


# [CENG 315 All Sections] Algorithms

[Dashboard](#) / [My courses](#) / [571 - Computer Engineering](#) / [CENG 315 All Sections](#) / [October 24 - October 30](#) / [THE1](#)

 Description

 [Submission view](#)

## THE1

 **Available from:** Friday, October 28, 2022, 11:59 AM

 **Due date:** Saturday, October 29, 2022, 11:59 PM

 **Requested files:** the1.cpp, test.cpp ( [Download](#))

**Type of work:**  Individual work

Specifications:

- There is **1 task** to be solved in **36 hours** in this take home exam.
- You will implement your solutions in **the1.cpp** file.
- You are free to add other functions to **the1.cpp**
- Do **not** change the first line of **the1.cpp**, which is **#include "the1.h"**
- Do **not** change the arguments and return value of the functions **kWayMergeSortWithHeap()** in the file **the1.cpp**
- Do **not** include any other library or write include anywhere in your **the1.cpp** file (not even in comments).
- You are given a test.cpp file to **test** your work on **Odtuclass** or your **locale**. You can and you are encouraged to modify this file to add different test cases.
- If you want to **test** your work and see your outputs you can **compile** your work on your locale as:

```
>g++ test.cpp the1.cpp -Wall -std=c++11 -o test
> ./test
```

- You can test your **the1.cpp** on virtual lab environment. If you click **run**, your function will be compiled and executed with test.cpp. If you click **evaluate**, you will get a feedback for your current work and your work will be **temporarily** graded for **limited** number of inputs.
- The grade you see in lab is **not** your final grade, your code will be reevaluated with **different** inputs after the exam.

The system has the following limits:

- a maximum execution time of 1 minute (your functions should return in less than 1 seconds for the largest inputs)
- a 256 MB maximum memory limit
- a stack size of 64 MB for function calls (ie. recursive solutions)
- Each task has a complexity constraint explained in respective sections.
- Solutions with longer running times will not be graded.
- If you are sure that your solution works in the expected complexity constraints but your evaluation fails due to limits in the lab environment, the constant factors may be the problem.
- If your solution is correct, the time and memory limits may be adjusted to accept your solution after the lab. Please send an email if that is the case for you.

```
int kWayMergeSortWithHeap(int* arr, int K, int size, long& comparison,    long& swap);
```

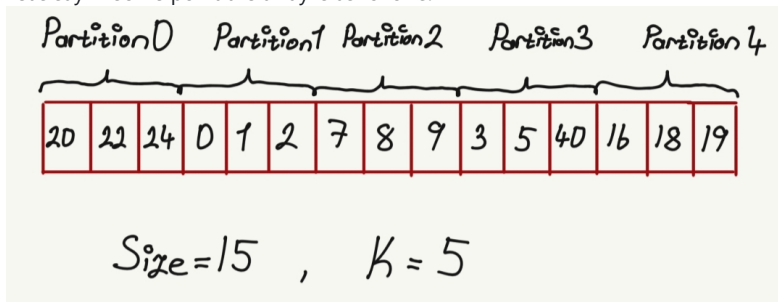
In this exam, you are asked to complete the function definitions to sort the given array **arr** with **ascending** order.

- **kWayMergeSortWithHeap()** should count the number of **comparison** and **swap** executed during sorting process (Comparisons are only between the values to be sorted during insertion sort and heapify process) and returns the total number of calls of **kWayMergeSortWithHeap()**.

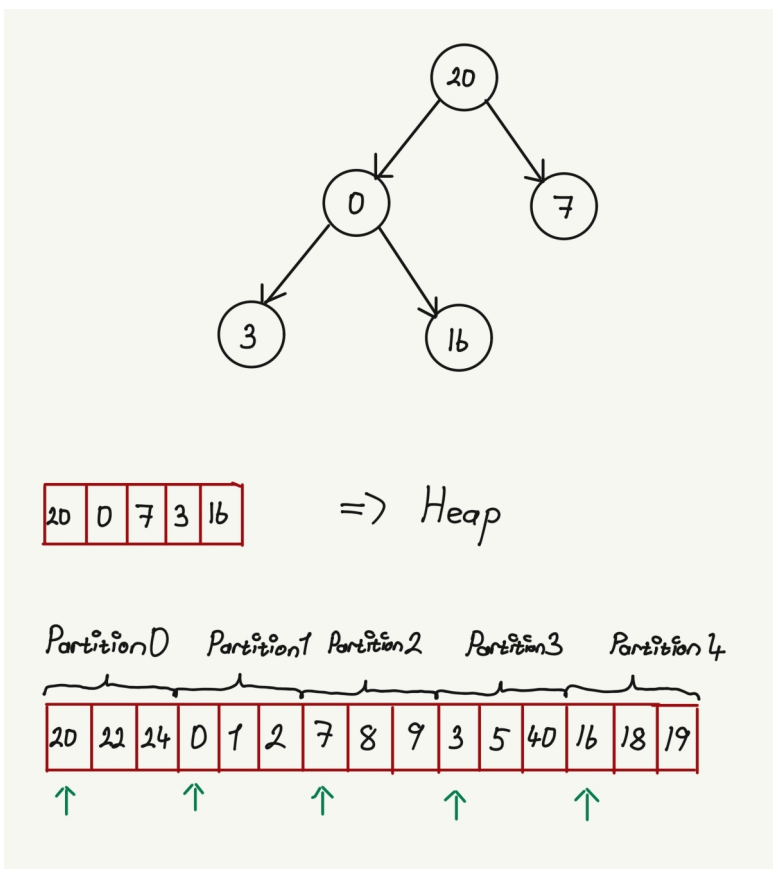
K Way Merge Sort With Heap algorithm (**kWayMergeSortWithHeap()**) is as follows:

- If the size of the array is less than K, then sort the array by using insertion sort.(You can use the insertion sort algorithm given to you in [THE0.](#))
- Otherwise, split the array into K sub-arrays and do K recursive calls to sort the partitions.
  - Then, merge K sorted arrays.

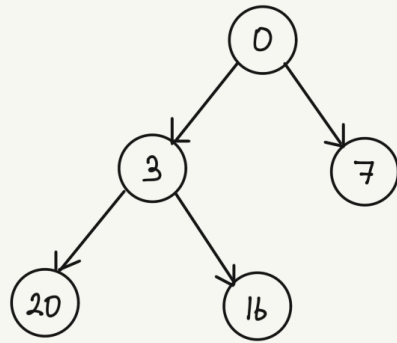
- When merging K sorted-arrays, you should use a Binary Min Heap to select the minimum element between the minimum elements of K partition arrays.
- When creating the array of the heap,
  - Firstly, generate a linear array whose elements are the minimum elements of the K partition arrays. At the beginning, the position of the each element is determined by the belonging partition. For example, the element coming from partition 0 is placed to heap\_array[0] and the element coming from partition 1 is placed to heap\_array[1] etc.
  - Then, heapify the initial array.
- After finding the minimum element, you should insert a new element from the related partition to the Min Heap.
  - Read the minimum element in the heap and record it.
  - Then, replace the minimum element with a new element from the partition that has the last minimum element.(New element insertion is not a swap operation. Swap has to be counted only inside the heap or insertion sort.)
  - Then, heapify the current array.
- In case of equality during heapify and insertion sort, do not swap the elements.
- Count the comparison and swap between any 2 elements of the array H in both insertion sort and heapify, such as  $H[i] > H[j]$
- Return the total number of **kWayMergeSortWithHeap()** calls.
- Let's have an example case:
  - Let's say in some point the array is as follows:



- Create a heap array and place the first elements of the partitions

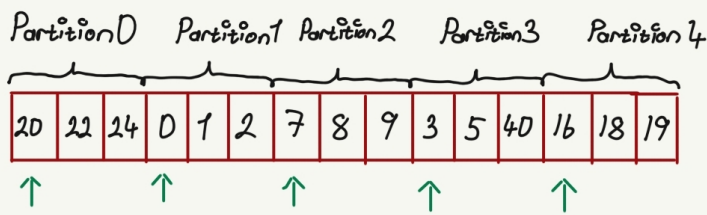


- Heapify the array(6 comparisons and 2 swaps are required.)

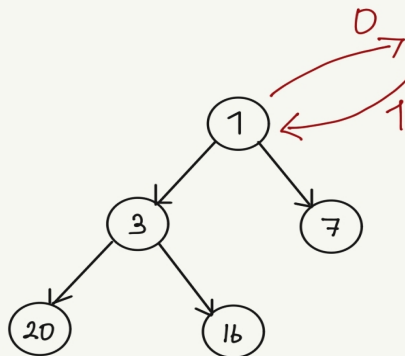


0	3	7	20	16
---	---	---	----	----

 $\Rightarrow$  Heap

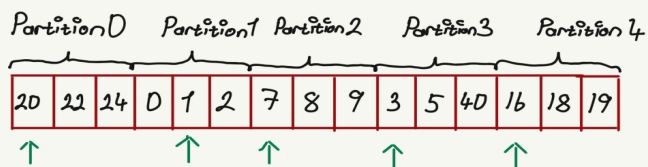


- Record the minimum and insert a new element (It is not counted as swap.)



1	3	7	20	16
---	---	---	----	----

 $\Rightarrow$  Heap



0														
---	--	--	--	--	--	--	--	--	--	--	--	--	--	--

 $\Rightarrow$  Sorted Array

- Then, heapify again.

#### Constraints:

- Maximum array size is  $2^{11}$ .
- You can make sure that size of the array is  $\beta K^{\text{depth}-1}$ , where  $\beta < K$  and depth is equal to recursion depth. That means, you can split the array into equal sized sub-arrays during recursive calls.

- Binary Min Heap should be implemented by using a linear array.
- $2 < K < 65$ .
- The maximum element inside the list is `INT_MAX-1` and all elements are integer. Therefore, you can insert `INT_MAX` to the heap as an empty location.

**Evaluation:**

- After your exam, black box evaluation will be carried out. You will get full points if you fill the **arr** variable as stated and return the number of comparisons, function calls and swaps correctly for the cases that will be tested.
- Because evaluation function checks the comparison and swap numbers, you will get zero point if you implement the merge function by using another way other than binary heap.

**Example IO:**

1)

Array size: 7, K: 7

Initial Array: {7, 6, 5, 4, 3, 2, 1}

Sorted Array: {1, 2, 3, 4, 5, 6, 7}

Number of comparison: 25

Number of swap: 14

Number of calls: 8

2)

Array size: 10, K: 15

Initial Array: {20, 45, 65, 78, 98, 65, 32, 74, 9, 1}

Sorted Array: {1, 9, 20, 32, 45, 65, 65, 74, 78, 98}

Number of comparison: 33

Number of swap: 26

Number of calls: 1

3)

Array size: 16, K: 4

Initial Array: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}

Sorted Array: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}

Number of comparison: 60

Number of swap: 20

Number of calls: 21

4)

Array size: 20, K: 5

Initial Array: {79, 63, 21, 78, 52, 63, 45, 10, 0, 1, 22, 100, 89, 66, 2, 63, 89, 98, 99, 785}

Sorted Array: {0, 1, 2, 10, 21, 22, 45, 52, 63, 63, 63, 66, 78, 79, 89, 89, 98, 99, 100, 785}

Number of comparison: 72

Number of swap: 32

Number of calls: 6

Requested files

the1.cpp

```
1  #include "the1.h"
2  #include <climits>
3
4
5  //You can add your own helper functions
6
7  int kWayMergeSortWithHeap(int* arr, int K, int size, long& comparison, long& swap){
8
9      int number_of_calls = 1;
10
11      //Your code here
12      return number_of_calls;
13  }
14
15
```

test.cpp

```

1 //This file is entirely for your test purposes.
2 //This will not be evaluated, you can change it and experiment with it as you want.
3 #include <iostream>
4 #include <fstream>
5 #include <random>
6 #include <ctime>
7 #include "the1.h"
8
9 // the1.h only contains declaration of the function:
10 // int kWayMergeSortWithHeap(int* arr, int K, int size, long& comparison, long& swap) ;
11
12 using namespace std;
13
14 void randomFill(int*& arr, int size, int minval, int interval){
15     arr = new int [size];
16     for (int i = 0; i < size; i++)
17     {
18         arr[i] = minval + (random() % interval);
19     }
20 }
21
22 void print_to_file(int* arr, int size){
23     ofstream ofile;
24     ofile.open("sorted.txt");
25     for(int i = 0; i < size; i++)
26         ofile << arr[i] << endl;
27 }
28
29 void read_from_file(int*& arr, int& K, int& size){
30
31     char addr[] = "input01.txt";
32     ifstream infile (addr);
33     if (!infile.is_open())
34     {
35         cout << "File \"<\"< addr
36             << "\" can not be opened. Make sure that this file exists.\" << endl;
37         return;
38     }
39     infile >> K;
40     infile >> size;
41     arr = new int [size];
42     for (int i=0; i<size;i++) {
43         infile >> arr[i];
44     }
45 }
46
47
48
49 void test(int* arr, int K, int array_size){
50     clock_t begin, end;
51     double duration;
52
53     //data generation and initialization- you may test with your own data
54     long comparison = 0;
55     long swap = 0;
56     int calls;
57
58
59
60
61
62     // Print initial array
63     cout << "Array size: " << array_size << ", K: " << K << endl << endl;
64     cout << "Initial Array: {"<";
65     for(int i=0; i<array_size; i++){
66         cout << arr[i];
67         if(i != array_size-1) cout << ", ";
68     }
69     cout << "}" << endl;
70
71     // Function call and and calculate the duration
72     if ((begin = clock() ) ==-1)
73         cerr << "clock error" << endl;
74
75     calls = kWayMergeSortWithHeap(arr, K, array_size, comparison, swap);
76
77     if ((end = clock() ) ==-1)
78         cerr << "clock error" << endl;
79
80
81     cout << "Sorted Array: {"<";
82     for(int i=0; i<array_size; i++){
83         cout << arr[i];
84         if(i != array_size-1) cout << ", ";
85     }
86     cout << "}" << endl << endl;
87
88     duration = ((double) end - begin) / CLOCKS_PER_SEC;
89     cout << "Duration: " << duration << " seconds." << endl;
90     cout << "Number of comparison: " << comparison << endl <<
91         "Number of swap: " << swap << endl <<
92         "Number of calls: " << calls << endl;
93     print_to_file(arr, array_size);
94     // Calculation and output end
95 }
96
97
98 int main(){
99     int size = 15;
100     int K = 5;
101     int minval = 0;
102     int interval = 100;
103     int *arr;
104     // Randomly generate initial array:
105     randomFill(arr, size, minval, interval);
106     // Read the test inputs. input01.txt through input04.txt exists.
107     // read_from_file(arr, K, size);
108     srand(time(0));
109     test(arr, K, size);
110     cout << endl;
111     return 0;

```

112 }  
113

[VPL](#)

You are logged in as anil eren gocer (Log out)

CENG 315 All Sections

ODTÜClass Archive

2021-2022 Summer

2021-2022 Spring

2021-2022 Fall

2020-2021 Summer

2020-2021 Spring

2020-2021 Fall

2019-2020 Summer

2019-2020 Spring

2019-2020 Fall

Class Archive

ODTÜClass 2021-2022 Summer School

Get the mobile app

