## Concurrent & Parallel Programming

**Question 1**

A program is required to find the smallest value together with the frequency of its occurrence in a huge integer array. This program will be executed on a multi-core machine and the program must maximize the use of this resource. Using a Threadpool and the Callable interface write a solution to this problem.

(Hint: Create a class to hold the result for the Callable interface.)

**Question 2**

*MergeSort* continuously divides the data into segments until segments of size 1 are reached. It then begins the merging phase. This is the expensive part. Improvements could be made if we could reduce the cost of merging. It turns out that *InsertionSort* is very efficient for small data sequences (say sequences of 100 values) where the data is partially ordered in the correct order and the displacement is small.

The idea is to combine *MergeSort* and *InsertionSort* to reduce the over head of merging. To do this we terminate the *MergeSort* division when segments of some given size are reached, use *InsertionSort* to sort the segments and then do the merging as before.

Your task is to implement this solution using the ForkJoin framework. You should test it with integer 10000000 integer array.

static void mergeSort(int f[], int lb, int ub){

//termination reached when a segment of size 1 reached - lb+1 = ub

if(lb+1 < ub){

int mid = (lb+ub)/2;

mergeSort(f,lb,mid);

mergeSort(f,mid,ub);

merge(f,lb,mid,ub);

}

}

static void merge(int f[], int p, int q, int r){

//p<=q<=r

int i = p; int j = q;

//use temp array to store merged sub-sequence

int temp[] = new int[r-p]; int t = 0;

while(i < q && j < r){

if(f[i] <= f[j]){

temp[t]=f[i];i++;t++;

}

else{

temp[t] = f[j]; j++; t++;

}

}

//tag on remaining sequence

while(i < q){ temp[t]=f[i];i++;t++;}

while(j < r){ temp[t] = f[j]; j++; t++;}

//copy temp back to f

i = p; t = 0;

while(t < temp.length){ f[i] = temp[t]; i++; t++;}

}

}