Problem 1 Write an algoirthm that ***runs in time $\boldsymbol{O(n)}$*** which takes a ***sorted*** array *A* of length *n* with no repeated values and an integer g. If there exist distinct indices i, j such that $A[i] + A[j] = g$, then return array $\{i, j\}$. If there are no such indices, return the array `{-1}`.

```
int [] twoElementSum(int n, int [] A, int g)  {. . .}
```

Problem 2 Write an algoirthm that ***runs in time $\boldsymbol{O(n^2)}$*** which takes an array *A* (***not*** sorted) of length *n* with no repeated values and an integer g. If there exist distinct indices i, j, k such that $A[i] + A[j] + A[k] = g$, then return array $\{i, j, k\}$. If there are no such indices, return the array `{-1}`.

```
int [] threeElementSum(int n, int [] A, int g)  {. . .}
```

Problem 3 Write an algorithm that ***runs in time $\boldsymbol{O(n)}$*** which takes ***sorted*** array *A* whose values are all distinct and ***sorted*** array *B* whose values are all distinct. The length of each array is *n*. The algorithm returns the number of values that are in both *A* and *B*.

```
int numValuesInBoth(int n, int [] A, int [] B)  {. . .}
```

Problem 4 Write the method below which takes an array *A* of length *n*. The user is prompted to enter integers one at a time until every integer in *A* is been inputted at least once. Once this happens, the algorithm returns the total number of integers entered by the user. (Why can't we determine the running time of this algorithm?)

```
int howManyEntries(int n, int [] A)  {. . .}
```

Problem 5 Write an algorithm that ***runs in time $\boldsymbol{O(\lg n)}$*** which takes a sorted integer array *A* of length *n* and an integer *x* and returns the index where *x* is located in *A*. If *x* is not in *A*, return the value `-1`.

```
int indexOf(int n, int [] A, int x)  {. . .}
```

Problem 6 Write an algorithm that ***runs in time $\boldsymbol{O(n^3)}$*** that determines if *n* by *n*, two-dimensional array *A* has two rows that are identical. For example, if rows 3 and 7 are identical, return the array `{3, 7}`. If there are no two identical rows, return the array `{-1}`.

```
int [] twoIdenticalRows(int n, int [][] A)  {. . .}
```

(continued on next page…)

Problem 7 Write an algorithm that ***runs in time $O(n^2)$*** which determines if *n* by *n* two-dimensional array *A* is ***totally sorted***, which means every row is sorted and the first value of each row is greater than the last value in the previous row.

```
boolean totallySortedTwoD(int n, int [][] A)  {. . .}
```

Problem 8 Write an algorithm that ***runs in time $O(\lg n)$*** which takes an *n* by *n* totally sorted two-dimensional array *A* and an integer *x*, and returns the coordinates where *x* is found in *A*. For example, if *x* is in row 8, column 4 of *A*, return the array `{8, 4}`. If *x* is not in *A*, return the array `{-1}`. (For this problem, the rows and columns are indexed from 0 to $n – 1$).

```
int [] coordinatesOf(int n, int [][] A, int x)  {. . .}
```

Problem 9 Write an algorithm that ***runs in time $O(n + k)$*** which takes integer array *A* of length *n* and a positive integer *k* and returns any integer between 1 and *k* which is not found in array *A*. If all integers between 1 and *k* are in *A*, return the value `-1`.

```
int missing(int n, int [] A, int k)  {. . .}
```

Problem 10 Write an algorithm that ***runs in time $O(n \lg n)$*** that takes a two-dimensional array *A* with *n* rows and two columns and determines if two rows of *A* are identical pairs. For example, if the pairs (`A[37][1], A[37][2]`) and (`A[68][1], A[68][2]`) are identical, the method would return the array `{37, 68}`. If there are no two identical pairs, it would return the array `{-1}`.

```
int [] twoIdenticalRows(int n, int [][] A)  {. . .}
```