Name:	Jude Onyia
Student ID:	V00947095
Course:	ECE 596C
Due Date:	July 3, 2020

Assignment 4: Non – Programming Exercise

8.17 a)

For this scenario, one can use a bounded array container (i.e. the standard library's array class). Assuming there is sufficient memory to hold the large set of integers, this container should be sufficient because all its contents are known at the time of its creation and they do not change. Also, the container is contiguous in memory, therefore, it can easily and quickly iterate over elements in sorted order. A particular value in this container can easily be found.

8.17 b)

A doubly linked list container can be used for this scenario. For example, the boost::intrusive::list container can be used because elements need to be inserted in and removed from any position in the container. Assuming a sorted order that is the same as the container's order, the container also allows easy iteration over the elements in the sorted order because each element can find its successor and predecessor in constant time. A particular value in this container can easily be found.

8.17 c)

For this scenario, a singly linked list container will be sufficient. An example is the boost::intrusive::slist container, which can be used because elements need to be inserted in any position in the container. Assuming that the element with the largest value is always the element at the start of the list, this container can accomplish the task of removing the largest value quickly because it can remove elements at the start of the list in constant time. Since the maximum size is known at the time of its creation, memory can be allocated for the list (without creating any objects / calling any constructors).

8.18)

In a case where a program uses a container that can store a certain maximum number of objects, one might want to separate the operations of memory allocation and construction. This is desired because one would want enough memory available for the container to use if it needs to hold the maximum amount of objects, but one would not want to spend time and resources creating that many objects if it might not need them. Therefore, these operations must be separated so that after allocation, memory is guaranteed for the container, but objects are not created in that memory until the objects are needed.

Similarly, one might want to separate the operations of memory deallocation and destruction because one might want to destroy each object right after processing. Then, when the container is empty, quickly deallocate the memory for the container. If these operations are not separated, this would lead to having to destroy all the objects at the end and spending a considerable amount of time destroying objects before deallocating. This amount of time spent at the end might be very inconvenient.

8.19)

Two advantages that array-based containers have over node-based containers include:

- Array-based containers can access any element in constant time, whereas, node-based containers can only guarantee finding a successor (and predecessor if doubly linked) in constant time.
- Array-based containers have less overhead than node-based containers because array-based containers use contiguous memory.

8.23 a)

In this scenario, an intrusive container should be used because the object type is neither copyable nor movable, therefore, nonintrusive containers cannot be used since they depend on creating a copy or moving an object. Also, some objects could be in the first container (for existing processes) and the second (for processes ready to run). Unlike nonintrusive containers, intrusive containers allow objects to be in more than one container.

8.23 b)

An intrusive container should be used since none of the nonintrusive containers that are available for use provide stable references due to other design constraints.

8.23 c)

A nonintrusive container should be used because of the mutual exclusive nature of std::mutex. It can not belong to more than one container, this would defeat the purpose of mutual exclusion, therefore, it must be in a nonintrusive container. Assuming all mutexes are created within their respective containers.