**Linear Data Structures**

**ArrayList**

* Supported operations and complexity:
  + **size()**, **isEmpty()**, **get()**, **set()** – **O(1)**
  + **add()** – the operation runs in **amortized constant** time
  + adding **n** elements requires **O(n)** time
  + all of the other operations like: **add(int index, E element)**, **contains()**, **indexOf(), remove(int index) etc**., run in **linear** **time** **O(n)** (roughly speaking)

**Stack**

* Supported operations and complexity:
  + **size**(), **isEmplty**(), **push**(), **pop**(), **peek**()– **O(1)**
  + all of the other operations run in linear time (roughly speaking):
    - **forEach**()
    - **contains**()
    - etc…

**Queue**

* Supported operations and complexity:
  + **size**(), **isEmplty**(), **poll**(), **peek**()– **O(1)**
  + **offer**():
    - if we keep the reference to the that node – **O(1)**
    - If we have to chase pointers to that node – **O(n)**
  + all of the other operations run in linear time (roughly speaking):
    - **forEach**(), **contains**(),etc…

**Singly Linked List**

* Supported operations and complexity:
  + **addFirst**(), **removeFirst**(), **getFirst**(), **size**()– **O(1)**
  + How about operations on the **last element**?
    - **addLast**(), **removeLast**(), **getLast**() – again depends if we keep the reference to the last node or no can be constant – **O(1)** or linear – **O(n)**
  + operations that **index** into the list will run in **linear** **time** **O(n)** (roughly speaking)

Tree terminology

* **Node** – a structure which may contain a **value** or condition, or represent a separate **data** **structure**.
* **Edge** – the **connection** **between** one **node** and **another**.
* **Root** – the **top** node in a **tree**, the **prime** **ancestor**.
* **Parent** – the **converse** notion of a **child**, an **immediate** **ancestor**.
* **Child** – node **directly** connected to **another** node when moving **away** from the **root**, an immediate descendant.
* **Siblings** – a **group** of **nodes** with the **same** **parent**.
* **Ancestor** – node reachable by repeated proceeding **from** **child** **to** **parent**.
* **Descendant** – node reachable by repeated proceeding **from** **parent** **to** **child**.
* **Leaf** – node with **no** **children**.
* **Branch** – node with **at least one child**
* **Degree** – number of children for node zero for a leaf.
* **Path** – sequence of nodes and edges connecting a node with a descendant.
* **Distance** – number of edges along the shortest path between two nodes.
* **Depth** – distance between a node and the root.
* **Level** – depth + 1.
* **Height** – The number of edges on the longest path between a node and a descendant leaf.
* **Width** – number of nodes in a level.
* **Breadth** – number of leaves.
* **Height** – the maximum level in the tree.
* **Forest** – set of disjoint trees.
  + {17}, {9, 6, 5}, {14}, {15, 8}
* **Sub Tree** – tree T is a tree consisting of a node in T and all of its descendants in T.