DATA STRUCTURE COMPLEXITY TABLE:

AGNI DATTA

Type	Notes
Array	Most efficient use of memory; use in cases where data size is fixed.
List	Implementation is optimized for speed. In many cases, List will be the best choice.
Collection	List is a better choice, unless publicly exposed as AP.
Linked List	Many operations are fast, but watch out for cache coherency.
Stack	Should not be selected for performance reasons, but algorithmic ones.
Queue	Should not be selected for performance reasons, but algorithmic ones.

Type	Add to end	Insert at middle	Search for specific element
Array	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
List	best case $\Theta(1)$; worst case $\Theta(n)$	$\Theta(n)$	$\Theta(n)$
Collection	best case $\Theta(1)$; worst case $\Theta(n)$	$\Theta(n)$	$\Theta(n)$
Linked List	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$
Stack	best case $\Theta(1)$; worst case $\Theta(n)$	N/A	N/A
Queue	best case $\Theta(1)$; worst case $\Theta(n)$	N/A	N/A

Type	Random Access	In-order Access	Remove from end	Remove from middle
Array	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$
List	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$
Collection	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$
Linked List	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Stack	$\Theta(n)$	N/A	$\Theta(1)$	N/A
Queue	$\Theta(n)$	N/A	$\Theta(1)$	N/A