# Assignment overview

This lab has two parts:

1. Answers to questions in Part 1 below, written here in this file. *This part is due by the end of lab today – we’ll be checking upload time to Learning Hub!*
2. A working Java program for Part 2; submit your Java code. (Due date as noted on Learning Hub)

This lab requires Java programming. You may (and should!) discuss the lab and coding techniques with your classmates, but all of the code you submit to Learning Hub must be your own.

# Submission information

What to submit:

* Part 1: This Word file with your answers filled in *(DUE BY THE END OF LAB TODAY)*
* Part 2: Java source file

Please do not zip or compress your submissions.

# Grading

This lab is graded on a 10-point scale. Point breakdown:

* Part 1: A: 2pts, B: 2pts, C: 1pt
* Part 2: 5pts

# Assignment details

### Part 1

Problem: The Trans-Canada Highway follows a nearly straight line as it crosses the Canadian prairies. Gas stations (N of them in total) occur at various intervals along the highway. Your task is to find the distance between two closest stations. (The distance between two stations x and y is computed as |x − y|.)

Input is an array of size N containing the mile marker locations of the gas stations, *unsorted*. For example:

480 231 0 477 121 ... 2000 1176 501

Output for this example (based on the input we can see) would be |480-477| = 3.

1. Design and write pseudocode for a pre-sorting-based transform-and-conquer algorithm that solves this problem. [2 points]

ALGORITHM PresortMode(A[0..n-1]) {

sort the array A

diff <- MAX\_VALUE //Initialize difference as infinite

i <- 0

while i < n-1 do

if (|A[i+1] - A[i]| < diff)

diff <- |A[i+1] - A[i]|

i <- i+1

return diff

}

1. Design and write pseudocode for a brute-force algorithm that solves this problem. [2 points]

ALGORITHM BruteForceMode(A[0..n-1]) {

diff <- MAX\_VALUE //Initialize difference as infinite

for i <- 0 to n-2 do

for j <- i+1 to n-1 do

if (|A[i] - A[j]| < diff)

diff <- |A[i] - A[j]|

return diff

}

1. State and compare the efficiency of your algorithms for Parts A and B. [1 point]

A) Efficiency: O(nlogn) + O(n) = O(nlogn)

B) Efficiency: O()

Therefore, Part A is more efficient in terms of Big O class efficiency.

### Part 2

Design and implement an algorithm that finds the smallest K numbers (in value) out of N numbers. For example, given an array {4, 5, 1, 6, 2, 7, 3, 8} and the number K=4, return the smallest 4 numbers: 1, 2, 3, and 4.

Note: An algorithm that sorts the array and outputs the first K numbers is *not the best answer* and will receive only partial marks. The complexity of such an algorithm would be at least O(nlogn) due to the sort. For full marks you should design a more efficient algorithm that uses a Heap.

**Hints:**

* Do not write your own implementation of a Heap or Priority Queue (PQ). Use the Java PriorityQueue class, which features a heap implementation.
* Note that with this class you can create a PQ of a specific maximum size.
* Methods of interest to you are:

offer() – adds an element to the PQ (returns true)

peek() – inspect the top (maximum) element on the heap, without removing it (returns the data type that is specified in your implementation of the PQ)

poll() – returns the top (maximum) element on the heap, *and removes it from the heap* (returns the data type that is specified in your implementation of the PQ)

size() – returns the number of elements currently in the PQ

* The file “priority queue example.zip” on Learning Hub contains some sample Java code that shows how to implement this class, and how to use the above methods.
* Official docs for the Java PriorityQueue class:
  + https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html