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, the1_solution.cpp (Download)
Maximum number of files: 2
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 **temporarly** 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)

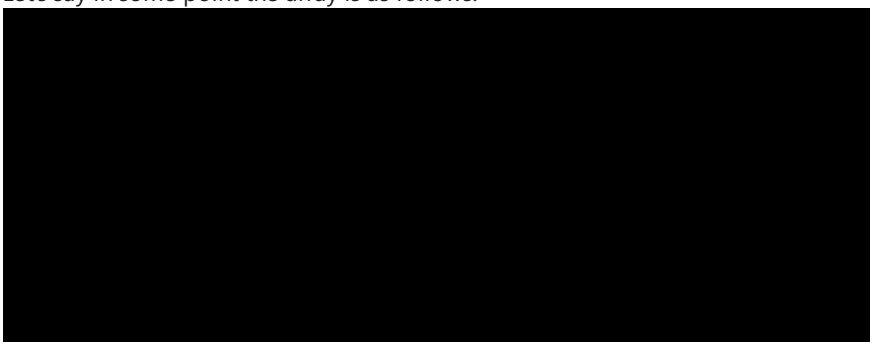
- Each task has a complexity constraint explained in respective sections.
- Solutions with longer running times will not be graded.

- If you are sure that your solution works in the expected complexity constrains but your evaluation fails due to limits in the lab environment, the constant factors may be the problem.
- If you 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.

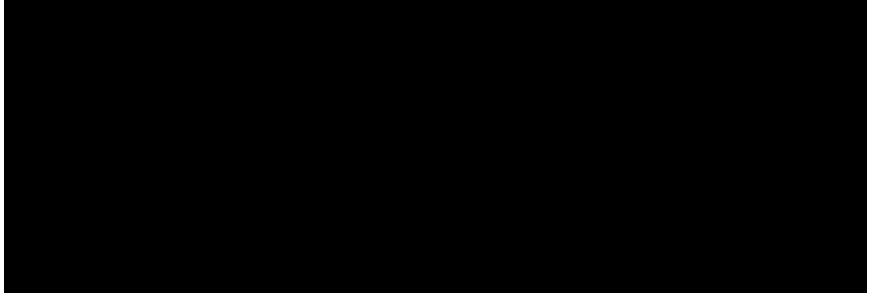
```
int kWayMergeSortWithHeap(int* arr, int K, int size, long& comparison,     long& swap);
```

        - 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.

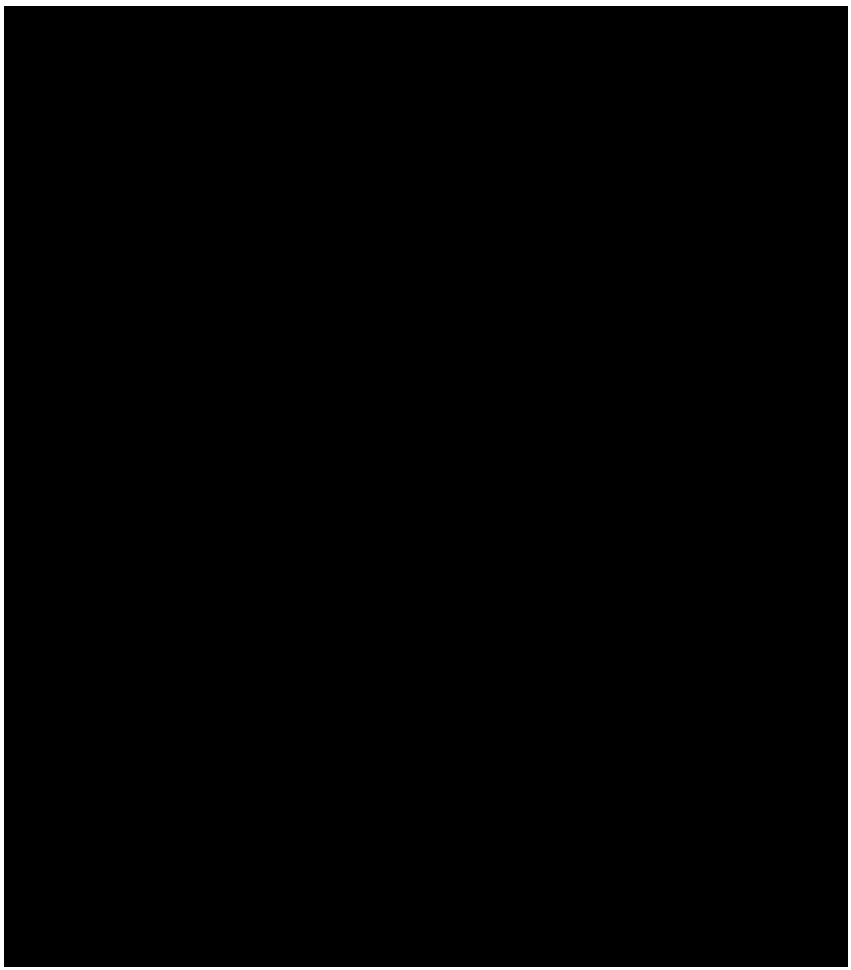
- In case of equality during heapify and insertion sort, do not swap the elements.
- Count the comparison and swap between any 2 elements of the array H in both insertion sort and heapify, such as H[i]>H[j]
- Return the total number of **kWayMergeSortWithHeap()** calls.
- Let's have an example case:
    - Let's say in some point the array is as follows:



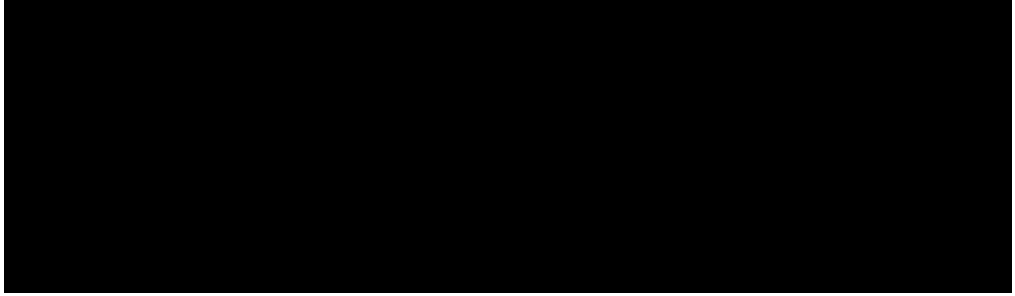
        - Create a heap array and place the first elements of the partitions



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



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



- Then, heapify again.

Constraints:

- Maximum array size is 2^11**.**
- You can make sure that size of the array is $\beta K^{depth-1}$, 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: 72
Number of swap: 32
Number of calls: 6

Requested files

the1.cpp

```cpp
#include "the1.h"
#include <climits>


//You can add your own helper functions

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

  int number_of_calls = 1;

    //Your code here
    return number_of_calls;
}


```

test.cpp

```cpp
//This file is entirely for your test purposes.
//This will not be evaluated, you can change it and experiment with it as you want.
#include <iostream>
#include <fstream>
#include <random>
#include <ctime>
#include "the1.h"

// the1.h only contains declaration of the function:
// int kWayMergeSortWithHeap(int* arr, int K, int size, long& comparison, long& swap) ;

using namespace std;

void randomFill(int*& arr, int size, int minval, int interval){
    arr = new int [size];
    for (int i = 0; i < size; i++)
    {
        arr[i] = minval + (random() % interval);
    }
}

void print_to_file(int* arr, int size){
    ofstream ofile;
    ofile.open("sorted.txt");
    for(int i = 0; i < size; i++)
        ofile << arr[i] << endl;
}

void read_from_file(int*& arr, int& K, int& size){
```