Computer System Engineering
50.005
Prof. David Yau
Dr. Jit Biswas

Week 3: Lab 2 (40 marks)

Lab2: Multi-Threads

Contact us
chuadhry@mymail.sutd.edu.sg
eyasu_chekole@mymail.sutd.edu.sg
shuailong_liang@mymail.sutd.edu.sg
Hao_zhang@mymail.sutd.edu.sg

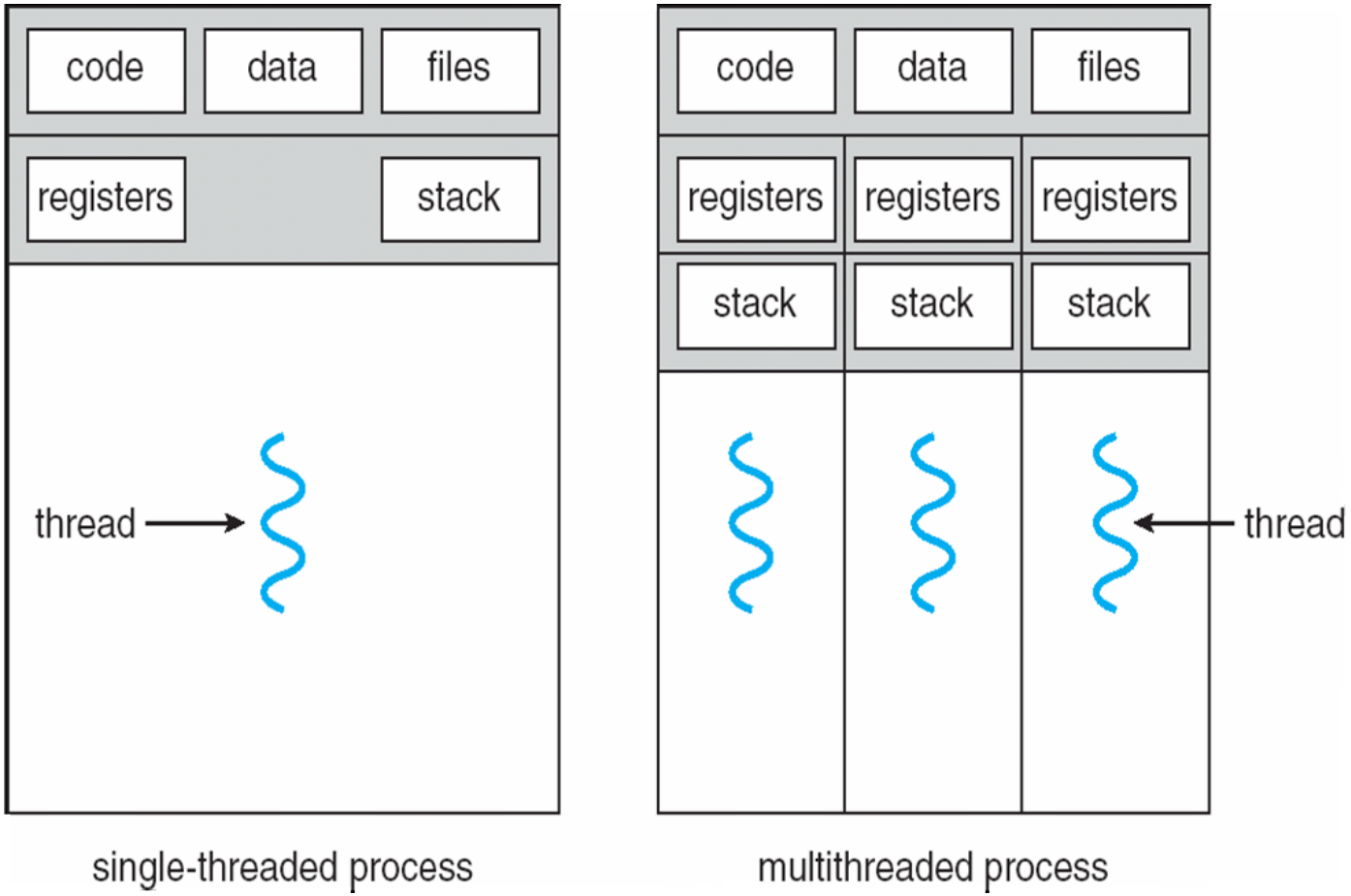
Objectives

- Understanding the Thread concept
- Implementing threads in Java/C.
- Analyzing performance enhancements in multi-threaded processes.

Threads

- We have seen process in Lab 1.
  - A process is an execution (a running instance) of a program.
- Today, we will see threads
  - A thread is a single independent locus of control within the scope of a process.
  - It is a lightweight sub-process, and the smallest unit of execution.

Single and Multithreaded Processes



Java Threads

- Java threads are managed by the JVM
- Java threads may be created by:
  - Implementing the Runnable interface

```
public interface Runnable
{
    public abstract void run();
}
```

Java Threads - Example Program

NB.
i.Runnable is an abstract interface that must be implemented
ii.An object that implements **Runnable** can run as a separate thread
iii.The thread's execution starts at **run()** method

```
class MutableInteger
{
  private int value;
  public int getValue() {
   return value;
  }
  public void setValue(int value) {
   this.value = value;
  }
}

class Summation implements Runnable
{
  private int upper;
  private MutableInteger sumValue;
  public Summation(int upper, MutableInteger sumValue) {
   this.upper = upper;
   this.sumValue = sumValue;
  }
  public void run() {
   int sum = 0;
   for (int i = 0; i <= upper; i++)
      sum += i;
   sumValue.setValue(sum);
  }
}
```

Java Threads - Example Program

NB:
i.Summation, which implements **Runnable**, is passed as argument to **Thread** constructor as object to run in new thread **thrd**
ii.Calling thread uses **start**(), **join**() methods to respectively start **thrd** and wait for it to finish

```
public class Driver
{
  public static void main(String[] args) {
   if (args.length > 0) {
    if (Integer.parseInt(args[0]) < 0)
     System.err.println(args[0] + " must be >= 0.");
    else {
     // create the object to be shared
     MutableInteger sum = new MutableInteger();
     int upper = Integer.parseInt(args[0]);
     Thread thrd = new Thread(new Summation(upper, sum));
     thrd.start();
     try {
        thrd.join();
        System.out.println
                ("The sum of "+upper+" is "+sum.getValue());
     } catch (InterruptedException ie) { }
    }
   }
   else
    System.err.println("Usage: Summation <integer value>");
   }
}
```

Pthreads (in C)

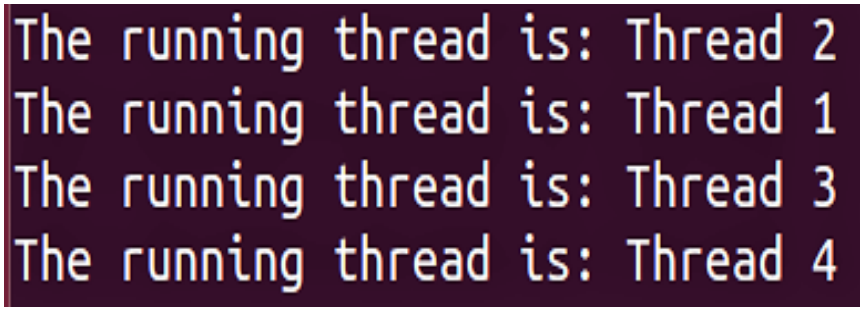
- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- API specifies behavior of the thread library, implementation is up to developers of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS/X)

Implementing multi-threading

```
public class MultiThreadExample {
    public static void main(String[] args) {
        //Creating four threads from the MultiThread class
        MultiThread thread1 = new MultiThread();
        thread1.setName("Thread 1");
        MultiThread thread2 = new MultiThread();
        thread2.setName("Thread 2");
        MultiThread thread3 = new MultiThread();
        thread3.setName("Thread 3");
        MultiThread thread4 = new MultiThread();
        thread4.setName("Thread 4");

        //Starting the threads (calling the run() method)
        thread1.start();
        thread2.start();
        thread3.start();
        try {
            //Wait until thread3 terminates
            thread3.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        thread4.start();
    }
}
class MultiThread extends Thread{
    @Override
    public void run() {
        System.out.println("The running thread is: "+Thread.currentThread().getName())
    }
}
```

Implementing multi-threading



Implementing multi-threading

```
public class MultiThreadExample {
       public static void main(String[] args) {
          MultiThread[] threadsList = {
             new MultiThread("Thread 1"),
             new MultiThread("Thread 2"),
             new MultiThread("Thread 3"),
             new MultiThread("Thread 4")
          };
          for (MultiThread thread : threadsList) {
             thread.start();
          }
       }
    }

class MultiThread extends Thread{
    private String name;
    //Constructor
    public MultiThread(String name) {
        this.name = name;
    }

    @Override
    public void run() {
        for (int i = 1; i <= 5; ++i) {
            System.out.println(name + ": " + i);
        }
    }
}
```

```
Thread 1: 1
Thread 1: 2
Thread 3: 1
Thread 2: 1
Thread 3: 2
Thread 3: 3
Thread 1: 3
Thread 4: 1
Thread 1: 4
Thread 3: 4
Thread 2: 2
Thread 3: 5
Thread 1: 5
Thread 4: 2
Thread 4: 3
Thread 2: 3
Thread 2: 4
Thread 2: 5
Thread 4: 4
Thread 4: 5
```

Java recommended functions

Function	Used for	Class it belongs to
.start()	Starting a thread by calling its run() method	Thread
.join()	Waiting for a thread to terminate	Thread
run()	An entry point for the thread	Thread
.subList	Returning a specific portion of a list	ArrayList<Integer>

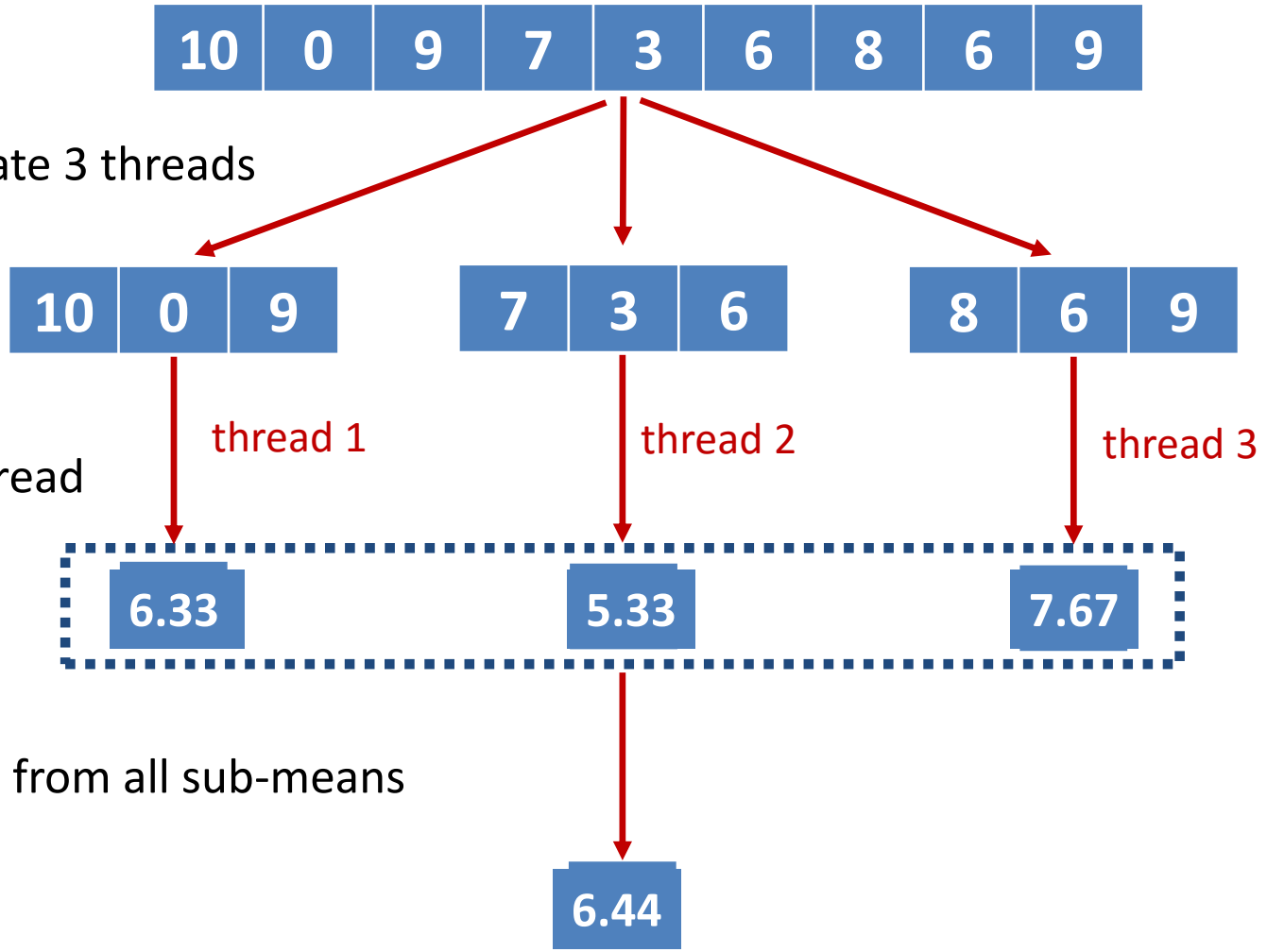
Lab2 Tasks

1. Find mean value of an array using multi-thread (15 marks)

2. Find median of an array using multi-thread (25 marks)

Task 1: Finding mean

1. Split into 3 parts, create 3 threads
2. Find mean in each thread
3. Find final mean value from all sub-means



Task 1: Finding mean

- For Task 1, you are expected to do the following:

1. Read data from “input.txt” and store it in an array (array length=524288).

2. Partition the array into N parts, and run N threads for finding mean value.

3. Print out intermediate values, e.g., print out N mean values.

4. Compute and print out global mean value from the N mean values obtained on step 3 (should be 5001.017… //I will check and update it).

5. Do the experiment for different number of threads and record its corresponding execution time (see the following table), and plot it.

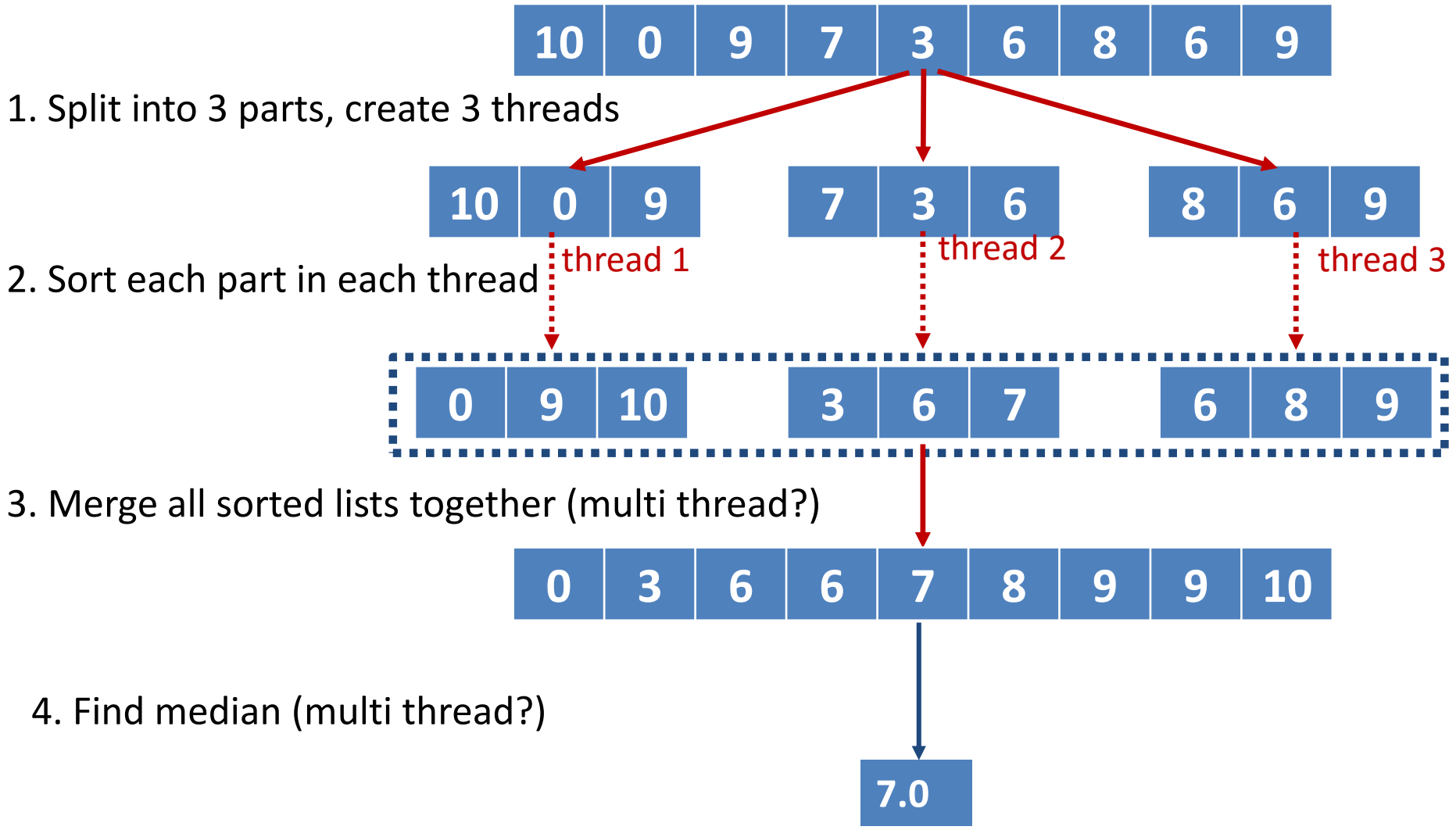
Number of threads	1	2	4	8	16	32	64	128	256	1024	2048
Execution time (ms)											

Task 2: Finding median

- Computing median is similar to computing mean, but it involves sorting and merging of the array.

- Hence, it is a bit complex and takes more execution time.

Task 2: Finding median



Sort & Merge

- Sort algorithm: bubble sorting as example

```
procedure bubbleSort( A : list of sortable items )
    n = length(A)
    repeat
       swapped = false
       for i = 1 to n-1 inclusive do
          if A[i-1] > A[i] then
             swap(A[i-1], A[i])
             swapped = true
          end if
       end for
       n = n - 1
    until not swapped
end procedure
```

You can choose any sorting algorithm you want.

code from: https://en.wikipedia.org/wiki/Bubble_sort

Sort & Merge

- Merge algorithm: merge N sorted arrays

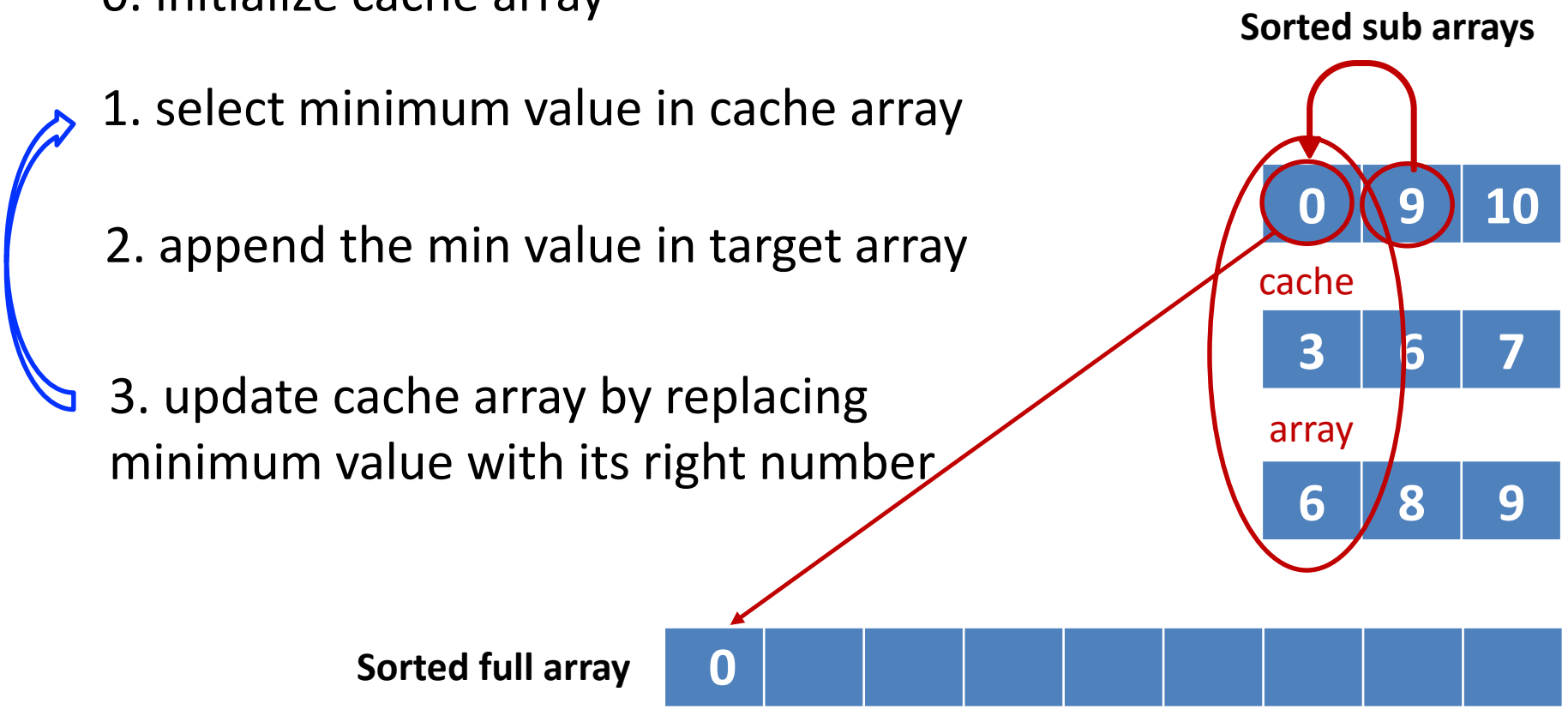
Solution 1: single thread

0. initialize cache array

1. select minimum value in cache array

2. append the min value in target array

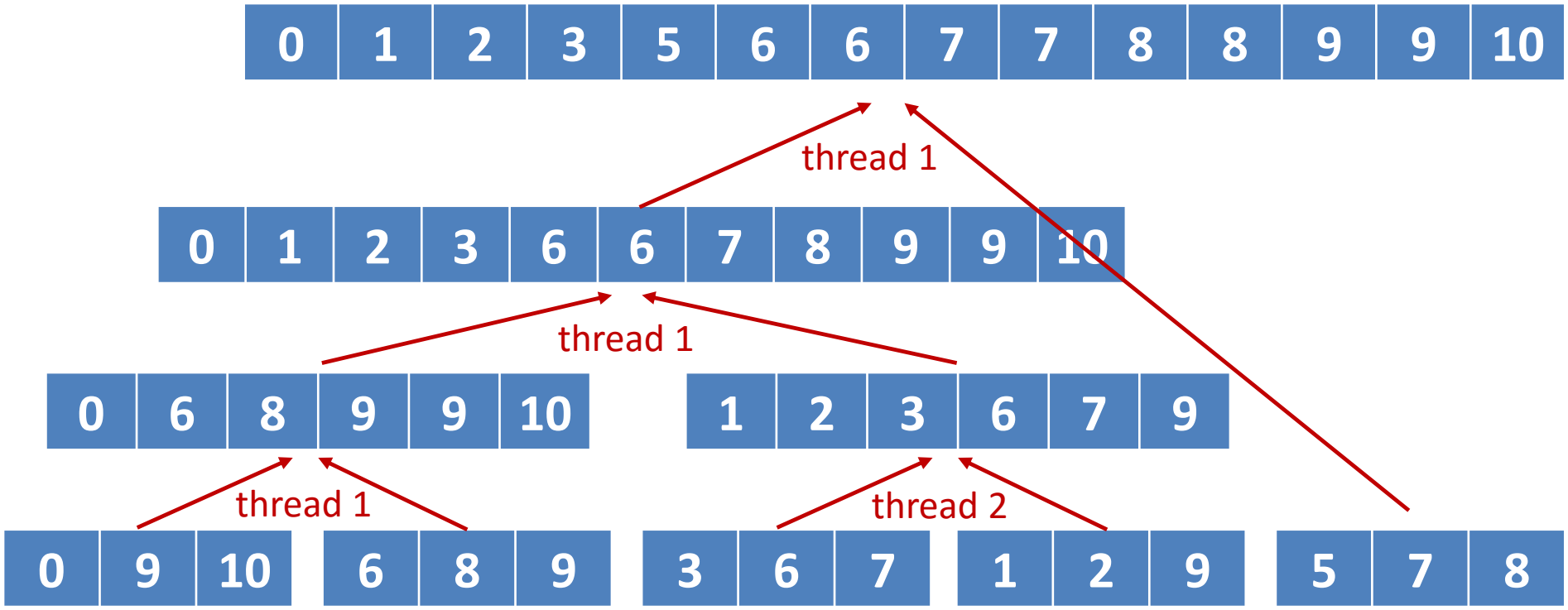
3. update cache array by replacing minimum value with its right number



Sort & Merge

- Merge algorithm: merge N sorted arrays

Solution 2: multi thread



Task 2: Finding median

- For Task 2, you are expected to do the following:

1. Read data from “input.txt” and store it in an array (array length=524288).

2. Partition the array into N parts, and run N threads for merge sort.
   => You are free to choose any sorting algorithm.
   => In-place or out of place sorting is ok.

3. Combine the N sorted subarrays into one sorted array.

4. Print out the sorted array.

5. Compute median of the array (should be 4999.0).

6. Do the experiment for different number of threads and record its corresponding execution time (see the following table), and plot it.

Number of threads	1	2	4	8	16	32	64	128	256	1024	2048
Execution time (ms)											

Code tips: multi-thread (Java)

```java
public class MeanThread {
    public static void main(String[] args) throws InterruptedException, FileNotFoundException {
        // TODO: Read file and store the data in an array
        // TODO: partition the array list into N subArrays part
        // TODO: create N threads and assign subArrays to the threads
        MeanMultiThread thread1 = new MeanMultiThread(subArray1);
        MeanMultiThread threadn = new MeanMultiThread(subArrayn);
        // TODO: get the N mean values
        thread1.start(); //start thread1 on from run() function
        threadn.start();//start thread2 on from run() function
        thread1.join();//wait until thread1 terminates
        threadn.join();//wait until threadn terminates
        // TODO: show the N mean values
        System.out.println("Mean value of thread n is ... ");
        // TODO: get the mean value from N mean values
        System.out.println("The global mean value is ... ");
    }
}
//Extend the Thread class
class MeanMultiThread extends Thread {
    private ArrayList<Integer> list;
    private double mean;
    MeanMultiThread(ArrayList<Integer> array) {
        list = array;
    }
    public double getMean() {
        return mean;
    }
    public void run() {
        // TODO: implement your actions here, e.g., mean(...)
        mean = mean(list);
    }
}
```

Code tips: multi-thread (C)

```c
int main (int argc, const char * argv[])
{
    //TODO: do something

    // prepare data0, data1
    pthread_t workers[2];
    pthread_create(&workers[0], NULL, get_temporal_mean, data0);// create thread1
    pthread_create(&workers[1], NULL, get_temporal_mean, data1); // create threade2
    //TODO: do something

    pthread_join(workers[0], NULL);// wait thread1 to finish
    pthread_join(workers[1], NULL);// wait thread2 to finish
    //TODO: do something

}

// get mean values for sub arrays
void *get_temporal_mean(void *params) {
    //TODO: do something
    pthread_exit(NULL);
}

// get global mean value
void *get_global_mean(void *params) {
    //TODO: do something
    pthread_exit(NULL);
}
```

Code tips: multi-thread (C)

C example: how to use *pthread*

pthread_create(&workers[0], NULL, get_temporal_mean, data0); // create thread

pthread_exit(NULL); // terminate thread

pthread_join(workers[0], NULL); // wait until thread terminates

Requirement

1. Submit your codes and description file at eDimension **before 23:59 22 Feb, 2017!**

2. Remember adding your name and student ID in your submitted file.