1. 針對講義 P320, binary search 例子，利用等價分割的方式設計測試案例，並使用 JUnit 來測試
2. 針對講義 P325, next date 例子，利用等價分割的方式設計測試案例，並使用 JUnit 來測試

Binary Search

```
public static int binarySearch(int[] array, int key) {
        int left = 0, right = array.length - 1;
        int mid;
        while (left <= right) {
            mid = (left + right) / 2;
            if (array[mid] == key) {//found the hey
                return mid;
            } else if (array[mid] < key) {//key is bigger than
array[mid], search right side
                left = mid + 1;
            } else {//key is smaller than array[mid], search left
side
                right = mid - 1;
            }
        }
        //not found
        return -1;
    }
```

Binary Search Test

```
assertEquals(-1, App.binarySearch(new int[] {}, 1));// no element
not found

        assertEquals(0, App.binarySearch(new int[] { 1 }, 1));//
one element found
        assertEquals(-1, App.binarySearch(new int[] { 2 }, 1));//
one element not found

        assertEquals(0, App.binarySearch(new int[] { 1, 2 },
1));// two(even) element found at first
        assertEquals(1, App.binarySearch(new int[] { 1, 2 },
2));// two(even) element found at last
        assertEquals(-1, App.binarySearch(new int[] { 1, 2 },
3));// two(even) element not found
```

```
        assertEquals(0, App.binarySearch(new int[] { 1, 2, 3 },
1));// three(odd) element found at first
        assertEquals(1, App.binarySearch(new int[] { 1, 2, 3 },
2));// three(odd) element found at mid
        assertEquals(2, App.binarySearch(new int[] { 1, 2, 3 },
3));// three(odd) element found at last
        assertEquals(-1, App.binarySearch(new int[] { 1, 2, 3 },
4));// three(odd) element not found
```

把array分成沒有元素，一個元素，多個元素（奇數個及偶數個）

每個分割都測試有找到及沒找到

NextDate

```
public static String nextDate(String date) throws Exception {
        // YYYY/MM/DD
        String[] tokens = date.split("/");
        int year = Integer.valueOf(tokens[0]), month =
Integer.valueOf(tokens[1]), day = Integer.valueOf(tokens[2]);
        if (year < 1812 || year > 2012) {
            throw new Exception("year must between 1812 - 2012");
        }
        if (month < 1 || month > 12) {
            throw new Exception("month must between 1 - 12");
        }
        if (day < 1 || day > 31) {
            throw new Exception("day must between 1 - 31");
        }
        if (month == 2) {
            boolean isLeapyear = false;
            if (year % 4 == 0 && (year % 100 != 0 || year % 400 ==
0)) {
                isLeapyear = true;
            }
            if (day == 29 && isLeapyear == false) {
                throw new Exception(String.format("it does not
have 29 days in %d", year));
```

```
            }
            if ((day == 29 && isLeapyear == true) || (day == 28 &&
isLeapyear == false)) {
                day = 1;
                month += 1;
            } else {
                day += 1;
            }
        } else if (month == 1 || month == 3 || month == 5 || month
== 7 || month == 8 || month == 10 || month == 12) {
            if (day == 31) {
                day = 1;
                month += 1;
            } else {
                day += 1;
            }
        } else if (month == 4 || month == 6 || month == 9 || month
== 11) {
            if (day == 30) {
                day = 1;
                month += 1;
            } else {
                day += 1;
            }
        }
        if (month > 12) {
            month = 1;
            year += 1;
        }
        return String.format("%d/%02d/%02d", year, month, day);
    }
```

NextDate Test

```
// test 1812(min)
            assertEquals("1812/01/02",
App.nextDate("1812/01/01"));
            assertEquals("1812/01/11",
App.nextDate("1812/01/10"));
            assertEquals("1812/02/01",
App.nextDate("1812/01/31"));

            assertEquals("1812/05/01",
App.nextDate("1812/04/30"));
            assertEquals("1812/05/02",
App.nextDate("1812/05/01"));
            assertEquals("1812/06/01",
App.nextDate("1812/05/31"));

            assertEquals("1812/12/02",
App.nextDate("1812/12/01"));
            assertEquals("1812/12/11",
App.nextDate("1812/12/10"));
            assertEquals("1813/01/01",
App.nextDate("1812/12/31"));
            // test 1999(mid)
            assertEquals("1999/01/02",
App.nextDate("1999/01/01"));
            assertEquals("1999/01/11",
App.nextDate("1999/01/10"));
            assertEquals("1999/02/01",
App.nextDate("1999/01/31"));

            assertEquals("1999/05/01",
App.nextDate("1999/04/30"));
            assertEquals("1999/05/02",
App.nextDate("1999/05/01"));
            assertEquals("1999/06/01",
App.nextDate("1999/05/31"));

            assertEquals("1999/12/02",
App.nextDate("1999/12/01"));
```

```
            assertEquals("1999/12/11",
App.nextDate("1999/12/10"));
            assertEquals("2000/01/01",
App.nextDate("1999/12/31"));
            // test 2012(max)
            assertEquals("2012/01/02",
App.nextDate("2012/01/01"));
            assertEquals("2012/01/11",
App.nextDate("2012/01/10"));
            assertEquals("2012/02/01",
App.nextDate("2012/01/31"));

            assertEquals("2012/05/01",
App.nextDate("2012/04/30"));
            assertEquals("2012/05/02",
App.nextDate("2012/05/01"));
            assertEquals("2012/06/01",
App.nextDate("2012/05/31"));

            assertEquals("2012/12/02",
App.nextDate("2012/12/01"));
            assertEquals("2012/12/11",
App.nextDate("2012/12/10"));
            assertEquals("2013/01/01",
App.nextDate("2012/12/31"));
            // test leap year
            assertEquals("2000/02/29",
App.nextDate("2000/02/28"));
            assertEquals("2000/03/01",
App.nextDate("2000/02/29"));
            assertEquals("1900/03/01",
App.nextDate("1900/02/28"));
```

把年月日都分成min, mid, max

Year:1812, 1999, 2012

month:1, 2(閏年), 5, 12

Day:1, 10, 30, 31

針對每個組合進行測試

NextDate Range Test

針對min-, max+進行測試

```
// test year(min- , max+)
        assertThrows(Exception.class, () ->
App.nextDate("1811/01/01"), "year must between 1812 - 2012");
        assertThrows(Exception.class, () ->
App.nextDate("2013/01/01"), "year must between 1812 - 2012");
        // test month(min- , max+)
        assertThrows(Exception.class, () ->
App.nextDate("1812/00/01"), "month must between 1 - 12");
        assertThrows(Exception.class, () ->
App.nextDate("1812/13/01"), "month must between 1 - 12");
        // test day(min- , max+)
        assertThrows(Exception.class, () ->
App.nextDate("1812/01/00"), "day must between 1 - 31");
        assertThrows(Exception.class, () ->
App.nextDate("1812/01/32"), "day must between 1 - 31");
        // test wrong leap year
        assertThrows(Exception.class, () ->
App.nextDate("1900/02/29"), "it does not have 29 days in 1900");
```

測試結果