Due date:

This file contains Lab 6. You must submit your answers to the D2L Dropbox "Lab-6"

1. Answer for question 1 should be written on this file, due at the **end of lab today.**

2. A working java program and print screen of output for question 2 by the **end of today (11:30 pm).**

Lab 6 requires Java programming. You can work in pairs (but you must still submit your own work to D2L).

**Note that late assignments will not be graded. I will not accept any labs by email.**

Please do not zip or compress your submissions. D2L allows you to upload multiple files

1- The Trans-Canada Highway (TCH) follows a completely straight line as it crosses the Canadian prairies. Gas stations occur at various intervals. Assuming that the first station is a mile 0, and the last station is at mile n, find the distance between two closest stations. (The distance between two stations x and y is computed as |x − y|.)

Input is the mile maker locations of the gas stations, for example:

480 231 0 477 121 ... 1176 501 2000 (for n=2000)

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

A). Design and write pseudocode for a presorting-based transform and-conquer algorithm that solve this problem. [2 mark]

1. Algorithm PresortElementMinDistances (A[0..n−1])

2. //Solve the element MinDistances by sorting with merge sort algorithm first

3. //Input: An array A[0…n-1]

4. //Output: Returns the shortest distance between two miles.

5. sort the array A

6. minDistance ← a[i+1] – a[i] //min distance between two miles so far

7. for i ← 1 to n-1 do

8. if a[i]=a[i+1]

9. return 0

10. else if a[i+1] – a[i] < minDistance

11. minDistance ← a[i+1] – a[i]

12. return minDistance

B). Design and write pseudocode for a brute-force algorithm that solve this problem. [2 mark]

1. Algorithm BruteForceElementMinDistances (A[0..n−1])

2. //Solve the element MinDistances by comparing each element and record the distance

3. //Input: An array A[0…n-1]

4. //Output: Returns the shortest distance between two miles.

5. minDistance ← 0 //min distance between two miles so far

6. for i ← 0 to n-1 do

7. minDistanceTemp ← 0

8. for j ← i+1 to n-1 do

9. if |a[j] – a [i]| < minDistanceTemp

10. minDistanceTemp ← |a[j] – a [i]|

11. if minDistanceTemp < minDistance

12. minDistance ← minDistanceTemp

13. return minDistance

if (A[j] - A[i] < min AND A[j] - A[i] > 0) Then  
 index\_first <- j  
 index\_second <- i  
 min <- A[index\_first] - A[index\_second]

C). Compare the efficiency of your algorithm for part A with your algorithm for part B. [1 mark]

Part A have two steps (Merge Sort and PresortElementMinDistances)

The efficiency for Merge Sort is T(n) = 2 T(n/2) + n -> T(n) = O(nlogn)

The efficiency for the PresortElementMinDistances is O(nlogn)

The overall efficiency is O(nlogn)

Part B scan the list and compute the difference of all distinct values and find the min distance between two miles. The efficiency is O(n2)

2- Design and implement an algorithm that finds the smallest k numbers (in value) out of n numbers. For example, if given an array with eight numbers {4, 5, 1, 6, 2, 7, 3, 8}, return the least four numbers 1, 2, 3, and 4.

The algorithm that sort the n input numbers increasingly and returns the first K number is not acceptable for the answer. Since it needs to sort, its time complexity is. You should design more efficient algorithm using Max-heap. [5 mark]



Hint.

You should create a max-heap with capacity k that will contain the least k numbers out of n input numbers.

In Java, the PriorityQueue class is implemented as a priority heap. You can find a sample code in D2l that shows the basic operations of PriorityQueue such as offer(), peek(), poll(), and size().