```java
package _requirements.part1;

public class HW2_Part1 {

        public static int example1(int[] arr) {
            int n = arr.length, total = 0;          // c1 & c2
            for (int j=0; j < n; j++)               // loop executed n times
                total += arr[j];
            return total;}                          // c3

        /* Total running time f(n) = O(n)
         * This method runs in linear time denoted by O(n). n is the number of items in
         * the array. If the array has 10 items, the loop runs 10 times.
         */


        public static int example2(int[] arr) {
            int n = arr.length, total = 0;          // c1 & c2
            for (int j=0; j < n; j+=2)              // loop executed n/2 times
                total += arr[j];
            return total;}                          // c4

        /* Total running time f(n) = O(n)
         * This method runs in linear time denoted by O(n). The loop adds j+=2, therefore
         * j becomes > n twice as fast as the first method. If the array has 10 items, the
         * loop runs 5 times. O(n/2) is a valid time complexity, however not the conventional
         * way of denoting running time. In this case, O(n) is understood.
         */


        public static int example3(int[] arr) {
            int n = arr.length, total = 0;          // c1 & c2
            for (int j=0; j < n; j++)               // loop executed n times
                for (int k=0; k <= j; k++)          // nested loop executed n times
                    total += arr[j];
            return total;}                          // c3

        /* Total running time f(n) = O(n^2)
         * This method runs in quadratic time denoted by O(n^2). This method has two nesting
         * loops. For an array that has n items, the outer loop runs n times and the inner
         * loop runs n times for each iteration of the outer loop, giving us n^2 total prints.
         * If the array has 10 items, it would iterate 100 times.
         */


        public static int example4(int[] arr) {
            int n = arr.length, prefix = 0, total = 0;  // c1 & c2 & c3
            for (int j=0; j < n; j++) {                 // loop executed n times
                prefix += arr[j];
                total += prefix;}
            return total;}                              // c4

        /* Total running time f(n) = O(n)
         * This method runs in linear time denoted by O(n). n is the number of items in
         * the array. If the array has 10 items, the loop runs 10 times.
         */

```

```
58
59          public static int example5(int[] first, int[] second) {
60              // assume equal-length arrays
61              int n = first.length, count = 0;        // c1 & c2
62              for (int i=0; i < n; i++) {             // loop executed n times
63                  int total = 0;
64                  for (int j=0; j < n; j++)           // nested loop executed n times
65                      for (int k=0; k <= j; k++)      // nested nested loop executed n times
66                          total += first[k];
67                  if (second[i] == total)
68                      count++;}                       // if statement executed n times
69              return count;}                          // c3
70
71          /* Total running time f(n) = O(n)
72           * This method falls into the ‘worst case’ running time denoted by O(n). It has three
73           * nested loops, however the if statement could be true after the first n iteration of
74           * the loop. The return statement is used to terminate a function early.
75           */
76
77      }
78
```