

i Information

Written Exam in IN3030/IN4330

2025 Spring Semester

Duration: 4 hours

Date: May 27th, 2025

General information:

- IN4330 students are given the same questions as IN3030 students but are expected to give better and more in-depth answers.
- Your answer should reflect your own independent work and should be a result of your own learning and work effort.
- Concerning answers that involve Java programming: the prime concern is what the program does; it is a secondary concern that the program is entirely syntactically correct, so good pseudo-code where boring or less relevant Java details are missing is acceptable: Spend your time on getting the algorithms right rather than the syntax. Do not worry about importing libraries.

Support material:

All written materials. All lecture materials from 2025 and 2024 are available as pdfs in Inspera at the bottom of the page.

The answer must be written with an academic standard.

You can answer in English, Norwegian (bokmål or nynorsk), Danish or Swedish. Or a mixture of these languages.

1 Paralellization of Shell Sort

Shell Sort

Given the sorting algorithm below, describe how you would parallelize it, so that you achieve a speedup for large arrays (the example below is merely for an array of size five).

IMPORTANT: your parallel algorithm MUST be LOYAL to the original algorithm. As an example of a NON-LOYAL solution is one where each of k threads does a Shell Sort of $1/k$ of the array followed by a merge sort of the k sorted parts. This is not acceptable as it substantially modifies the original algorithm: if we use the same technique for the sequential version, we will vastly improve the original sequential algorithm. Thus you are to parallelize the algorithm merely by reordering the original sorting steps and then running some of them in parallel.

```
// Java implementation of Shell Sort
// Shell Sort is a variant of Insertion Sort.
// When running it with a gap of 1, it is essentially Insertion Sort
// By first running with large gaps then large elements are moved large distances
// quickly, thus making the final passes with small gaps more efficient.
```

```
class ShellSort {
    /* An utility function to print array of size n*/
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    /* function to sort arr using shellSort */
    int sort(int arr[])
    {
        int n = arr.length;

        // Start with a big gap, then reduce the gap
        for (int gap = n / 2; gap > 0; gap /= 2) {
            // Do a gapped insertion sort for this gap size.
            // The first gap elements a[0..gap-1] are already
            // in gapped order keep adding one more element
            // until the entire array is gap sorted
            for (int i = gap; i < n; i += 1) {
                // add a[i] to the elements that have been gap
                // sorted save a[i] in temp and make a hole at
                // position i
                int temp = arr[i];

                // shift earlier gap-sorted elements up until
                // the correct location for a[i] is found
```

```
int j;
for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {
    arr[j] = arr[j - gap];
}

// put temp (the original a[i]) in its correct
// location
arr[j] = temp;
}
}
return 0;
}

// Driver method
public static void main(String args[])
{
    int arr[] = { 12, 34, 54, 2, 3 };
    System.out.println("Array before sorting");
    printArray(arr);

    ShellSort ob = new ShellSort();
    ob.sort(arr);

    System.out.println("Array after sorting");
    printArray(arr);
}
}
/*This Java code was written by Rajat Mishra */
```

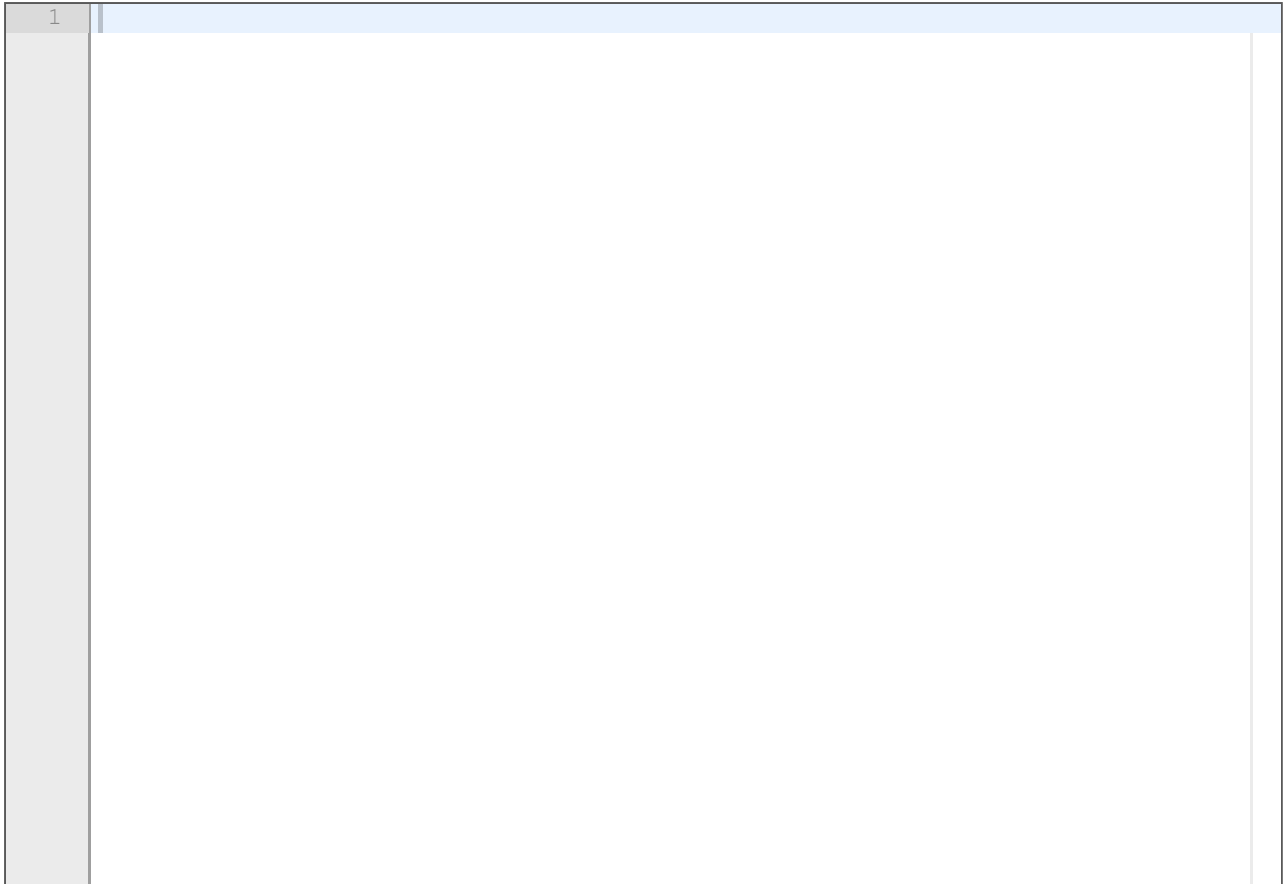
Write your suggestion here:

Maximum marks: 20

2 Implementation of parallel Shell Sort

Implement your suggested parallel version of Shell Sort from the previous question. Make sure that your suggestion is LOYAL to the original algorithm.

Write the code for your parallel version of Shell Sort



Maximum marks: 12

3 Cache friendliness of Shell Sort

Is the Shell Sort program presented earlier in this exam (in the first problem) cache friendly? Comment on why or why not it is cache friendly. Note that parts of an algorithm can be cache friendly, while other parts can be cache unfriendly.

Fill in your comments here



Maximum marks: 14

4 Speed of Light in Vacuum

What is the speed of light in vacuum? Choose the precise answer to get two points (an approximate answer that is within 1 percent of the correct value will be given one point).

Choose ONE alternative:

- ☐ 300 000 000 km/s
- ☐ 299 792 458 m/s
- ☐ 299 792 458 km/s
- ☐ 299 458 792 km/s
- ☐ 300 000 000 000 m/s
- ☐ 300 000 km/s

Maximum marks: 2

5 Effects of the speed of light

The slow speed of light:

Select one alternative:

- ☐ The latency it causes can be overcome by increasing the clock cycle on CPUs. For example, increasing the CPU speed by 50% will reduce the latency by a factor of 1.5.
- ☐ Can be significantly increased by using fiber optic cables.
- ☐ The slow speed of light can be compensated for by using caching.
- ☐ Moore's law has for many years meant that CPU speeds have been exponentially increasing thus making the latency caused by the slow speed of light insignificant.
- ☐ Is not important because the latency that it causes is significant only for machines far apart.

Maximum marks: 2

6 Moore's Law

Moore's law essentially says that: (only the MOST correct answer gives points)

Select one alternative:

- ☐ The number of transistors that can economically be put onto a square centimeter of a chip doubles every 1-2 years.
- ☐ Doubling the voltage to a chip doubles the speed of the chip.
- ☐ CPU speeds double every year.
- ☐ Doubling the clock cycle of a chip reduces the heat dissipated by 50%

Maximum marks: 2

7 Distance travelled by light in vacuum

What is the distance travelled by light in vacuum in 1 nanosecond? (The best answer gives full points; an answer within 50% gives partial points):

Select one alternative:

- ☐ About 30 cm.
- ☐ About 10 cm.
- ☐ About 333 cm.
- ☐ About 26 cm.
- ☐ About 3 meter.
- ☐ About 1 mm.
- ☐ About 1/3 meter.

Maximum marks: 2

8 Panama Canal Locks

Panama Canal Locks Synchronization

This question is about writing code to synchronized ships that are entering a lock on the Panama Canal.

A canal lock connects two part of a body of water where the water level in the two parts are different. A lock has doors at each end. When a door is open at one end, then the water level is the same on both sides of the door. This allows one or more ships to sail into the lock where after the door closes, the lock fills with water until the water level in the lock is the same as the water level at the other end of the lock. Thereafter the doors at that end can open and the ship can continue its journey - now at a higher level.

For simplicity, we assume that we are administrating the lock as a one way lock and thus can assume that there is no traffic going the opposite direction. The lock operates in rounds where it accepts ships as long as there is room. If there is not enough room for a ship, it waits until there is space for it in a later round. The waiting ships wait in a first-come, first-served order.

Every half hour, the Harbor Master closes the lock, if there is at least one ship in the lock. Once closed, the lock fills with water raising the ships in the lock, whereafter the lock is opened in the other end and the ships leave one by one, then the lock closes, lets the water out until the water level in the locks is level with the water at the lower end of the lock and then the locks is reopened for another round of ships. You can assume that any ship that is signalled that it can leave the lock, leaves immediately, i.e., after the last ship is signalled, you can reopen the lock at the low end immediately thus ignoring that in the real world, it takes time for the ships to leave and to close the lock at the upper end, and empty the lock: you may assume all this happens in no time.

Upon reopening, any waiting ships are allowed to reenter in order, if there is room, otherwise they must wait for yet another round. Note that this means that shorter ships may pass a longer ship. You may assume that all ships are shorter than the total length of the lock.

Write a Java class, **PanamaLockC**, that has the following methods:

void enterLock(int shipId, int shipLength)

that is called by a ship wanting to enter the lock. Each ship has an unique id, shipID, which is just an integer. The ship also provides its length (rounded up to nearest meter).

If the lock is closed, the ships waits (on a semaphore) until the lock is open again.

When the lock is open, the ship is allowed to enter the lock, but only if the remaining space is large enough to hold the ship, otherwise it waits for the next round of ships entering the lock.

Once in the lock, the ship waits for the ship to be released from the lock where after the call to **enterLock** returns.

void closeLock()

that is called every half hour by the Harbor Master. If there is no ship in the lock, **closeLock** returns immediately. Otherwise, it closes the lock, fills the lock with water, opens the lock doors at the high end. It then releases the ships in the lock by signalling them on whatever semaphore they are waiting. As mentioned, we assume that the ships leave the lock in no time and so after the last ship in the lock has been signalled, the the doors at the upper end close and then you can unlock the lock doors at the low end and start another round (e.g., by signalling a semaphore where ships might be waiting) and, finally, **closeLock()** returns.

At first, you do not have to write the code for a Ship, nor the code for the Harbor Master in your answer below. (The code for a Ship and for the Habor Master is to be part of your answer to the NEXT problem.)

ALL synchronization is to be done using Semaphores *only*. You are only allowed to use the **acquire** and **release** methods of Semaphore.

Write the **PanamaLockC** class in Java below, assume that the total length of the lock is a parameter to the constructor of the class:

Write your PanamaLockC Java Class here:

1

Maximum marks: 18

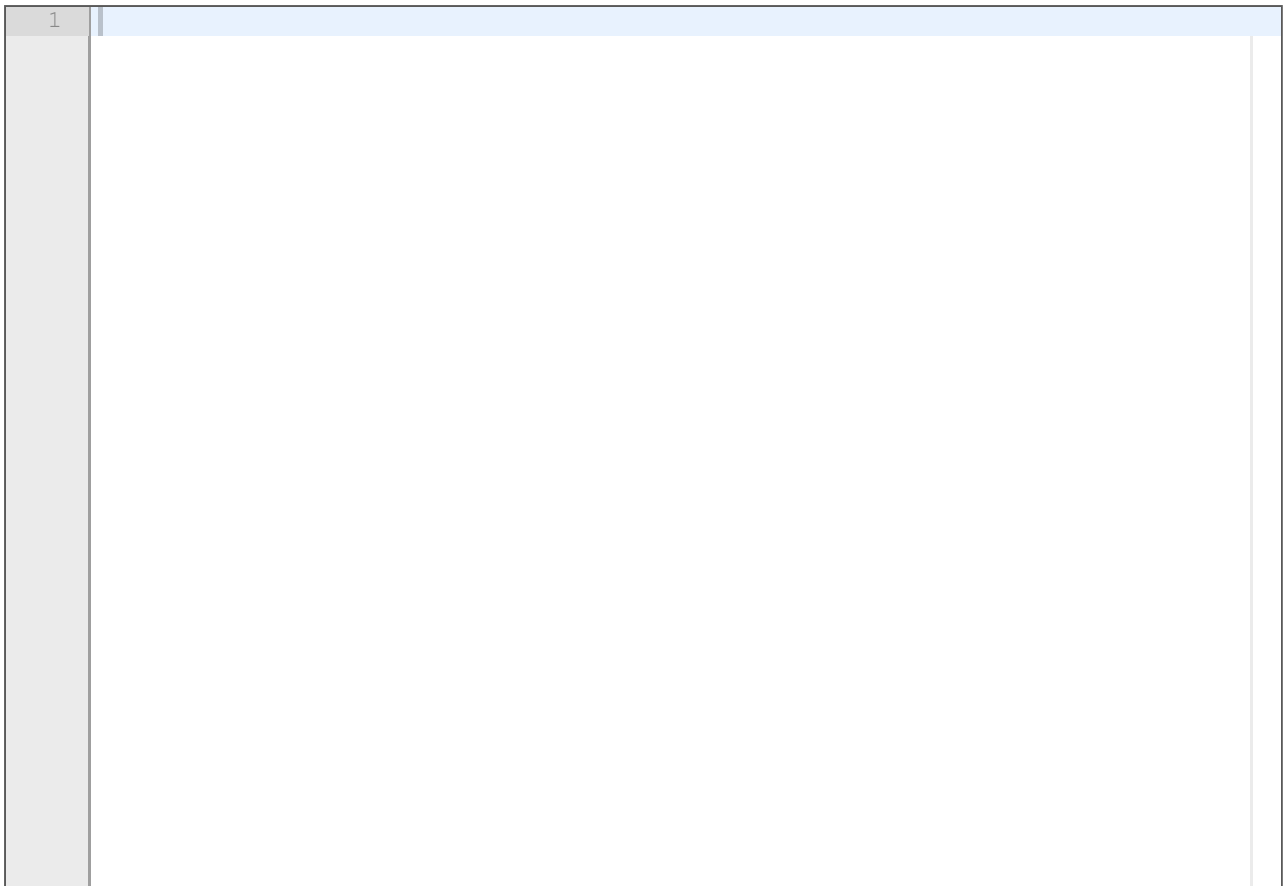
9 Test Program for PanamaCanaC

Write a Java test program that demonstrates the representative unit testing for the implemented your **PanamaCanalC** class. Ensure that the output of the test explains the steps of the execution. You will likely have to write several classes, one for ships, one for the Harbor Master, and perhaps a test program.

Explain the test that you chose and why it gives good test coverage.

Submit the program, its output and any comments that you might have.

Write your Java test program here:



Maximum marks: 8

10 Amdahl's Law

If a program has an initial sequential part taking 50% of the execution time, what is the maximum achievable speedup according to Amdahl's Law regardless of how many cores that are available?

Select one alternative:

- ☐ 4
- ☐ 3
- ☐ 8
- ☐ 1.5
- ☐ 0.2
- ☐ 2
- ☐ 8/10

Maximum marks: 2

11 Max Speedup

If all parts of a program are easily parallelizable, what is the maximum achievable speedup on an 16 core machine? Assume that the effective number of cores is 16, i.e., including any performance boosting techniques such as hyperthreading.

Select one alternative:

- ☐ 16
- ☐ 4
- ☐ 32
- ☐ 8

Maximum marks: 2

12 Max Speedup with 64 Cores

On a machine that *effectively* has 64 cores (assume no hyperthreading), when you achieve a speedup of greater than 64, what can you conclude:

Select one alternative

- ☐ Your original sequential program was not optimal.
- ☐ Something is wrong: With 64 cores speedups cannot exceed 32 because of Amdahl's law. So a speedup of more than 32 is indication that the program is wrong.
- ☐ By applying suitable parallelization techniques, it can often be possible to achieve speedups of way more than 64. So you have done a good job.
- ☐ You have a sufficiently large L1 and L2 caches.

Maximum marks: 4

13 Java Threads Termination

Java Threads Termination

In a Java program, if the main program terminates, what happens?

Select an alternative:

- ☐ The remaining threads continue executing.
- ☐ When the main() method terminates, the entire program including all other threads terminate.
- ☐ The main thread will cause the termination of ALL threads and will await that they have all terminated before it itself terminates.
- ☐ All threads created by the main() program terminate.

Maximum marks: 3

14 Java Threads Execution on Multicore

Java Threads Execution on Multicore

Which of the following statements are true for Java Threads that are created in the same program? More than one statement might be true, so select all alternatives that you think are true statements. An correct answer gives one point; an incorrect answer -1 point; a blank answer 0 point.

Select one or more alternatives:

- ☐ Java threads always execute one at a time. All the others must wait.
- ☐ Java threads can execute in parallel independently of one another on multi-core machines.
- ☐ Each core runs one Java thread. When that thread terminates, another Java thread is selected and run until it terminates - and so on.
- ☐ If there are more threads in a program than available cores, the program cannot be run.
- ☐ All Java threads execute at the same speed because each thread is assigned to its own core .
- ☐ Java threads appear to execute in parallel independently of one another even on single-core machines because the Java Virtual Machine runs each thread for a short time then switches to another thread, if there is one that wants to run. This is achieved by time slicing the threads.
- ☐ Java threads execute in parallel: Every core runs a thread until it either terminates or has used up a time slice, whereafter another thread (possibly the same thread) is run on the core - and so on. If no threads are available to run, the core waits for a runnable thread to show up.

Maximum marks: 4

15 Safety of Java

Safety of Java

When using multiple Java threads, it is, in general, safe to do the following (choose the answers that are true; an extra point when you have chosen exactly the set of correct answers - note that there can be between 0 and 5 answers that are true):

Select one or more alternatives:

- ☐ Two Java threads can safely write to adjacent array elements in an integer array.
- ☐ It is OK for two threads to both read and write a shared integer variable as long as they make sure to synchronize their access to the variable.
- ☐ When multiple threads simultaneously update a shared integer then it is always safe, if the integer is an instance of the Java class AtomicInteger.
- ☐ Two Java threads cannot write to different elements in an array safely.
- ☐ Two threads can READ from the same memory location, e.g., the same element in an array without synchronization problems as long as no other thread writes to the memory location.

Maximum marks: 5