8.24

a)

An ordered, array-based set would be an ideal container for these purposes, since searching for elements is typically very quick (O(log n)). Also, since the set is ordered, elements can be iterated over in the order that they are stored in memory, making the iteration process fast and easy. Since all elements are known at the time of the set's creation, it is safe to use an array to store the elements and their location will never need to change.

b)

An ordered, node-based set would be an ideal container for these purposes. Since the container sees frequent insertions and deletions, the node-based structure will provide a performance boost over an array-based set. Searching for values is quick as per usual with an ordered set. Iterating over ordered elements is fast too since the set is stored in an orderly fashion.

c)

An ordered, node-based set would be an ideal container for these purposes. While the node-based architecture increases memory usage, which is unfortunate being that the maximum size of the set is known at creation, it is still the optimal option because it allows the fastest insertion/removal process available. Values can be removed simply by popping one end of the set, and the combination of orderly storage and a node-based architecture allow quick insertions under all circumstances.

8.26

Consider the case where a user wants to create a contiguous area of memory to store objects, but does not yet know how those objects will be constructed. In this case it is advantageous to allocate memory without constructing, since the user does not yet know how to properly construct the objects. In general, allocating without constructing is beneficial when you know object(s) will be constructed somewhere, but you don't know exactly how to construct them.

Destruction without deallocation is beneficial when the user may want to construct other objects in the place of one that is no longer useful. For example, it would be beneficial to destroy an object that allocates memory so that its destructors run and deallocate any relevant memory, but if it was known that another object of the same type might be constructed in its place later on then it would be advantageous to leave the memory allocated.

8.27

1. Array-based containers demand less overhead per element stored since elements are stored contiguously in memory
2. Array-based containers are effective when there is need to access elements by their position in the container, again since elements are stored contiguously in memory

8.29

a)

An intrusive container must be used, for if it weren't it would be impossible to "move" a non-moveable and non-copyable process from the scheduling queue to the process list or vice versa. Furthermore, it would increase performance by omitting the need to copy/move all the state captured in the processes. Assumption: performance is important in this scenario.

b)

An intrusive container should be used. Since there aren't any nonintrusive containers that provide stable references, the lack of which is one of the major downsides of intrusive containers, one might as well go with an intrusive container and get the performance benefits.

c)

An intrusive container should be used. Since the mutexes only ever need to exist inside the container, there should be no issues where a mutex is moved out and causes references to be destabilized.