# Test task

1 – A 200Gb file filled with text lines divided by “\n”.

2 – Lexicographically sort lines of this file.

3 – Accept input/output file name (source\_file.txt, sorted\_file.txt)

+ work with small files, empty files, files much bigger than 200Gb

+ work well with non-ascii symbols

+ assume that zero-byte is not present in the file

+ assume that memory limit (8Gb of RAM) is much bigger than the size of the longest line

4 – Simple main.cpp solution

5 – Note

+ Use External Sort

+ Care about bottle necks from IO

+ Use multiple threads

+ Use efficient Sorting Algorithms (both time and memory consumption)

+ Use Sorting Algorithms from standard libraries

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Problem 1. Lexicographically sort text lines

## What does it mean by ''string lexicographically orders between two strings''?

<https://www.quora.com/What-does-it-mean-by-string-lexicographically-orders-between-two-strings>

Lexicographical order is similar to alphabetical order, so A comes before B comes before C.

There are sometimes unexpected results, for example case-sensitive ordering where "B" comes before "a" because the capital letters are ordered before lowercase ones (as in the ASCII and Unicode alphabets), and a special character like "é" could be ordered after all the normal letters and numbers.

We also sometimes get into trouble when we lexicographically order strings that contain numbers. For example, we naturally understand that "image2" should come before "image10", but "1" comes before "2" so many lexicographical ordering functions will put "image10" before "image2".

## Lib to compare 2 text lines:

std::lexicographical\_compare

<http://www.cplusplus.com/reference/algorithm/lexicographical_compare/>

work well with non-ascii symbols? => Need to test

+ compare non-ascii symbols:

Write a function my\_comp() for std::lexicographical\_compare

<https://stackoverflow.com/questions/39971585/sorting-a-vector-of-strings-by-the-first-letter-in-non-ascii-order-in-c>

## Sorting strings in C with qsort

<https://bewuethr.github.io/sorting-strings-in-c-with-qsort/>

string::c.str() returns a string of type const char \*

## Compare C qsort() and C sort()

<https://www.geeksforgeeks.org/c-qsort-vs-c-sort/>

When we tried to sort one million integers on C++14, Time taken by C qsort() was 0.247883 sec and time taken by C++ sort() was only 0.086125 sec

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Problem 2: Input / Output file

## Reading files line by line in C++ using ifstream: dealing correctly with badbit, failbit, eofbit, and perror()

https://gehrcke.de/2011/06/reading-files-in-c-using-ifstream-dealing-correctly-with-badbit-failbit-eofbit-and-perror/

## Handling Non-Ascii Chars in C++

<https://stackoverflow.com/questions/17648966/handling-non-ascii-chars-in-c>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Problem 3: External sort

## External Sorting for sorting large files in disk (Java)

http://exceptional-code.blogspot.com/2011/07/external-sorting-for-sorting-large.html

6. Open all the temp files (and set the read pointer to the beginning of the files).

7. Find the minimum number from the set of numbers currently pointed to by the file read pointer.

8. Write the number to disk. (To increase efficiency you could write the number to a buffer first and then flush the buffer out to disk when the buffer is full. But modern I/O libraries should be doing this anyway for you).

9. Read another number from the file that contained the minimum number at step 7.

10. Repeat step 7 to 9 until all numbers from all the temp files have been processed, merged, and written out to disk.

=> Taking a large amount of time

## Sorting big files using k-way merge sort

http://www.sinbadsoft.com/blog/sorting-big-files-using-k-way-merge-sort/

## How to use the priority queue STL for objects?

<https://stackoverflow.com/questions/19535644/how-to-use-the-priority-queue-stl-for-objects>

## Priority queue with Pointers and Comparator C++

<https://stackoverflow.com/questions/23997104/priority-queue-with-pointers-and-comparator-c>

**Class CmpEdgePtrs**

struct CmpEdgePtrs

{

bool operator()(const Edge\* lhs, const Edge\* rhs) const

{

return lhs->getWeight() < rhs->getWeight();

}

};

**Class Graph**

class Graph

{

// rest of class stuff

private:

priority\_queue<Edge\*, vector<Edge\*>, CmpEdgePtrs> edges;

};

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Problem 4: Multiple sorting threads

<https://www.justsoftwaresolutions.co.uk/threading/multithreading-in-c++0x-part-1-starting-threads.html>