1. Name a data structure that, given the index of an element, allows constant-time (no loops) access to elements:

2. Name a data structure that allows constant-time removal of the first element and linear-time access to elements given an index:

3. Name a data structure such that additions and removals follow last-in-first-out (LIFO) rule:

4. Consider this code:

   ```java
   SingleLinkedList<Integer> list = new SingleLinkedList<>();
   list.add(1);
   list.add(2);
   list.add(3);
   ```

   Draw a diagram of what this list looks like in memory:

5. Write this method:

   ```java
   /**
    * Prints out all the elements in an Iterable collection, one per line.
    * @param collection The collection to print out
    */
   public void printAll(Iterable<E> collection)
   ```

   You can find the interfaces `Iterable` and `Iterator` at the end of this set of practice questions.

6. Write this method:

   ```java
   /**
    * Copies all elements from this collection into an array. The size of the array must be exactly the number of elements in the collection
    * @param collection The collection to copy
    * @return The array containing all elements from the collection
    */
   public String[] toArray(Iterable<String> collection)
   ```
7. Consider this Node class and the method below it:

```java
public class Node<E>
{
    public Node<E> next;
    public E data;

    public Node(E d, Node<E> n)
    {
        next = n;
        data = d;
    }
}

public Node<String> wurble()
{
    Node<String> head = new Node<"a", null>;
    head.next = new Node<"b", null>;
    head.next = new Node<"c", head.next>;
    head = new Node<"d", head.next>;

    return head;
}
```

Draw a picture of the structure referred to by the reference returned from wurble().
public interface Iterable<T> {
    /**
     * Returns an iterator over elements of type T.
     * @return an Iterator.
     */
    Iterator<T> iterator();
}

public interface Iterator<E> {
    /**
     * Returns true if the iteration has more elements.
     * (In other words, returns true if next() would return an element rather than throwing an exception.)
     * @return true if the iteration has more elements
     */
    boolean hasNext();

    /**
     * Returns the next element in the iteration.
     * @return the next element in the iteration
     * @throws NoSuchElementException if the iteration has no more elements
     */
    E next();

    /**
     * Removes from the underlying collection the last element returned
     * by this iterator (optional operation). This method can be called
     * only once per call to next().
     * @throws UnsupportedOperationException if the remove
     * operation is not supported by this iterator
     * @throws IllegalStateException if the next method has not
     * yet been called, or the remove method has already
     * been called after the last call to the next
     * method
     */
    void remove();
}