149     while(count_left < left_size && count_right < right_size) {
150         if(left[count_left] <= right[count_right])
151             input[pos++] = left[count_left++];
152         else
153             input[pos++] = right[count_right++];
154     }
155     int left_size
156     while(count_left < left_size)
157         input[pos++] = left[count_left++];
158     while(count_right < right_size)
159         input[pos++] = right[count_right++];
160 }

```

二、說明

宣告一個 global variable “input”，其為 10005 個 entries 的 int 矩陣，用以存放每一行 testcase。宣告一個叫 NODE 的 structure，裡面有兩個 int，分別存 merge sort 時範圍上、下界的 index。宣告三個 function，void* merge_sort(void *)、void* merge_final(void *) 及 void merge(void *)，merge_sort 跟 merge_final 用來執行 thread，merge_sort 是給剛開始的兩個 threads 執行，那兩個 thread 結束後再開第三個 thread，用 merge_final 把前面的結果 merge 起來。而 merge 是在執行 merge_sort 時 recursion 會用到的 function，用來將左、右兩半邊的 testcase 分別排序好。

在 main 裡，先檢查是不是 EOF，不是的話，先用 clock_t 紀錄開始時間，並且讀一行 testcase，再來把 testcase 分為左右兩段的上、下界 index 紀錄好，創造兩個 threads 分別叫做 left、right，透過 merge_sort function 對 left、right 進行 merge sort，做完後結束這兩個 threads。再 create 第三個 thread 叫做 merge_thread，透過 merge_final function 將前面兩個 threads 運算結果 merge 起來，做完後結束第三個 thread。接著，紀錄運算好的時間並 print 結果。然後將 input array 重新 initiate。重複以上動作直至遇到 EOF。

三、結果截圖

output.txt - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

1 5 11 21 32 45 59 76 77 88 89 132
duration: 0.000200

0 17 79 211 489 500 536
duration: 0.000100

2 18 27 32 34 63 1659
duration: 0.000086

1 4 18 73 74 74 156 210 512 1985
duration: 0.000092

123 563 5563 8512 12541 151412
duration: 0.000121