1. Merge-sort practice problem for the AP Computer Science Exam

Merge sort is a recursive algorithm to arrange the elements of a data structure, such as an array or list, in increasing or decreasing order. The algorithm partitions the portion of the array into 2 unsorted halves, sorts each half, then merges the 2 sorted halves into 1 sorted whole.

On the AP Computer Science Exam, you must recognize the algorithm when it is presented in code.

Assignment — Merge-sort implementation

Complete the private recursive helper method `sort` below. The method sorts `a[low]`…`a[high]` while leaving the rest of the array untouched. You may assume that method `merge` works as specified. The merge portion of merge sort is not covered on the AP Computer Science Exam.

```
public class MergeSort
{
  public static void sort(int[] a)
  { sort(a, 0, a.length - 1); }

  private static void sort(int[] a, int low, int high)
  { /* to be completed */ }

   /** Precondition: a[low]...a[mid] is sorted && a[mid+1]...a[high] is sorted
    *  Postcondition: a[low]...a[high] is sorted */
  private static void merge(int[] a, int low, int mid, int high)
  {
    int[] b = new int[mid + 1 - low];
    System.arraycopy(a, low, b, 0, b.length);

    int aLowerIndex = low, bIndex = 0, aHigherIndex = mid + 1;
    while (aLowerIndex < aHigherIndex && aHigherIndex <= high)
    {
      if (a[aHigherIndex] < b[bIndex])
        a[aLowerIndex++] = a[aHigherIndex++];
      else
        a[aLowerIndex++] = b[bIndex++];
    }

    while (aLowerIndex < aHigherIndex)
      a[aLowerIndex++] = b[bIndex++];
  }
}
```

2. Selection-sort practice problem for the AP Computer Science Exam

Selection sort is an algorithm to arrange the elements of a data structure, such as an array or list, in increasing or decreasing order. The algorithm partitions the array into sorted and unsorted parts and repeatedly finds the minimum (or maximum) in the unsorted part and swaps it with the last element of the sorted part.

On the AP Computer Science Exam, you must recognize the algorithm when it is presented in code. You may be expected to compare the efficiency to insertion sort. You must also be able to recognize a variation of the algorithm such as sorting in decreasing order or sorting from back to front.

Selection sort trace

Show each step as selection sort is run on the following array:

[71, 86, 79, 36, 78, 35, 75, 86, 24, 11]

At each step, separate the sorted and unsorted parts with the | symbol.

Assignment — Selection sort implementation

Method `sort` arranges the elements in `x` in increasing order using the selection sort algorithm. Complete method `sort` below. You may also declare and implement a `swap` method with the header of your choice.

```
public static void sort(int[] x)
```

3. Insertion-sort practice problem for the AP Computer Science Exam

Insertion sort is an algorithm to arrange the elements of a data structure, such as an array or list, in increasing or decreasing order. The algorithm partitions the array into sorted and unsorted parts and repeatedly inserts the first element of the unsorted part into its correct position in the sorted part.

On the AP Computer Science Exam, you must recognize the algorithm when it is presented in code. You may be expected to compare the efficiency to selection sort. You must also be able to recognize a variation of the algorithm such as sorting in decreasing order or sorting from back to front.

Insertion sort trace

Show each step as insertion sort is run on the array below.

```
[37, 32, 70, 15, 93, 40, 63, 3, 40, 63]
```

At each step, separate the sorted and unsorted parts with the | symbol.

Assignment — Insertion sort implementation

Method `sort` arranges the elements in `x` in increasing order using the insertion sort algorithm.
Complete method `sort` below.

```
public static void sort(int[] x)
```

4. Sequential-search practice problem for the AP Computer Science Exam

Sequential search is an algorithm in which a data structure, such as an array or a list, not known to be sorted is searched element by element from beginning to end for a specific value.

On the AP Computer Science Exam, you must recognize the algorithm when it is presented in code. You should also be able to recognize a variation of the algorithm, such as a search from the end to the beginning. You may be expected to compare the efficiency of sequential search to (the much more efficient) binary search.

Assignment — Sequential-search implementation

Method `sequentialSearch` is to returns the index of `key` in `x` or -1 if `key` is not in `x`. Complete method `sequentialSearch` below.

```
public static int sequentialSearch(int[] x, int key)
```

5. Binary search practice problem for the AP Computer Science Exam

Binary search is an algorithm in which a data structure, such as an array or a list, known to be in sorted order is searched efficiently for a specific value. Since the data structure is known to be sorted the middle element is checked against the value sought. If the value matches, the index of the middle element is returned. If the value does not match, the half of the data structure that could contain the value is searched recursively. If the value sought is not found, a number indicating where the value should be inserted (to maintain the sorted order) is returned.

On the AP Computer Science Exam, you must recognize the algorithm when it is presented in code. You may be expected to compare the efficiency to sequential search as well as recognize the significant limitation of binary search (that the data structure must be sorted).

Assignment — Binary-search implementation

The precondition for method `binarySearch` is: `x` is sorted in increasing order. The method returns the index of `key` in`x`. If `key` is not in `x`, the method returns (-(*insertion point*) – 1) where *insertion point* is the index at which `key` would be inserted in `x` to maintain `x` in sorted order.

Complete method binarySearch below. You must declare and write a recursive helper method with an appropriate header.

```
public static int binarySearch(int[] x, int key)
```

1.

```
private static void sort(int[] a, int low, int high)
  {
    if (low < high)
    {
      int mid = (low + high) / 2;
      sort(a, low, mid);
      sort(a, mid + 1, high);
      merge(a, low, mid, high);
    }
  }
```

2.

```
public static void sort(int[] x)
{
   for (int i = 0; i < x.length - 1; i++)
   {
      int minIndex = i;

      for (int j = i + 1; j < x.length; j++)
         if (x[j] < x[minIndex]) minIndex = j;

      swap(x, i, minIndex);
   }
}

private static void swap(int[] x, int i, int j)
{
   int temp = x[i];
   x[i] = x[j];
   x[j] = temp;
}
```

Selection sort trace

```
[71, 86, 79, 36, 78, 35, 75, 86, 24, 11]
[11 | 86, 79, 36, 78, 35, 75, 86, 24, 71]
[11, 24 | 79, 36, 78, 35, 75, 86, 86, 71]
[11, 24, 35 | 36, 78, 79, 75, 86, 86, 71]
[11, 24, 35, 36 | 78, 79, 75, 86, 86, 71]
[11, 24, 35, 36, 71 | 79, 75, 86, 86, 78]
[11, 24, 35, 36, 71, 75 | 79, 86, 86, 78]
[11, 24, 35, 36, 71, 75, 78 | 86, 86, 79]
[11, 24, 35, 36, 71, 75, 78, 79 | 86, 86]
[11, 24, 35, 36, 71, 75, 78, 79, 86 | 86]
```

Note that once the elements from `x[0]` to `x[x.length-2]` have been placed in their final positions, `x[x.length-1]` must also be in its final position.

3.

```
public static void sort(int[] x)
{
  for (int i = 1; i < x.length; i++)
  {
    int next = x[i];
    int j = i;

    while (j > 0 && next < x[j - 1])
    {
      x[j] = x[j - 1];
      j--;
    }

    x[j] = next;
  }
}
```

Insertion sort trace

```
[37 | 32, 70, 15, 93, 40, 63, 3, 40, 63]
[32, 37 | 70, 15, 93, 40, 63, 3, 40, 63]
[32, 37, 70 | 15, 93, 40, 63, 3, 40, 63]
[15, 32, 37, 70 | 93, 40, 63, 3, 40, 63]
[15, 32, 37, 70, 93 | 40, 63, 3, 40, 63]
[15, 32, 37, 40, 70, 93 | 63, 3, 40, 63]
[15, 32, 37, 40, 63, 70, 93 | 3, 40, 63]
[3, 15, 32, 37, 40, 63, 70, 93 | 40, 63]
[3, 15, 32, 37, 40, 40, 63, 70, 93 | 63]
[3, 15, 32, 37, 40, 40, 63, 63, 70, 93]
```

Before the first step, the first element is already part of the sorted part. Instead of placing elements into their final positions, as selection sort does, insertion sort shifts one element at a time into its proper position in the sorted part. No element is guaranteed to be in its final position until the method finishes.

4.

```
public static int sequentialSearch(int[] x, int key)
{
 for (int i = 0; i < x.length; i++)
        if (x[i] == key) return i;
 return -1;
}
```

5.

```
public static int binarySearch(int[] x, int key)
{
  return binarySearch(x, key, 0, x.length - 1);
}

private static int binarySearch(int[] x, int key, int start, int end)
{
  if (start > end)
    return -start - 1;
  else
  {
    int mid = (start + end) / 2;
    int result = key - x[mid];

    if (result < 0)
    { return binarySearch(x, key, start, mid - 1); }
    else
    {
      if(result > 0)
        return binarySearch(x, key, mid + 1, end);
      else return mid;
    }
  }
}
```