Function	**Common name**	**Running time**
$O(n!)$	Factorial	forever?
$O(2^c)$, $c > 1$	Exponential	> century
$O(n^c)$, $c > 3$	Polynomial	
$O(n^3)$	Cubic	31.7 years
$O(n^2)$	Quadratic	2.8 hours
$O(n \text{ sqrt } n)$		31.6 seconds
$O(n \log n)$	Linearithmic / Loglinear	1.2 seconds
$O(n)$	Linear	0.1 second
$O(\text{sqrt}(n))$	Root-n	$3.2 * 10^{-4}$ seconds
$O(\log n)$	Logarithmic	$1.2 * 10^{-5}$ seconds
$O(\log \log n)$	Double Logaithmic	
$O(1)$	Constant	

bag	**stack**	**queue**	**deque**
add	push	addBack	addBack
contains	pop	removeFront	addFront
remove	top	front	removeFront
	isEmpty	isEmpty	removeBack
			front
			back
			isEmpty

Applications

stack

```
1)Back and Forward Buttons in a Web Browser
2)Buffered Character Input
3)Checking Balanced Parenthesis
4)Conversion of infix to postfix
5)Evaluation of a postfix expression
```

Queues:

```
simulations (ie simulate a bank line/queue)
any collection where time is important
```

The **linked list** maintains a reference to a collection of elements of type link and allocates a new link every time a new element is added to the ADT. In contrast, a **dynamic array** uses a fixed large block of memory allocated at runtime. While inserting a link is a simple O(1) operation, inserting an element into a full of array has an expensive O (n) operation because a new array has to be created and all the elements copied over. Similarly, inserting an element within an **array** or deleing an element from the array are also costly, since all of the subsequent elements have to be shifted to accomodate a new element or to fill the gap from the element removed, both of these operations have a complexity of O(n) as well. Meanwhile, maintaining a **linked list** is much simpler, with adding, inserting, and removing all having the same complexity of O(1). Unfortunately, the **linked list** has 2 shortcomings: finding an element is always a O(n) operation, because linked lists don't have the capacity for indexing, unlike arrays. Also, **linked lists** consume more memory because while the array only needs to allocate 1 block of memory for every element it holds, the linked list has to allocate 1 block for the data and 1 or 2 blocks for pointers. Additionaly, accessing the data is faster in an **array** because the elements are stored in contiguous memory, whereas traversing an array requires that the computer follow a trail of pointers from one link to another.

```
  assert(index  < da->size);        assert (index >= o);              assert (da != 0);
struct dLink * newLink;
newLink = malloc (sizeOf (struct dLink));
assert (newLink != NULL);
```