Ejercicio 1

Hipótesis 1	El método *BS* funciona con el array [1, 0, 0, 0, 2, 0].
Predicción	La salida es el array ordenado [0, 0, 0, 0, 1, 2].
Experimento	Ejecutar *BS* con el array y su longitud (6) como parámetros.
Observación	El array resultante es [0, 0, 0, 1, 2, 0].
Conclusión	Hipótesis rechazada

Hipótesis 2	Se recorren y comparan todos los elementos del array en la función *BS*.
Predicción	Se harán tantas iteraciones del bucle *for* al ejecutarlo como número de elementos exista en el array menos 1 (pues en cada iteración se analiza el dato correspondiente al índice "i" con el siguiente, "i+1"), es decir, 5 veces.
Experimento	Ejecutar *BS* con los datos de la primera hipótesis y un *checkpoint* en el bucle *for*.
Observación	Se realizan únicamente 4 iteraciones cada vez que se ejecuta el bucle *for*.
Conclusión	Hipótesis rechazada

Hipótesis 3	La condición del bucle *for* en la función *BS* es incorrecta
Predicción	Si se cambia la condición del bucle con el fin de recorrer todo el array la función se ejecutará correctamente
Experimento	Ejecutar *BS* con los datos de la primera hipótesis y con la condición "count <= last – 1", cambiando el signo "menor que" por el "menor o igual que".
Observación	Se realizan 5 iteraciones en el bucle y la función ahora devuelve la salida correctamente: [0, 0, 0, 0, 1, 2].
Conclusión	Hipótesis confirmada

Hipótesis 4	El método *BS* funciona con el array [8, 7, 6, 5, 4, 9].
Predicción	La salida es el array ordenado [4, 5, 6, 7, 8, 9].
Experimento	Ejecutar *BS* con el array y su longitud (6) como parámetros.
Observación	El array resultante es [4, 5, 6, 7, 8, 9].
Conclusión	Hipótesis confirmada

Ejercicio 2

Hipótesis 1	El método *checkGoldbach* funciona con n=23.
Predicción	La salida son dos números primos que suman 23, o bien alertará de que no es posible encontrarlos.
Experimento	Ejecutar *checkGoldbach(23)*.
Observación	Los números primos que devuelve el algoritmo son el 19 y el 4, que no es primo.
Conclusión	Hipótesis rechazada

Hipótesis 2	El método *isPrime* (llamada dentro de *checkGoldbach*) funciona con n=4.
Predicción	La salida indica que 4 no es primo.
Experimento	Ejecutar *isPrime(4)*.
Observación	La salida devuelve *true*.
Conclusión	Hipótesis rechazada

Hipótesis 3	Se estudian todos los posibles números que puedan ser divisores de *n* en la función *isPrime* para n=4.
Predicción	Se comprobarán si el 2 y el 3 son divisores de 4 (el 1 y el 4 ya lo son por defecto).
Experimento	Ejecutar *isPrime(4)* con un *checkpoint* en el bucle *for*.
Observación	No se comprueba ni el 2 y ni el 3, pues p se inicializa a 2, y según la condición del bucle p debe ser menor que 1 ($\sqrt{4} - 1$).
Conclusión	Hipótesis rechazada

Hipótesis 4	La condición del bucle *for* de la función *isPrime* es incorrecta
Predicción	Si cambiamos la condición del bucle para que se comprueben todos los posibles números divisores de n la función se ejecutará correctamente.
Experimento	Ejecutar *isPrime(4)* con la condición del *for* "p < n".
Observación	Se comprueba tanto el 2 como el 3. Puesto que el 2 es divisor de 4, la función devuelve *false*.
Conclusión	Hipótesis confirmada

Hipótesis 5	El método *checkGoldbach* funciona con n=23.
Predicción	La salida son dos números primos que suman 23, o bien alertará de que no es posible encontrarlos.
Experimento	Ejecutar *checkGoldbach(23)*.
Observación	La función devuelve *false*, por lo que no es posible encontrar dos números primos que sumen 23.
Conclusión	Hipótesis confirmada

Hipótesis 6	El comentario que indica que si n es impar el método *checkGoldbach* devuelve *true* es incorrecto
Predicción	Si llamáramos a la función *checkGoldbach* con un número impar como el 23, ésta devolverá *false*.
Experimento	Ejecutar *checkGoldbach(23)*.
Observación	La función devuelve *false*. Según la conjetura de Goldbach, todo número par se puede descomponer en dos números primos. Es por ello por lo que solamente si n es par el método devolverá *true* y los dos números primos que sumen n.
Conclusión	Hipótesis confirmada

Ejercicio 3

Hipótesis 1	El método push del programa funciona.
Predicción	Llamando al método 4 veces cada una con un parámetro de tipo int (en nuestro caso: 1 2 3 4), se apila 1 2 3 4.
Experimento	Incluimos en el código un print que nos imprima los elementos de la pila tras apilarlos con el método push y ejecutamos.
Observación	Por pantalla nos imprime: null 1 2 3 4
Conclusión	Hipótesis rechazada porque no debería haber un elemento que fuese null.

Hipótesis 2	El método push empieza a apilar en la pos. 1 en vez de en la 0.
Predicción	El primer elemento se apila en la posición 1.
Experimento	Usamos el debug para ver qué valor toma top inicialmente.
Observación	Top=1.
Conclusión	Hipótesis confirmada

Hipótesis 3	Top se inicializa de manera incorrecta.
Predicción	Al crear la pila, top se inicializa a 1 en vez de a 0.
Experimento	Visualizamos dónde se inicializa top en el código.
Observación	En el constructor, nos encontramos con la sentencia top=1.
Conclusión	Hipótesis confirmada

Hipótesis 4	Si inicializamos top a 0, el método push apilará desde la posición inicial.
Predicción	El primer elemento se apilará en la posición 0.
Experimento	Cambiamos el valor de top y ejecutamos con los mismos datos que la primera hipótesis.
Observación	Se imprime: 1 2 3 4
Conclusión	Hipótesis confirmada

Hipótesis 5	El método push gestiona correctamente la capacidad máxima de la pila.
Predicción	Al llegar al máximo de elementos que puede tener la pila, el método push alertará de ello y no apilará el exceso.
Experimento	Tras crear la pila, llamamos al método 6 veces cada una con un parámetro de tipo int (en nuestro caso: 1 2 3 4 5 6).
Observación	Se imprime que la pila está llena, pero se genera una excepción de tipo IndexArrayOutOfBoundsException.
Conclusión	Hipótesis rechazada

Hipótesis 6	El método push necesita un else que comprenda las dos últimas instrucciones.
Predicción	Gracias a este else, no saltará la excepción porque no intentará apilar un elemento que ya no cabe en la pila.
Experimento	Ejecutamos el método con el cambio nombrado en la hipótesis y los mismos datos que en la anterior.
Observación	Se apilan los números del 1 al 5 correctamente y al intentar apilar el sexto se imprime “The stack is full” sin generar la excepción.
Conclusión	Hipótesis confirmada

Hipótesis 7	El método pull funciona correctamente.
Predicción	Al ejecutar el método pull tras haber apilado 1 2 3 4 en la pila, este devolverá el último de ellos (en este caso, 4).
Experimento	Creamos la pila, apilamos los 4 elementos y llamamos al método pull observando en el debugger qué nos devuelve.
Observación	Nos devuelve 3.
Conclusión	Hipótesis rechazada

Hipótesis 8	El método pull devuelve la posición anterior.
Predicción	Si en la instrucción de devolver, omitimos el -1 ya que hemos actualizado el top justo en la línea de encima (top--), nos devolverá 4.
Experimento	Creamos la pila, apilamos los 4 elementos y llamamos al método pull modificado observando en el debugger qué nos devuelve.
Observación	El método devuelve 4.
Conclusión	Hipótesis confirmada