Give Θ running times for each of the following. Identify best- and worst-case running times separately if there is a difference.

1.

```
int sum = 0;
for ( int i = 0 ; i < numbers.length ; i++ ) {
  sum += numbers[i];
}
return sum-(numbers.length-1)*numbers.length/2;
```

2.

```
boolean[] found = new boolean[numbers.length];
for ( int i = 0 ; i < numbers.length ; i++ ) {
  found[i] = false;
}
for ( int i = 0 ; i < numbers.length ; i++ ) {
  if ( found[numbers[i]] ) { return numbers[i]; }
  found[numbers[i]] = true;
}
```

3. We grow an array by increasing its length by 1 each time.

```
double[] numbers = new double[1];
for ( int i = 0 ; i < n ; i++ ) {
  if ( i >= numbers.length ) {
    numbers = Arrays.copyOf(numbers,numbers.length+1);
  }
  numbers[i] = Math.random();
}
```

4. We grow an array by doubling its length each time.

```
double[] numbers = new double[1];
for ( int i = 0 ; i < n ; i++ ) {
  if ( i >= numbers.length ) {
    numbers = Arrays.copyOf(numbers,numbers.length*2);
  }
  numbers[i] = Math.random();
}
```

5.

```
int findMatch ( String p, String t ) {
  for ( int i = 0 ; i <= t.length()-p.length() ; i++ ) {
    int j = 0;
    for ( ; j < p.length() && t.charAt(i+j) == p.charAt(j) ; j++ ) {}
    if ( j == p.length() ) { return i; }
  }
  return -1;
}
```

6.

```
void hanoi ( int n, int src, int dst, int spare ) {
  if ( n == 1 ) {
    System.out.println("move disk from "+src+" to "+dst);
  } else {
    hanoi(n-1,src,spare,dst);
    System.out.println("move disk from "+src+" to "+dst);
    hanoi(n-1,spare,dst,src);
  }
}
```

7. Mergesort.

```
void mergesort ( int[] arr, int left, int right ) {
  if ( right > left ) {
    int middle = (left+right)/2;
    mergesort(arr,left,middle);
    mergesort(arr,middle+1,right);
    merge(arr,left,middle,right);
  }
}

void merge ( int[] arr, int left, int middle, int right ) {
  int[] merged = new int[right-left+1];
  int int i = left, j = middle+1, k = 0;
  for ( ; i <= middle && j <= right ; k++ ) {
    if ( arr[i] < arr[j] ) { merged[k] = arr[i]; i++; }
    else { merged[k] = arr[j]; j++; }
  }
  for ( ; i <= middle ; i++, k++ ) {
    merged[k] = arr[i];
  }
  for ( ; j <= right ; j++, k++ ) {
    merged[k] = arr[i];
  }
  System.arraycopy(merged,0,arr,left,merged.length);
}
```

8.

```
int binarySearch ( int[] arr, int left, int right, int target ) {
  if ( right >= left ) {
    int mid = (left+right)/2;
    if ( arr[mid] < target ) {
      return binarySearch(arr,mid+1,right,target);
    } else if ( arr[mid] > target ) {
      return binarySearch(arr,left,mid-1,target);
    } else { return mid; }
  } else {
    return -1;
  }
}
```

9.

```
algorithm bubbleSort ( A, n ) :
  input: array A storing n items
  output: items in A are sorted in increasing order

  repeat
    swapped ← false
    for ( j ← 0 ; j < n-1 ; j++ ) do
      if A[j] > A[j+1] then    // if elements are reversed...
        temp ← A[j]            // ...swap them
        A[j] ← A[j+1]
        A[j+1] ← temp
        swapped ← true
  until swapped is false
```

10. We try to improve bubble sort by noting that each pass puts at least one element in place at the end of A so we can stop each pass earlier – there's no point re-checking things already known to be sorted.

```
algorithm smarterBubbleSort ( A, n ) :
  input: array A storing n items
  output: items in A are sorted in increasing order

  last ← n
  repeat
    swapped ← false
    for ( j ← 0 ; j < last-1 ; j++ ) do
      if A[j] > A[j+1] then    // if elements are reversed...
        temp ← A[j]            // ...swap them
        A[j] ← A[j+1]
        A[j+1] ← temp
        swapped ← true
    last--
  until swapped is false
```

11.

```
algorithm insertionSort ( A, n ) :
  input: array A storing n items
  output: items in A are sorted in increasing order

  for ( i ← 1 ; i < n ; i++ ) do
    value ← A[i]        // current value to insert

    // find insertion place, shifting things that come after
    for ( j ← i-1 ; j >= 0 and A[j] > value ; j-- ) do
      A[j+1] ← A[j]

    A[j+1] ← value            // do insertion
```

12. We try to improve bubble sort by moving elements backwards as well as forwards.

```
algorithm cocktailSort ( A, n ) :
   input:  array A storing n items
   output:  items in A are sorted in increasing order

   repeat
      // forward pass
      swapped ← false
      for ( j ← 0 ; j < n-1 ; j++ ) do
         if A[j] > A[j+1] then           // if elements are reversed...
            temp ← A[j]                  // ...swap them
            A[j] ← A[j+1]
            A[j+1] ← temp
            swapped ← true

      if swapped is false then
         break

      // backward pass
      swapped ← false
      for ( j ← n-2 ; j >= 0 ; j-- ) do
         if A[j] > A[j+1] then           // if elements are reversed...
            temp ← A[j]                  // ...swap them
            A[j] ← A[j+1]
            A[j+1] ← temp
            swapped ← true
   until swapped is false
```