Assignment 1
2015313754 TaehyungGil (길태형)

October 25, 2020

My code consists of three parts.

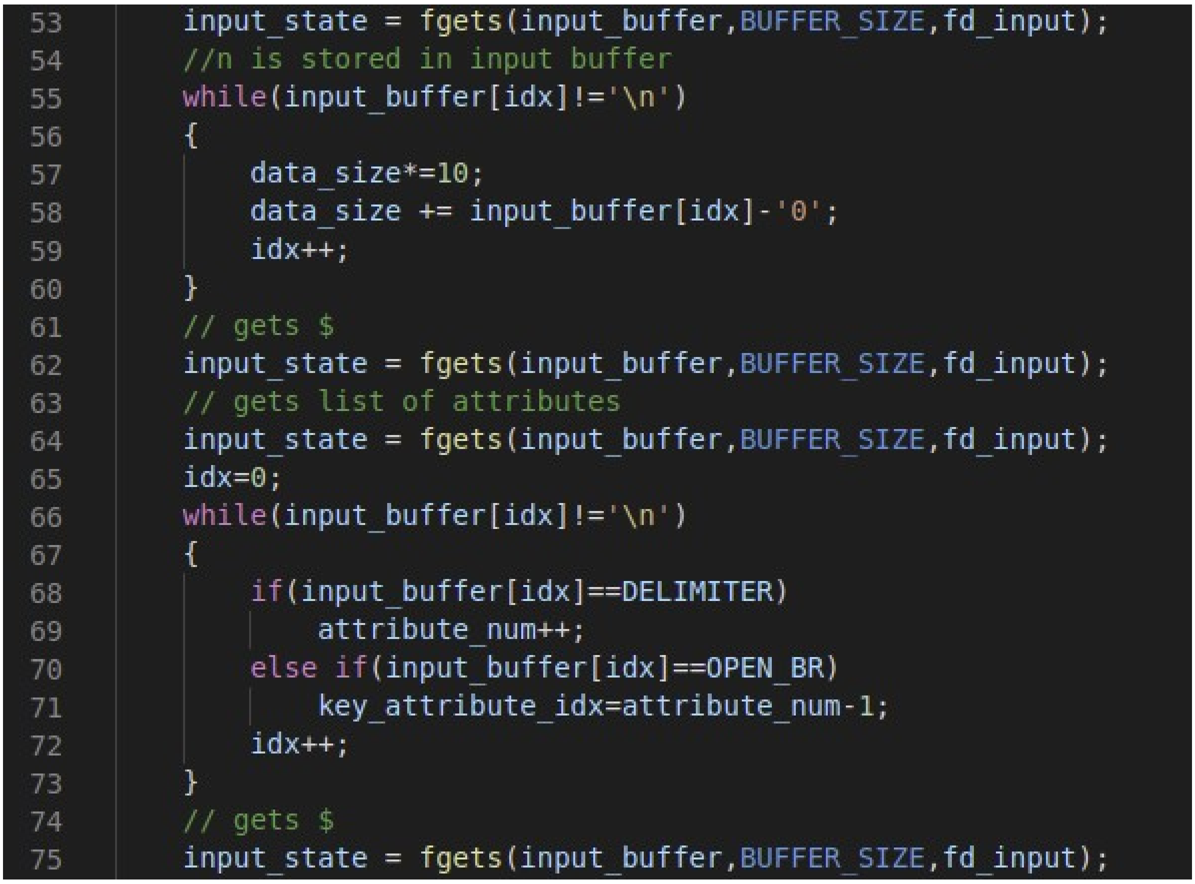
1. Get the whole input data from ‘hw1_input.txt’,
and map key attribute string to integer values between 0 and 63.

2. Sort only key attribute values using optimized radix sort.
(Optimized by adding two mapping lists)

3. Write ‘hw1_output.txt’ in order according to map_list.

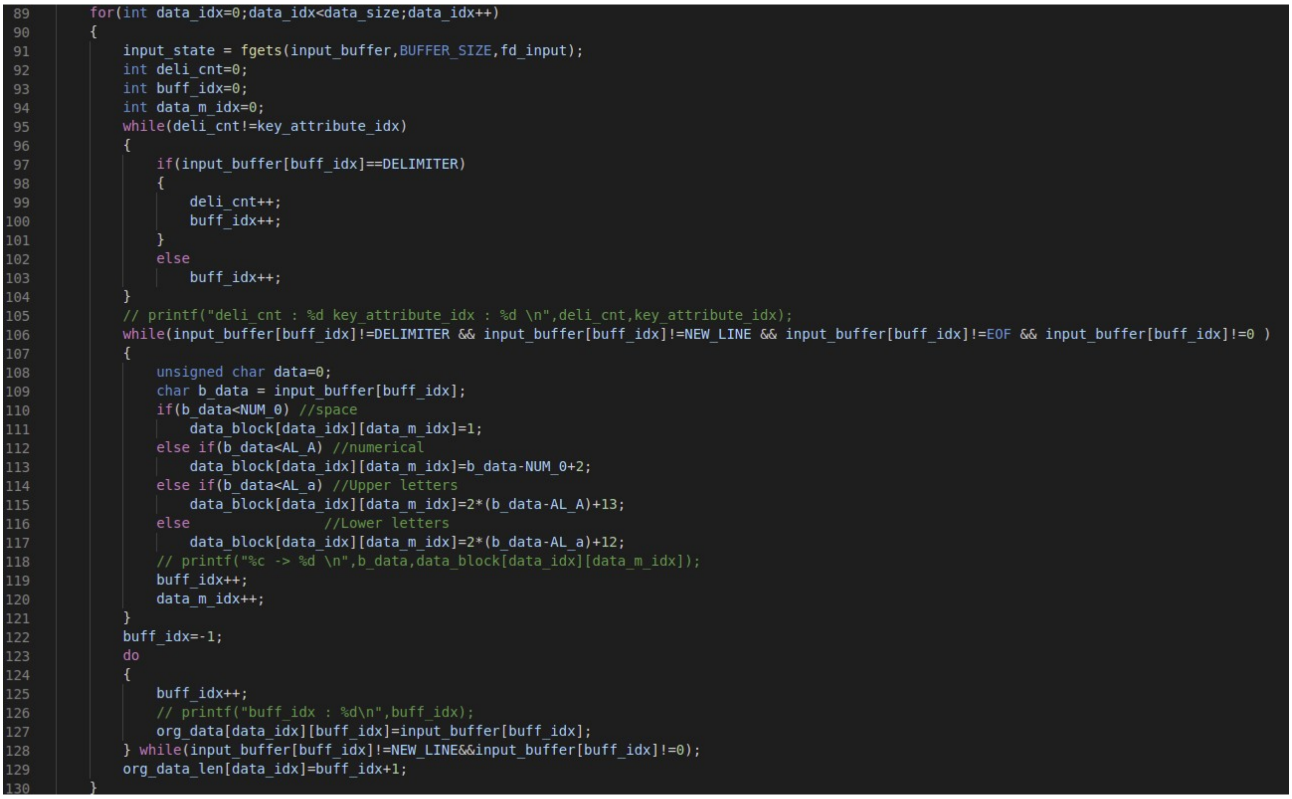
Below are detailed explanations and performance analysis of each part.

1. Get the whole input data from ‘hw1_input.txt’,
and map key attribute string to integer values between 0 and 63.



```
input_state = fgets(input_buffer,BUFFER_SIZE,fd_input);
//n is stored in input buffer
while(input_buffer[idx]!='\n')
{
    data_size*=10;
    data_size += input_buffer[idx]-'0';
    idx++;
}
// gets $
input_state = fgets(input_buffer,BUFFER_SIZE,fd_input);
// gets list of attributes
input_state = fgets(input_buffer,BUFFER_SIZE,fd_input);
idx=0;
while(input_buffer[idx]!='\n')
{
    if(input_buffer[idx]==DELIMITER)
        attribute_num++;
    else if(input_buffer[idx]==OPEN_BR)
        key_attribute_idx=attribute_num-1;
    idx++;
}
// gets $
input_state = fgets(input_buffer,BUFFER_SIZE,fd_input);
```

It gets the value of n and t, and gets what is the key attribute.

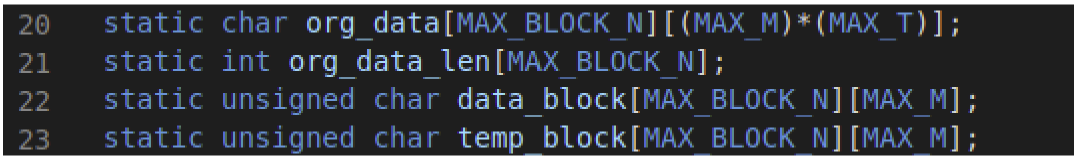


```c
for(int data_idx=0;data_idx<data_size;data_idx++)
{
    input_state = fgets(input_buffer,BUFFER_SIZE,fd_input);
    int deli_cnt=0;
    int buff_idx=0;
    int data_m_idx=0;
    while(deli_cnt!=key_attribute_idx)
    {
        if(input_buffer[buff_idx]==DELIMITER)
        {
            deli_cnt++;
            buff_idx++;
        }
        else
            buff_idx++;
    }
    // printf("deli_cnt : %d key_attribute_idx : %d \n",deli_cnt,key_attribute_idx);
    while(input_buffer[buff_idx]!=DELIMITER && input_buffer[buff_idx]!=NEW_LINE && input_buffer[buff_idx]!=EOF && input_buffer[buff_idx]!=0 )
    {
        unsigned char data=0;
        char b_data = input_buffer[buff_idx];
        if(b_data<NUM_0) //space
            data_block[data_idx][data_m_idx]=1;
        else if(b_data<AL_A) //numerical
            data_block[data_idx][data_m_idx]=b_data-NUM_0+2;
        else if(b_data<AL_a) //Upper letters
            data_block[data_idx][data_m_idx]=2*(b_data-AL_A)+13;
        else                //Lower letters
            data_block[data_idx][data_m_idx]=2*(b_data-AL_a)+12;
        // printf("%c -> %d \n",b_data,data_block[data_idx][data_m_idx]);
        buff_idx++;
        data_m_idx++;
    }
    buff_idx=-1;
    do
    {
        buff_idx++;
        // printf("buff_idx : %d\n",buff_idx);
        org_data[data_idx][buff_idx]=input_buffer[buff_idx];
    } while(input_buffer[buff_idx]!=NEW_LINE&&input_buffer[buff_idx]!=0);
    org_data_len[data_idx]=buff_idx+1;
}
```

And gets each input line. Here, it maps each key attribute character into a integer value as below.

Blank → 0 (Blank is not space. It means there is no character.)
Space → 1
'0' → 2
'1' → 3
.
.
.
'a' → 12
'A' → 13
.
.
.
'z' → 62
'Z' → 63

It stores these mapped integer values into 'data_block' and 'temp_block'.
And it stores the whole attributes to 'org_data'.



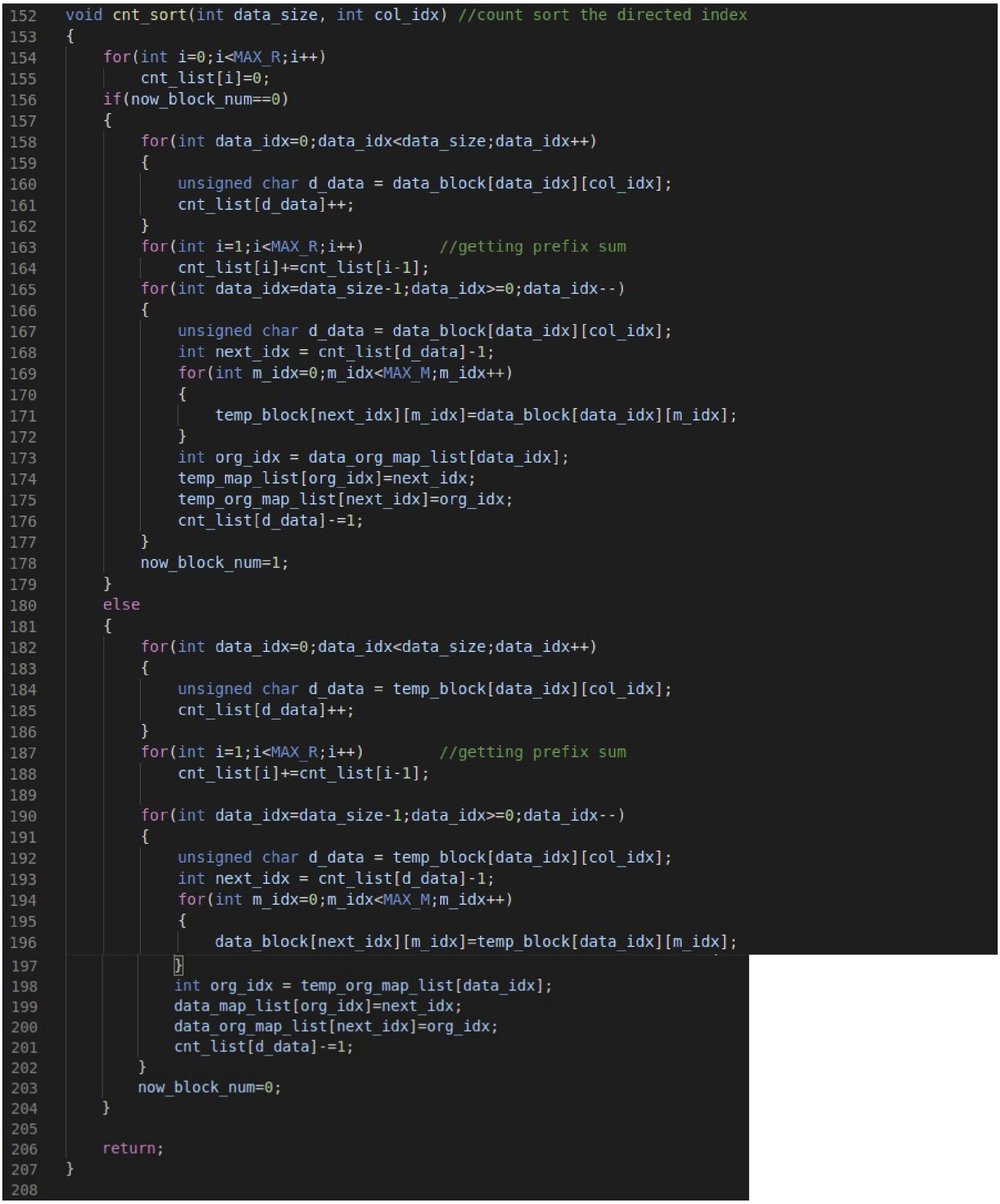
```c
static char org_data[MAX_BLOCK_N][(MAX_M)*(MAX_T)];
static int org_data_len[MAX_BLOCK_N];
static unsigned char data_block[MAX_BLOCK_N][MAX_M];
static unsigned char temp_block[MAX_BLOCK_N][MAX_M];
```

I declared global static arrays, 'data_block', 'temp_block', and 'org_data'.

This code block takes O(nmt) , where n is the number of objects, m is the maximum length of each attribute value, and t is the number of attributes for each object. Since each maximum values of n,m, and t are 1,000,000, 15, 10, the execution count is as similar as 150,000,000. (not sec, just representation of counts the code should run.) It does O(nmt) twice because it first maps into integer values and then store the whole attributes. I implemented like this for some optimization for storing whole attributes.

Anyway, **this code blocks costs O(nmt)**

2. Sort only key attribute values using optimized radix sort.
(Optimized by adding two mapping lists)

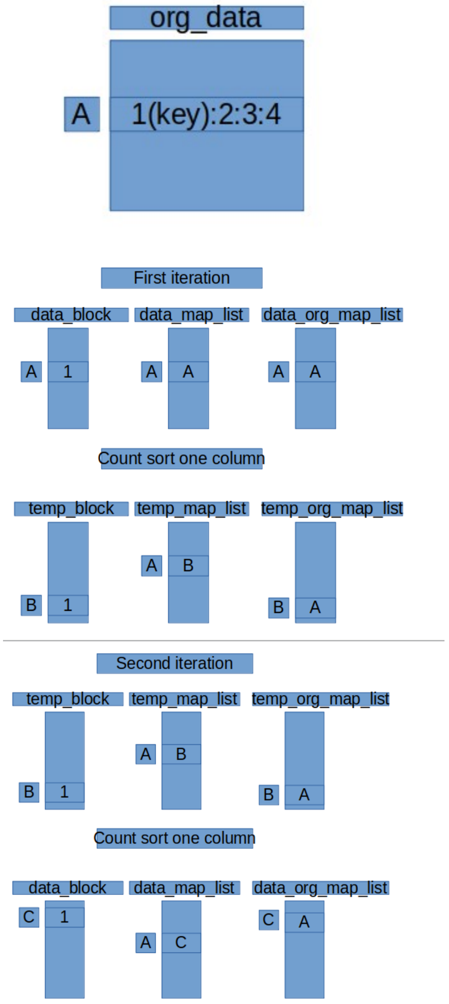


```
void cnt_sort(int data_size, int col_idx) //count sort the directed index
{
    for(int i=0;i<MAX_R;i++)
        cnt_list[i]=0;
    if(now_block_num==0)
    {
        for(int data_idx=0;data_idx<data_size;data_idx++)
        {
            unsigned char d_data = data_block[data_idx][col_idx];
            cnt_list[d_data]++;
        }
        for(int i=1;i<MAX_R;i++)        //getting prefix sum
            cnt_list[i]+=cnt_list[i-1];
        for(int data_idx=data_size-1;data_idx>=0;data_idx--)
        {
            unsigned char d_data = data_block[data_idx][col_idx];
            int next_idx = cnt_list[d_data]-1;
            for(int m_idx=0;m_idx<MAX_M;m_idx++)
            {
                temp_block[next_idx][m_idx]=data_block[data_idx][m_idx];
            }
            int org_idx = data_org_map_list[data_idx];
            temp_map_list[org_idx]=next_idx;
            temp_org_map_list[next_idx]=org_idx;
            cnt_list[d_data]-=1;
        }
        now_block_num=1;
    }
    else
    {
        for(int data_idx=0;data_idx<data_size;data_idx++)
        {
            unsigned char d_data = temp_block[data_idx][col_idx];
            cnt_list[d_data]++;
        }
        for(int i=1;i<MAX_R;i++)        //getting prefix sum
            cnt_list[i]+=cnt_list[i-1];

        for(int data_idx=data_size-1;data_idx>=0;data_idx--)
        {
            unsigned char d_data = temp_block[data_idx][col_idx];
            int next_idx = cnt_list[d_data]-1;
            for(int m_idx=0;m_idx<MAX_M;m_idx++)
            {
                data_block[next_idx][m_idx]=temp_block[data_idx][m_idx];
            }
            int org_idx = temp_org_map_list[data_idx];
            data_map_list[org_idx]=next_idx;
            data_org_map_list[next_idx]=org_idx;
            cnt_list[d_data]-=1;
        }
        now_block_num=0;
    }

    return;
}
```

I implemented count sorting function. The column index which should be sorted is given to the function. As I said above, there is only 64 integer values. So, it takes O(n+64) to count sort one column. As count sorting doesn't sort in place, I implemented one more data block called 'temp_block'. The code uses both blocks alternatively. And two mapping lists are given to each block. One is 'map_list' and another one is 'org_map_list'. 'map_list' records where the original data is in block. And, 'org_map_list' records which original data is stored in block. It is illustrated below.



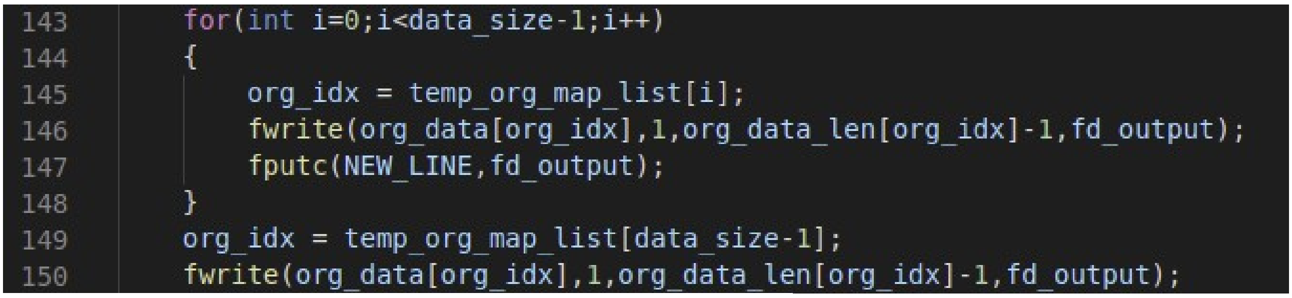
It iterates this for 15 times. So after 15 iterations, the sorted information is stored in 'temp_block', 'temp_map_list', and 'temp_org_map_list'.

This count sorting has some overhead. This one needs to manage two map_list, and it takes 2*n for one iteration. So, the time complexity for count sorting once is O(n+64) + O(2*n) = O(n).

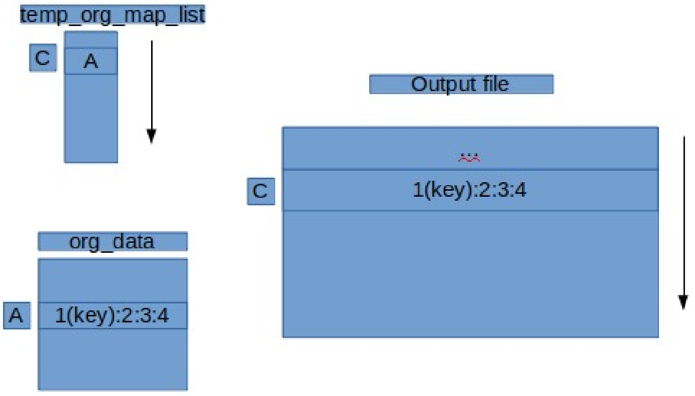
And this radix sort does count sorting for 15 times. So, it takes O(15*n) = O(n).

<u>**In conclusion, this sorting is done in O(n).**</u>

3. Write 'hw1_output.txt' in order according to map_list.



```
    for(int i=0;i<data_size-1;i++)
    {
        org_idx = temp_org_map_list[i];
        fwrite(org_data[org_idx],1,org_data_len[org_idx]-1,fd_output);
        fputc(NEW_LINE,fd_output);
    }
    org_idx = temp_org_map_list[data_size-1];
    fwrite(org_data[org_idx],1,org_data_len[org_idx]-1,fd_output);
```



In ‘temp_org_map_list’, the original object index is stored. So, just writing the proper original object in order is enough to make ‘hw1_output.txt’. It can decide to write which object in the following place. It just goes through the ‘temp_org_map_list’ from 0~(n-1) index, writing the corresponding original object to output file. **<u>This code block takes O(nmt).</u>**

In summary,
first part takes O(nmt), second part takes O(n), last part takes O(nmt).

So, the total complexity is O(nmt) + O(n) + O(nmt) = <u>O(nmt)</u>

Thank you for reading.