# Quin’darius Ali Lyles MINUS Woods

# Soloman Walker

# November 13, 2018

Pseudocode Assignment

Module 7A

In this Pseudocoding Assignment the applications will:

* Uses a bubble sort to sort an array of 10000 integers.
* Uses parallelism to sort the same array 10000 integers.
* Display the time it takes for each application to run.

Testing pseudocode

CLASS testSortingMethods{  
METHOD main (String ARRAY args){  
final int SIZE EQUALS 1000   
int ARRAY list1 EQUALS NEW int[SIZE]   
int ARRAY list2 EQUALS NEW int[SIZE]   
//Filling in Array with random values

FOR EACH TIME (int i EQUALS 0 i < LIST1 LENGTH i PLUS ONE)  
 list1[i] EQUALS list2[i] EQUALS (int)(Math.random() MULTIPLIED BY 10000)   
long startTime EQUALS SYSTEM TIME ()  
ParallelMergeSort.parallelMergeSort(list1)   
long endTime EQUALS SYSTEM TIME ()  
PRINT("\n The parallel merge sort took " PLUS (endTime MINUS startTime) PLUS " milliseconds”)

long startTime2 EQUALS SYSTEM TIME ()  
BubbleSort.bubbleSort(list2)  
long endTime2 EQUALS SYSTEM TIME ()  
PRINT("\n The bubble sort sort took " PLUS (endTime2 MINUS startTime2) PLUS " milliseconds”)  
}  
}

Parallel Merge Sort pseudocode

CLASS ParallelMergeSort {  
//Method to pass list into Parallel Merge Sort  
METHOD parallelMergeSort (int ARRAY list) {  
RECURSIVEACTION mainTask EQUALS NEW SortTask(list)   
FOR EACH TIME JoinPool pool EQUALS NEW FOR EACH TIME JoinPool()

pool. SUMMON (mainTask)   
1}  
private static class SortTask extends RECURSIVEACTION {  
private final int THRESHOLD EQUALS 10   
private int ARRAY list   
SortTask(int ARRAY list){  
this.list EQUALS list   
}  
 OVERRIDE!   
COMPUTE(){  
IF ( LIST LENGTH < THRESHOLD)  
java.util.Arrays.sort(list)   
else {  
//Obtain the first half   
int ARRAY firstHalf EQUALS NEW int [ LIST LENGTH /2]   
COPY ARRAY (list, 0, firstHalf, 0, LIST LENGTH /2)   
// Obtain the second half   
47   
int secondHalfLength EQUALS LIST LENGTH MINUS LIST LENGTH /2   
int ARRAY secondHalf EQUALS NEW int [secondHalfLength]   
 COPY ARRAY (list,,LIST LENGTH /2, ,secondHalf, 0 ,secondHalfLength)   
// Recursively sort the two Halves  
invokeAll( NEW SortTask(firstHalf), NEW SortTask(secondHalf))   
MergeSort.merge(firstHalf, ,secondHalf,, list)   
}  
}  
}  
}

Merge Sort pseudocode

CLASS MergeSort {  
//The method FOR EACH TIME sorting the numbers  
METHOD mergeSort (int ARRAY list ) {  
IF ( LIST LENGTH > 1){  
// Merge sort the first half  
int ARRAY firstHalf EQUALS NEW int [ LIST LENGTH /2]   
COPY ARRAY (list, 0, firstHalf, 0, LIST LENGTH /2)   
mergeSort(firstHalf)   
// Merge sort the second half   
int secondHalfLength EQUALS LIST LENGTH MINUS LIST LENGTH /2   
int ARRAY secondHalf EQUALS NEW int [secondHalfLength]   
COPY ARRAY (list, LIST LENGTH /2, secondHalf, 0, mergeSort(secondHalf)   
merge(firstHalf, secondHalf, list)   
}  
}  
METHOD merge (int ARRAY list1, int ARRAY list2, int ARRAY temp){  
int current1 EQUALS 0 //current index in list 1  
int current2 EQUALS 0 //current index in list 2  
int current3 EQUALS 0 //current index in temp  
WHEN THIS IS TRUE (current1 < LIST1 LENGTH AND current2 < LIST2 LENGTH ){  
41   
IF (list1[current1]< list2[current2])  
temp[current3 PLUS ONE ] EQUALS list1[current1 PLUS ONE ]   
else   
temp[current3 PLUS ONE ] EQUALS list2[current2 PLUS ONE ]   
}  
WHEN THIS IS TRUE (current1 < LIST1 LENGTH ){  
temp [current3 PLUS ONE ] EQUALS list1[current1 PLUS ONE ]   
}  
WHEN THIS IS TRUE (current2 < LIST2 LENGTH )   
temp[current3 PLUS ONE ] EQUALS list2[current2 PLUS ONE ]   
}  
}

Bubble Sort pseudocode

CLASS BubbleSort{  
//Bubble sort method 3   
METHOD bubbleSort(int ARRAY list){  
BOOLEAN needNextPass EQUALS true   
FOR EACH TIME (int arrayPostion EQUALS 1   
arrayPostion < LIST LENGTH AND needNextPass   
arrayPostion PLUS ONE ){  
// Array may be sorted and next pass not needed  
needNextPass EQUALS false   
FOR EACH TIME (int i EQUALS 0 i < LIST LENGTH MINUS arrayPostion i PLUS ONE ){  
IF (list[i] > list[i PLUS 1]){  
//Swap list[i] with list[i PLUS 1]  
int temp EQUALS list[i]   
list[i] EQUALS list[i PLUS 1]   
list[i PLUS 1] EQUALS temp   
needNextPass EQUALS true

//Next Pass is still needed   
}  
}  
}  
}  
}