4: n  
5: 2+ n+2n  
6: 2n  
7: n\*[2+n-1+2n-2] = n\*[3n-1]  
8:3\*(n^2-n)  
9: 4n  
11 1

LINEAR  
4: n+2  
5: 2  
6: 2+ 2n-2 + 2n-2 = 4n-2  
7: (n-1)\*3  
8: (n-1)\*4  
10:1

Stack: Add and remove from top (Last in First Out)

Methods: (**From the st1 2022**, **it appears that these methods must be done/known if the Stack class is asked for)**

push: adds element to top of stack  
pop: removes top element from stack and returns it.  
top: returns last added element without removing it  
size: returns size of stack  
isEmpty: returns whether stack is empty

**Performance:**  
Let n be number of elements.  
The space used is O(n)  
Each operation runs in time O(1) – as accessing array is time 1.  
  
**Limitations**Max size of stack must be pre-determined and cant change  
You cannot push an element on a full stack (error/exception)

**Parentheses Matching Algorithm 1**

As soon as a match is found, that match is popped.  
So: {[( then adding ] wont work, as ( wont match ].  
However: {[( then add ) = {[ then add ] = { then add }  
  
So: {[(])} wont work.  
But: {[()]} will work

**Evaluating Arithmetic Expressions using Parentheses**  
Remember \* first before +/-  
Associativity ect  
so: 14 – 3 \* 2 + 7 = (14 – (3 \* 2)) + 7  
  
**Arithmetic Expression III example**: When you encounter a operation that has a lower/equal precedence than the previous operator, you must first complete previous operation.

Eg  
4 – 3 \* 2 + 7  
4-  
4-3\*  
4-3\*2 + (don’t add + to stack yet)  
Must complete 3\*2 as + lower than \*  
so: 4 – 6 must be done as – and + have same precedence  
-2 + 7  
5  
**So**: You only ever push higher precedence operators on top of the stack, lower/equal result in calculations on the stack being done before being added.

**Queue ADT**

Insertions and deletions follow First in First Out.

Enqueue: add object of end of queue  
dequeue: removes and returns element at front of queue (returns object)

Other **methods**

object front: returns element at front without removing  
integer size: returns size  
Boolean isEmpty: if empty

**Using Array for Queues**

You treat the array in a circular way.  
The array is fixed length  
You have variables storing the front and end indexes of the variables.  
  
This is done so that sorting the array each time an element is removed is not needed, as the variables in the array fill/move along the circle.

Size() works by return: (N – f + r ) mod N  
where N is the max size of the array  
mod N gets the remained, so  
5 mod 10 = 5  
12 mod 10 = 2  
This way, if this returns 9 (if N is 10), then when enqueueing return fullQueueException.

In the same way, you don’t increase f and r by going f++, but instead doing   
f = (f+1) mod N. As you get f++ till the max size, then it wraps around to the beginning of the array size.

**Double ended queues**

Has methods:  
addFirst  
addLast  
removeFirst  
removeLast

Other supporting methdos, eg First: returns first without removing

The runtime of all these methdos is O(1)