**CIS6007 Parallel and Distributed Systems**

Artem Meshcheryakov – [st20283944](https://e.mail.ru/compose?To=st20283944@outlook.cardiffmet.ac.uk) (Cardiff), 2117065 (VUM). Email: [st20283944@outlook.cardiffmet.ac.uk](https://e.mail.ru/compose?To=st20283944@outlook.cardiffmet.ac.uk)

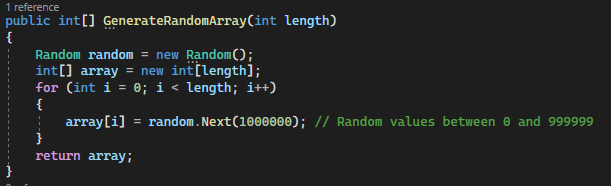
# **Task 1 (Bubble sort):**

**1. The Structure of the Program:**

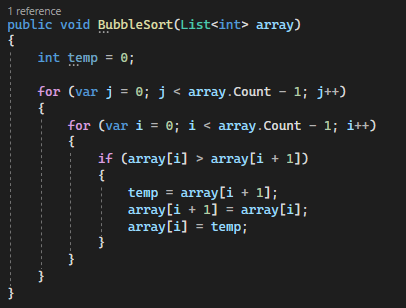
The program for Task 1 is relatively simple and consists of a parallel implementation of the bubble sort algorithm. It involves dividing the input array into segments and sorting these segments concurrently. Here's a high-level structure of the program:

* Generate a random array of at least 100,000 elements.
* Implement a parallel version of the bubble sort algorithm.
* Measure execution time for 2, 3, 4, and 6 threads.
* Write the execution times to a terminal.

Generator:



Algorithm itself:



**2. Evaluation of Task 1:**

Is this problem able to be parallelized?

* Yes, bubble sort can be parallelized by dividing the array into segments and sorting these segments concurrently.

How would the problem be partitioned?

* The problem is partitioned by dividing the array into smaller segments, and each segment is sorted independently.

Are communications needed?

* Minimal communication is required for merging sorted segments back together after sorting.

Are there any data dependencies?

* No, data dependencies are minimal in bubble sort as each segment can be sorted independently.

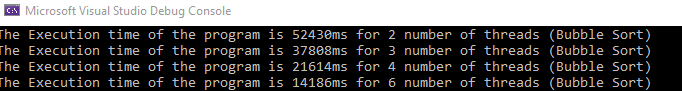
Are there synchronization needs?

* Yes, synchronization is needed to ensure that segments are not overwritten while sorting.

Will load balancing be a concern?

* Load balancing may be a concern if segments have varying sizes, but this can be managed by dynamically assigning segments to threads.

**3. Test Results of Task 1:**

****

The time of program execution gradually decreases with the increasing amount of threads which is a great indicator of the success of implementation.

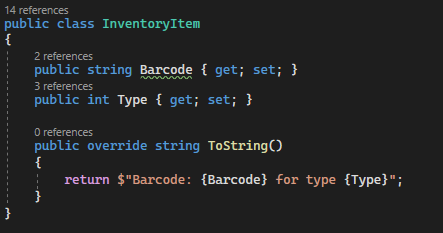
# **Task 2 (Inventory Searching):**

**1. The Structure of the Program:**

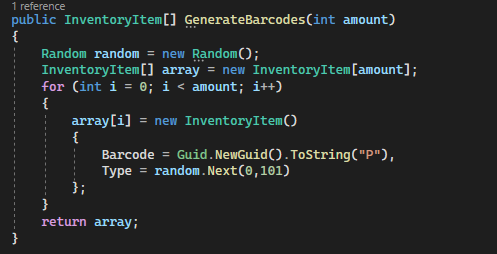
The program for Task 2 involves searching a large collection of inventory in parallel for specific items of different types. Here's a high-level structure of the program:

* Create a class of the instance of Inventory Item which has a type (int), barcode (string)
* Generate an array with 100 000 elements with a random barcode and a random type which is a number between 1 and 100.
* Create an algorithm which will search for a specific type through the array and stop when the necessary amount is found.
* Execute the algorithm on 2, 3, 4 and 6 threads.
* Measure the time of the execution and print results into the terminal.

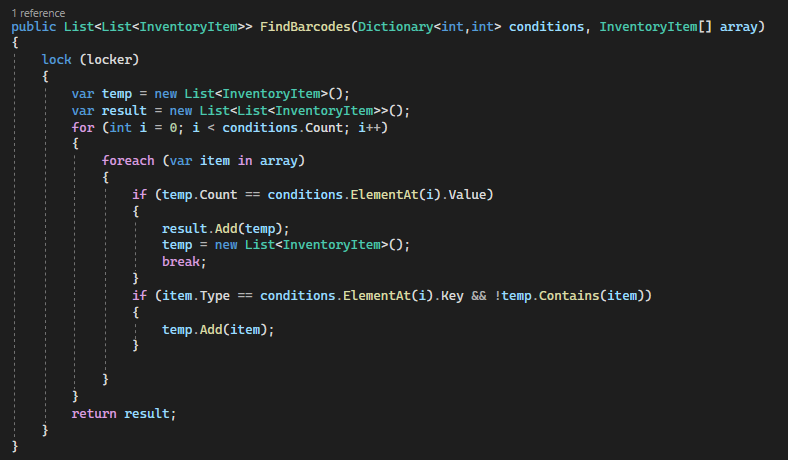
Inventory Item Object:



Generator:



Algorithm itself:



**2. Evaluation of Task 2:**

Is this problem able to be parallelized?

* Yes, Searching algorithms can be paralyzed by dividing a grand array into segments which will be divided by the number of threads.

How would the problem be partitioned?

* The problem is partitioned by dividing the array into smaller segments, and necessary items will be extracted from the segments.

Are communications needed?

* Minimal communication is required for returning data from every segment.

Are there any data dependencies?

* No, data dependencies are minimal in searching algorithms cause every segment can be checked individually..

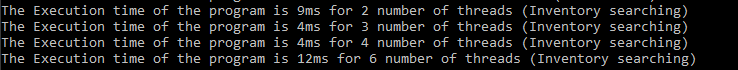
Are there synchronization needs?

* Yes, synchronization is needed to ensure that segments are not overchecked while searching.

Will load balancing be a concern?

* Load balancing may be a concern if the segment consists of all necessary items, but the chances of it not that high and can be managed by breaking the loop if a necessary amount has been collected.

**3. Test Results of Task 1:**

****

The execution time is the best for 3 or 4 threads and the worst for 6 threads. Possible Explanation for that could be overflowing the pc memory. However the results are still pretty fast and can be executed on several threads with not significant time loss.