MC Answers on Page 20

THE EXAM

The AP Computer Science A Exam is 3 hours long and seeks to determine how well students have mastered the concepts and techniques contained in the course outline. The exam consists of two sections: a multiple-choice section (40 questions in 1 hour and 30 minutes), which tests proficiency in a wide variety of topics, and a free-response section (4 questions in 1 hour and 30 minutes), which requires the student to demonstrate the ability to solve problems involving more extended reasoning.

The multiple-choice and the free-response sections of the AP Computer Science A Exam require students to demonstrate their ability to solve problems, including their ability to design, write, and analyze programs and subprograms. Minor points of syntax are not tested on the exam. All code given is consistent with the AP Java subset. All student responses involving code must be written in Java. Students are expected to be familiar with and able to use the standard Java classes and interfaces listed in the AP Java subset. For both the multiple-choice and the free-response sections of the exam, a quick reference to the classes and interfaces in the AP Java subset will be provided. The Java Quick Reference is included in Appendix B.

In the determination of the grade for the exam, the multiple-choice section and the free-response section are given equal weight. Because the exam is designed for full coverage of the subject matter, it is not expected that many students will be able to correctly answer all the questions in either the multiple-choice section or the free-response section in the time allotted.

Multiple-choice questions on the exam are classified according to the type of content that is tested in the question. Questions may be listed in one or more of the classification categories. For example, a question that uses a looping construct to traverse the elements of an array would be listed under both the Data Structures and the Programming Fundamentals categories. The table below shows the classification categories and how they are represented in the multiple-choice section of the exam. Because questions can be classified in more than one category, the total of the percentages is greater than 100%.

Classification Category	Percent of multiple-choice items
Programming Fundamentals	55–75%
Data Structures	20–40%
Logic	5–15%
Algorithms/Problem Solving	25–45%
Object-Oriented Programming	15–25%
Recursion	5–15%
Software Engineering	2–10%

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

Computer Science A: Sample Multiple-Choice Questions

Following is a representative set of questions. The answer key for the Computer Science A multiple-choice questions is on page 43. Multiple-choice scores are based on the number of questions answered correctly. Points are not deducted for incorrect answers, and no points are awarded for unanswered questions. Because points are not deducted for incorrect answers, students are encouraged to answer all multiple-choice questions. Students should eliminate as many choices as they can on any questions for which they do not know the answer, and then select the best answer among the remaining choices.

Directions: Determine the answer to each of the following questions or incomplete statements, using the available space for any necessary scratch work. Then decide which is the best of the choices given and fill in the corresponding circle on the answer sheet. No credit will be given for anything written in the examination booklet. Do not spend too much time on any one problem

Notes:

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Assume that declarations of variables and methods appear within the context of an enclosing class.
- Assume that method calls that are not prefixed with an object or class name and are not shown within a complete class definition appear within the context of an enclosing class.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

1. Consider the following code segment.

```
for (int k = 0; k < 20; k = k + 2)
{
  if (k % 3 == 1)
  {
    System.out.print(k + "  ");
  }
}
```

What is printed as a result of executing the code segment?

(A) `4  16`

(B) `4  10  16`

(C) `0  6  12  18`

(D) `1  4  7  10  13  16  19`

(E) `0  2  4  6  8  10  12  14  16  18`

2. Consider the following code segment.

```
List<String> animals = new ArrayList<String>();

animals.add("dog");
animals.add("cat");
animals.add("snake");
animals.set(2, "lizard");
animals.add(1, "fish");
animals.remove(3);
System.out.println(animals);
```

What is printed as a result of executing the code segment?

(A) `[dog, fish, cat]`

(B) `[dog, fish, lizard]`

(C) `[dog, lizard, fish]`

(D) `[fish, dog, cat]`

(E) The code throws an `ArrayIndexOutOfBoundsException` exception.

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

3. Consider the following method.

```
public static void mystery(List<Integer> nums)
{
  for (int k = 0; k < nums.size(); k++)
  {
    if (nums.get(k).intValue() == 0)
    {
       nums.remove(k);
    }
  }
}
```

Assume that a `List<Integer> values` initially contains the following `Integer` values.

```
[0, 0, 4, 2, 5, 0, 3, 0]
```

What will `values` contain as a result of executing `mystery(values)`?

(A) `[0, 0, 4, 2, 5, 0, 3, 0]`

(B) `[4, 2, 5, 3]`

(C) `[0, 0, 0, 0, 4, 2, 5, 3]`

(D) `[0, 4, 2, 5, 3]`

(E) The code throws an `ArrayIndexOutOfBoundsException` exception.

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

4. At a certain high school students receive letter grades based on the following scale.

Integer Score	Letter Grade
93 or above	A
From 84 to 92 inclusive	B
From 75 to 83 inclusive	C
Below 75	F

Which of the following code segments will assign the correct string to `grade` for a given integer `score` ?

I.
```
if (score >= 93)
  grade = "A";
if (score >= 84 && score <= 92)
  grade = "B";
if (score >= 75 && score <= 83)
  grade = "C";
if (score < 75)
  grade = "F";
```

II.
```
if (score >= 93)
  grade = "A";
if (84 <= score <= 92)
  grade = "B";
if (75 <= score <= 83)
  grade = "C";
if (score < 75)
  grade = "F";
```

III.
```
if (score >= 93)
  grade = "A";
else if (score >= 84)
  grade = "B";
else if (score >= 75)
  grade = "C";
else
  grade = "F";
```

(A) II only

(B) III only

(C) I and II only

(D) I and III only

(E) I, II, and III

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

5. Consider the following output.

```
1  1  1  1  1
2  2  2  2
3  3  3
4  4
5
```

Which of the following code segments will produce this output?

(A)
```
for (int j = 1; j <= 5; j++)
{
  for (int k = 1; k <= 5; k++)
  {
    System.out.print(j + "  ");
  }
  System.out.println();
}
```

(B)
```
for (int j = 1; j <= 5; j++)
{
  for (int k = 1; k <= j; k++)
  {
    System.out.print(j + "  ");
  }
  System.out.println();
}
```

(C)
```
for (int j = 1; j <= 5; j++)
{
  for (int k = 5; k >= 1; k--)
  {
    System.out.print(j + "  ");
  }
  System.out.println();
}
```

(D)
```
for (int j = 1; j <= 5; j++)
{
  for (int k = 5; k >= j; k--)
  {
    System.out.print(j + "  ");
  }
  System.out.println();
}
```

(E)
```
for (int j = 1; j <= 5; j++)
{
  for (int k = j; k <= 5; k++)
  {
    System.out.print(k + "  ");
  }
  System.out.println();
}
```


© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

10. Consider the following instance variable and method.

```
private int[] arr;

/** Precondition: arr contains no duplicates;
 *                the elements in arr are in ascending order.
 *  @param low an int value such that 0 ≤ low ≤ arr.length
 *  @param high an int value such that low - 1 ≤ high < arr.length
 *  @param num an int value
 */
public int mystery(int low, int high, int num)
{
  int mid = (low + high) / 2;
  if (low > high)
  {
    return low;
  }
  else if (arr[mid] < num)
  {
    return mystery(mid + 1, high, num);
  }
  else if (arr[mid] > num)
  {
    return mystery(low, mid - 1, num);
  }
  else // arr[mid] == num
  {
    return mid;
  }
}
```

What is returned by the call `mystery(0, arr.length - 1, num)`?

(A) The number of elements in `arr` that are less than `num`

(B) The number of elements in `arr` that are less than or equal to `num`

(C) The number of elements in `arr` that are equal to `num`

(D) The number of elements in `arr` that are greater than `num`

(E) The index of the middle element in `arr`

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

Questions 11–12 refer to the following information.

Consider the following instance variable `nums` and method `findLongest` with line numbers added for reference. Method `findLongest` is intended to find the longest consecutive block of the value `target` occurring in the array `nums`; however, `findLongest` does not work as intended.

For example, if the array `nums` contains the values [7, 10, 10, 15, 15, 15, 15, 10, 10, 10, 15, 10, 10], the call `findLongest(10)` should return 3, the length of the longest consecutive block of 10s.

```
private int[] nums;

public int findLongest(int target)
{
    int lenCount = 0;
    int maxLen = 0;

Line 1:  for (int val : nums)
Line 2:  {
Line 3:    if (val == target)
Line 4:    {
Line 5:      lenCount++;
Line 6:    }
Line 7:    else
Line 8:    {
Line 9:      if (lenCount > maxLen)
Line 10:     {
Line 11:       maxLen = lenCount;
Line 12:     }
Line 13:   }
Line 14: }
Line 15: if (lenCount > maxLen)
Line 16: {
Line 17:   maxLen = lenCount;
Line 18: }
Line 19: return maxLen;
    }
```

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

11. The method `findLongest` does not work as intended. Which of the following best describes the value returned by a call to `findLongest ?`

    (A) It is the length of the shortest consecutive block of the value `target` in `nums`.

    (B) It is the length of the array `nums`.

    (C) It is the number of occurrences of the value `target` in `nums`.

    (D) It is the length of the first consecutive block of the value `target` in `nums`.

    (E) It is the length of the last consecutive block of the value `target` in `nums`.

12. Which of the following changes should be made so that method `findLongest` will work as intended?

    (A) Insert the statement `lenCount = 0;` between lines 2 and 3.

    (B) Insert the statement `lenCount = 0;` between lines 8 and 9.

    (C) Insert the statement `lenCount = 0;` between lines 10 and 11.

    (D) Insert the statement `lenCount = 0;` between lines 11 and 12.

    (E) Insert the statement `lenCount = 0;` between lines 12 and 13.

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

13. Consider the following instance variable and method.

```
private int[] numbers;

/** Precondition: numbers contains int values in no particular order.
 */
public int mystery(int num)
{
  for (int k = numbers.length - 1; k >= 0; k--)
  {
    if (numbers[k] < num)
    {
      return k;
    }
  }
  return -1;
}
```

Which of the following best describes the contents of `numbers` after the following statement has been executed?

```
int m = mystery(n);
```

(A) All values in positions `0` through `m` are less than `n`.

(B) All values in positions `m+1` through `numbers.length-1` are less than `n`.

(C) All values in positions `m+1` through `numbers.length-1` are greater than or equal to `n`.

(D) The smallest value is at position `m`.

(E) The largest value that is smaller than `n` is at position `m`.

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

14. Consider the following method.

```
/** @param x an int value such that x >= 0
 */
public void mystery(int x)
{
  System.out.print(x % 10);
  if ((x / 10) != 0)
  {
    mystery(x / 10);
  }
  System.out.print(x % 10);
}
```

Which of the following is printed as a result of the call `mystery(1234)`?

(A) `1234`

(B) `4321`

(C) `12344321`

(D) `43211234`

(E) Many digits are printed due to infinite recursion.

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

16. Consider the following recursive method.

```
public static int mystery(int n)
{
  if (n <= 1)
  {
    return 0;
  }
  else
  {
    return 1 + mystery(n / 2);
  }
}
```

Assuming that k is a nonnegative integer and $m = 2^k$, what value is returned as a result of the call `mystery(`m`)` ?

(A) 0

(B) k

(C) m

(D) $\frac{m}{2} + 1$

(E) $\frac{k}{2} + 1$

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

17. Consider the following instance variable and method.

```
private int[] array;

/** Precondition: array.length > 0
 */
public int checkArray()
{
  int loc = array.length / 2;
  for (int k = 0; k < array.length; k++)
  {
    if (array[k] > array[loc])
    {
      loc = k;
    }
  }
  return loc;
}
```

Which of the following is the best postcondition for `checkArray` ?

(A) Returns the index of the first element in array `array` whose value is greater than `array[loc]`

(B) Returns the index of the last element in array `array` whose value is greater than `array[loc]`

(C) Returns the largest value in array `array`

(D) Returns the index of the largest value in array `array`

(E) Returns the index of the largest value in the second half of array `array`

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

18. Consider the following methods.

```
public void changer(String x, int y)
{
  x = x + "peace";
  y = y * 2;
}

public void test()
{
  String s = "world";
  int n = 6;
  changer(s, n);

  /* End of method */
 }
```

When the call `test()` is executed, what are the values of `s` and `n` at the point indicated by `/* End of method */` ?

	s	n
(A)	world	6
(B)	worldpeace	6
(C)	world	12
(D)	worldpeace	12
(E)	peace	12

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

20. Consider the following method.

```
/** Precondition: arr contains only positive values.
 */
public static void doSome(int[] arr, int lim)
{
  int v = 0;
  int k = 0;
  while (k < arr.length && arr[k] < lim)
  {
    if (arr[k] > v)
    {
      v = arr[k]; /* Statement S */
    }
    k++; /* Statement T */
  }
}
```

Assume that `doSome` is called and executes without error. Which of the following are possible combinations for the value of `lim`, the number of times *Statement S* is executed, and the number of times *Statement T* is executed?

	Value of `lim`	Executions of *Statement S*	Executions of *Statement T*
I.	5	0	5
II.	7	4	9
III.	3	5	2

(A) I only

(B) II only

(C) III only

(D) I and III only

(E) II and III only

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

21. Consider the following instance variable, `arr`, and incomplete method, `partialSum`. The method is intended to return an integer array `sum` such that for all `k`, `sum[k]` is equal to `arr[0] + arr[1] + ... + arr[k]`. For instance, if `arr` contains the values `{ 1, 4, 1, 3 }`, the array `sum` will contain the values `{ 1, 5, 6, 9 }`.

```
private int[] arr;
public int[] partialSum()
{
  int[] sum = new int[arr.length];
  for (int j = 0; j < sum.length; j++)
  {
    sum[j] = 0;
  }
  /* missing code */
  return sum;
}
```

The following two implementations of `/* missing code */` are proposed so that `partialSum` will work as intended.

Implementation 1

```
for (int j = 0; j < arr.length; j++)
{
  sum[j] = sum[j - 1] + arr[j];
}
```

Implementation 2

```
for (int j = 0; j < arr.length; j++)
{
  for (int k = 0; k <= j; k++)
  {
    sum[j] = sum[j] + arr[k];
  }
}
```

Which of the following statements is true?

(A) Both implementations work as intended, but implementation 1 is faster than implementation 2.

(B) Both implementations work as intended, but implementation 2 is faster than implementation 1.

(C) Both implementations work as intended and are equally fast.

(D) Implementation 1 does not work as intended, because it will cause an `ArrayIndexOutOfBoundsException`.

(E) Implementation 2 does not work as intended, because it will cause an `ArrayIndexOutOfBoundsException`.

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

Consider the following proposed constructors for this class.

I.
```
public NamedPoint()
{
  name = "";
}
```

II.
```
public NamedPoint(int d1, int d2, String pointName)
{
  x = d1;
  y = d2;
  name = pointName;
}
```

III.
```
public NamedPoint(int d1, int d2, String pointName)
{
  super(d1, d2);
  name = pointName;
}
```

Which of these constructors would be legal for the `NamedPoint` class?

(A) I only

(B) II only

(C) III only

(D) I and III only

(E) II and III only

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

23. Consider a `shuffle` method that is intended to return a new array that contains all the elements from `nums`, but in a different order. Let `n` be the number of elements in `nums`. The `shuffle` method should alternate the elements from `nums[0]` … `nums[n / 2 - 1]` with the elements from `nums[n / 2]` … `nums[n - 1]`, as illustrated in the following examples.

Example 1

	0	1	2	3	4	5	6	7
`nums`	10	20	30	40	50	60	70	80

	0	1	2	3	4	5	6	7
`result`	10	50	20	60	30	70	40	80

Example 2

	0	1	2	3	4	5	6
`nums`	10	20	30	40	50	60	70

	0	1	2	3	4	5	6
`result`	10	40	20	50	30	60	70

The following implementation of the `shuffle` method does not work as intended.

```
public static int[] shuffle(int[] nums)
{
  int n = nums.length;
  int[] result = new int[n];

  for (int j = 0; j < n / 2; j++)
  {
    result[j * 2] = nums[j];
    result[j * 2 + 1] = nums[j + n / 2];
  }

  return result;
}
```

Which of the following best describes the problem with the given implementation of the `shuffle` method?

(A) Executing `shuffle` may cause an `ArrayIndexOutOfBoundsException`.

(B) The first element of the returned array (`result[0]`) may not have the correct value.

(C) The last element of the returned array (`result[result.length - 1]`) may not have the correct value.

(D) One or more of `nums[0]` … `nums[nums.length / 2 - 1]` may have been copied to the wrong position(s) in the returned array.

(E) One or more of `nums[nums.length / 2]` … `nums[nums.length - 1]` may have been copied to the wrong position(s) in the returned array.

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

25. The following `sort` method correctly sorts the integers in `elements` into ascending order.

```
Line 1:  public static void sort(int[] elements)
Line 2:  {
Line 3:    for (int j = 0; j < elements.length - 1; j++)
Line 4:    {
Line 5.      int index = j;
Line 6:
Line 7:      for (int k = j + 1; k < elements.length; k++)
Line 8:      {
Line 9:        if (elements[k] < elements[index])
Line 10:       {
Line 11:         index = k;
Line 12:       }
Line 13:     }
Line 14:
Line 15:     int temp = elements[j];
Line 16:     elements[j] = elements[index];
Line 17:     elements[index] = temp;
Line 18:   }
Line 19: }
```


© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

Which of the following changes to the `sort` method would correctly sort the integers in `elements` into **descending** order?

I. Replace line 9 with:
Line 9: `if (elements[k] > elements[index])`

II. Replace lines 15–17 with:
Line 15: `int temp = elements[index];`
Line 16: `elements[index] = elements[j];`
Line 17: `elements[j] = temp;`

III. Replace line 3 with:
Line 3: `for (int j = elements.length − 1; j > 0; j−−)`
and replace line 7 with:
Line 7: `for (int k = 0; k < j; k++)`

(A) I only

(B) II only

(C) I and II only

(D) I and III only

(E) I, II, and III

Answers to Computer Science A Multiple-Choice Questions

1 – B	6 – A	11 – C	16 – B	21 – D
2 – A	7 – A	12 – E	17 – D	22 – D
3 – D	8 – C	13 – C	18 – A	23 – C
4 – D	9 – D	14 – D	19 – D	24 – E
5 – D	10 – A	15 – D	20 – B	25 – D

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

Sample Free-Response Questions

Following is a representative set of questions. Additional sample questions can be found in the AP section of the College Board website.

Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

answers on Page 26-28

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

2. Consider the following incomplete `StringUtil` class declaration. You will write implementations for the two methods listed in this class. Information about the `Person` class used in the `replaceNameNickname` method will be presented in part (b).

```
public class StringUtil
{
  /** @param str a String with length > 0
   *  @param oldstr a String
   *  @param newstr a String
   *  @return a new String in which all occurrences of the substring
   *            oldstr in str are replaced by the substring newstr
   */
  public static String apcsReplaceAll(String str,
                                      String oldStr,
                                      String newStr)
  {  /* to be implemented in part (a) */  }
```

(a) Write the `StringUtil` method `apcsReplaceAll`, which examines a given `String` and replaces all occurrences of a designated substring with another specified substring. In writing your solution, you may NOT use the `replace`, `replaceAll`, or `replaceFirst` methods in the Java `String` class.

The following table shows several examples of the result of calling `StringUtil.apcsReplaceAll(str, oldstr,0 newstr)`.

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

str	oldstr	newstr	String returned	Comment
`"to be or not to be"`	`"to"`	`"2"`	`"2 be or not 2 be"`	Each occurrence of `"to"` in the original string has been replaced by "2"
`"advanced calculus"`	`"math"`	`"science"`	`"advanced calculus"`	No change, because the string `"math"` was not in the original string
`"gogogo"`	`"go"`	`"gone"`	`"gonegonegone"`	Each occurrence of `"go"` in the original string has been replaced by `"gone"`
`"aaaaa"`	`"aaa"`	`"b"`	`"baa"`	The first occurrence of `"aaa"` in the original string has been replaced by `"b"`

Complete method `apcsReplaceAll` below.

```
/** @param str a String with length > 0
 *  @param oldstr a String
 *  @param newstr a String
 *  @ return a new String in which all occurrences of the substring
 *            oldstr in str are replaced by the substring newstr
 */
public static String apcsReplaceAll(String str,
                                    String oldStr,
                                    String newStr)
```


© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

4. This question involves manipulation of one-dimensional and two-dimensional arrays. In part (a), you will write a method to shift the elements of a one-dimensional array. In parts (b) and (c), you will write methods to shift the elements of a two-dimensional array.

   (a) Consider the following incomplete `ArrayUtil` class, which contains a `static shiftArray` method.

```
public class ArrayUtil
{
  /** Shifts each array element to the next higher index, discarding the
   *  original last element, and inserts the new number at the front.
   *  @param arr the array to manipulate
   *         Precondition: arr.length > 0
   *  @param num the new number to insert at the front of arr
   *  Postcondition: The original elements of arr have been shifted to
   *                 the next higher index, and arr[0] == num.
   *                 The original element at the highest index has been
   *                 discarded.
   */
  public static void shiftArray(int[] arr, int num)
  {  /* to be implemented in part (a) */  }

  // There may be methods that are not shown.
}
```

   Write the `ArrayUtil` method `shiftArray`. This method stores the integer `num` at the front of the array `arr` after shifting each of the original elements to the position with the next higher index. The element originally at the highest index is lost.

   For example, if `arr` is the array `{11, 12, 13, 14, 15}` and `num` is 27, the call to `shiftArray` changes `arr` as shown below.

Before call	0	1	2	3	4
arr:	11	12	13	14	15

After call	0	1	2	3	4
arr:	27	11	12	13	14

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

Complete method `shiftArray` below.

```
/** Shifts each array element to the next higher index, discarding the
 *  original last element, and inserts the new number at the front.
 *  @param arr the array to manipulate
 *             Precondition: arr.length > 0
 *  @Param num the new number to insert at the front of arr
 *  Postcondition: The original elements of arr have been shifted to
 *                 the next higher index, and arr[0] == num.
 *                 The original element at the highest index has been
 *                 discarded.
 */
public static void shiftArray(int[] arr, int num)
```


© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

Suggested Solutions to Free-Response Questions

Note: There are many correct variations of these solutions.

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

Question 2

(a)
Iterative version:

```
public static String apcsReplaceAll(String str,
                                    String oldStr,
                                    String newStr)
{
  String firstPart = "";
  String lastPart = str;
  int pos = lastPart.indexOf(oldStr);
  while (pos >= 0)
  {
    firstPart += lastPart.substring(0, pos);
    firstPart += newStr;
    lastPart = lastPart.substring(pos + oldStr.length());
    pos = lastPart.indexOf(oldStr);
  }
  return firstPart + lastPart;
}
```

Recursive version:

```
public static String apcsReplaceAll(String str,
                                    String oldStr,
                                    String newStr)
{
  int pos = str.indexOf(oldStr);
  if (pos < 0)
  {
    return str;
  }
  else
  {
    String firstPart = str.substring(0, pos);
    String restOfStr = str.substring(pos + oldStr.length());
    String lastPart = apcsReplaceAll(restOfStr, oldStr, newStr);
    return firstPart + newStr + lastPart;
  }
}
```


© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

```
}
```

Question 4

(a)

```
public static void shiftArray(int[] arr, int num)
{
  for (int k = arr.length − 1; k > 0; k−−)
  {
    arr[k] = arr[k - 1];
  }

  arr[0] = num;
}
```


© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

APPENDIX B

Exam Appendix — Java Quick Reference

Accessible methods from the Java library that may be included on the exam

class java.lang.Object

- `boolean equals(Object other)`
- `String toString()`

class java.lang.Integer

- `Integer(int value)`
- `int intValue()`
- `Integer.MIN_VALUE` // minimum value represented by an `int` or `Integer`
- `Integer.MAX_VALUE` // maximum value represented by an `int` or `Integer`

class java.lang.Double

- `Double(double value)`
- `double doubleValue()`

class java.lang.String

- `int length()`
- `String substring(int from, int to)` // returns the substring beginning at `from`
  // and ending at `to-1`
- `String substring(int from)` // returns `substring(from, length())`
- `int indexOf(String str)` // returns the index of the first occurrence of `str`;
  // returns `-1` if not found
- `int compareTo(String other)` // returns a value < 0 if `this` is less than `other`
  // returns a value = 0 if `this` is equal to `other`
  // returns a value > 0 if `this` is greater than `other`

class java.lang.Math

- `static int abs(int x)`
- `static double abs(double x)`
- `static double pow(double base, double exponent)`
- `static double sqrt(double x)`
- `static double random()` // returns a `double` in the range [0.0, 1.0)

interface java.util.List<E>

- `int size()`
- `boolean add(E obj)` // appends `obj` to end of list; returns `true`
- `void add(int index, E obj)` // inserts `obj` at position `index` ($0 \le \texttt{index} \le \text{size}$),
  // moving elements at position `index` and higher
  // to the right (adds 1 to their indices) and adjusts size
- `E get(int index)`
- `E set(int index, E obj)` // replaces the element at position `index` with `obj`
  // returns the element formerly at the specified position
- `E remove(int index)` // removes element from position `index`, moving elements
  // at position `index + 1` and higher to the left
  // (subtracts 1 from their indices) and adjusts size
  // returns the element formerly at the specified position

class java.util.ArrayList<E> implements java.util.List<E>

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

APPENDIX C — SAMPLE SEARCH AND SORT ALGORITHMS

Sequential Search

The Sequential Search Algorithm below finds the index of a value in an array of integers as follows:

1. Traverse `elements` until `target` is located, or the end of `elements` is reached.
2. If `target` is located, return the index of `target` in `elements`; Otherwise return `-1`.

```
/**
 * Finds the index of a value in an array of integers.
 *
 * @param elements an array containing the items to be searched.
 * @param target the item to be found in elements.
 * @return an index of target in elements if found; -1 otherwise.
 */
public static int sequentialSearch(int[] elements, int target)
{
  for (int j = 0; j < elements.length; j++)
  {
    if (elements[j] == target)
    {
      return j;
    }
  }

  return -1;
}
```


© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

Binary Search

The Binary Search Algorithm below finds the index of a value in an array of integers sorted in ascending order as follows:

1. Set `left` and `right` to the minimum and maximum indexes of `elements` respectively.
2. Loop until `target` is found, or `target` is determined not to be in `elements` by doing the following for each iteration:
   a. Set `middle` to the index of the middle item in `elements[left] ... elements[right]` inclusive.
   b. If `target` would have to be in `elements[left] ... elements[middle - 1]` inclusive, then set `right` to the maximum index for that range.
   c. Otherwise, if `target` would have to be in `elements[middle + 1] ... elements[right]` inclusive, then set `left` to the minimum index for that range.
   d. Otherwise, return `middle` because `target == elements[middle]`.
3. Return `-1` if `target` is not contained in `elements`.

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

```
/**
 * Find the index of a value in an array of integers sorted in ascending order.
 *
 * @param elements an array containing the items to be searched.
 *                 Precondition: items in elements are sorted in ascending order.
 * @param target the item to be found in elements.
 * @return an index of target in elements if target found;
 *         -1 otherwise.
 */
public static int binarySearch(int[] elements, int target)
{
  int left = 0;
  int right = elements.length - 1;

  while (left <= right)
  {
    int middle = (left + right) / 2;
    if (target < elements[middle])
    {
      right = middle - 1;
    }
    else if (target > elements[middle])
    {
      left = middle + 1;
    }
    else
    {
      return middle;
    }
  }

  return -1;
}
```


© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

Selection Sort

The Selection Sort Algorithm below sorts an array of integers into ascending order as follows:

1. Loop from `j = 0` to `j = elements.length−2`, inclusive, completing `elements.length−1` passes.
2. In each pass, swap the item at index `j` with the minimum item in the rest of the array (`elements[j+1]` through `elements[elements.length−1]`).

At the end of each pass, items in `elements[0]` through `elements[j]` are in ascending order and each item in this sorted portion is at its final position in the array

```
/**
 * Sort an array of integers into ascending order.
 *
 * @param elements an array containing the items to be sorted.
 *
 * Postcondition: elements contains its original items and items in elements
 *                are sorted in ascending order.
 */
public static void selectionSort(int[] elements)
{
  for (int j = 0; j < elements.length − 1; j++)
  {
     int minIndex = j;

     for (int k = j + 1; k < elements.length; k++)
     {
       if (elements[k] < elements[minIndex])
       {
          minIndex = k;
       }
     }

     int temp = elements[j];
     elements[j] = elements[minIndex];
     elements[minIndex] = temp;
  }
}
```

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

Insertion Sort

The Insertion Sort Algorithm below sorts an array of integers into ascending order as follows:

1. Loop from `j = 1` to `j = elements.length−1` inclusive, completing `elements.length−1` passes.
2. In each pass, move the item at index `j` to its proper position in `elements[0]` to `elements[j]`:
   a. Copy item at index `j` to `temp`, creating a "vacant" element at index `j` (denoted by `possibleIndex`).
   b. Loop until the proper position to maintain ascending order is found for `temp`.
   c. In each inner loop iteration, move the "vacant" element one position lower in the array.
3. Copy `temp` into the identified correct position (at `possibleIndex`).

At the end of each pass, items at `elements[0]` through `elements[j]` are in ascending order.

```
/**
 * Sort an array of integers into ascending order.
 *
 * @param elements an array containing the items to be sorted.
 *
 * Postcondition: elements contains its original items and items in elements
 *                are sorted in ascending order.
 */
public static void insertionSort(int[] elements)
{
  for (int j = 1; j < elements.length; j++)
  {
    int temp = elements[j];
    int possibleIndex = j;
    while (possibleIndex > 0 && temp < elements[possibleIndex - 1])
    {
      elements[possibleIndex] = elements[possibleIndex - 1];
      possibleIndex--;
    }

    elements[possibleIndex] = temp;
  }
}
```


© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

Merge Sort

The Merge Sort Algorithm below sorts an array of integers into ascending order as follows:

`mergeSort`

This top-level method creates the necessary temporary array and calls the `mergeSortHelper` recursive helper method.

`mergeSortHelper`

This recursive helper method uses the Merge Sort Algorithm to sort `elements[from]` ... `elements[to]` inclusive into ascending order:

1. If there is more than one item in this range,
   a. divide the items into two adjacent parts, and
   b. call `mergeSortHelper` to recursively sort each part, and
   c. call the `merge` helper method to merge the two parts into sorted order.
2. Otherwise, exit because these items are sorted.

`merge`

This helper method merges two adjacent array parts, each of which has been sorted into ascending order, into one array part that is sorted into ascending order:

1. As long as both array parts have at least one item that hasn't been copied, compare the first un-copied item in each part and copy the minimal item to the next position in `temp`.
2. Copy any remaining items of the first part to `temp`.
3. Copy any remaining items of the second part to `temp`.
4. Copy the items from `temp[from]` ... `temp[to]` inclusive to the respective locations in `elements`.

```
/**
 * Sort an array of integers into ascending order.
 *
 * @param elements an array containing the items to be sorted.
 *
 * Postcondition: elements contains its original items and items in elements
 *                are sorted in ascending order.
 */
public static void mergeSort(int[] elements)
{
  int n = elements.length;
  int[] temp = new int[n];
  mergeSortHelper(elements, 0, n - 1, temp);
}
```

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

```
/**
 * Sorts elements[from] ... elements[to] inclusive into ascending order.
 *
 * @param elements an array containing the items to be sorted.
 * @param from the beginning index of the items in elements to be sorted.
 * @param to the ending index of the items in elements to be sorted.
 * @param temp a temporary array to use during the merge process.
 *
 * Precondition:
 *     (elements.length == 0 or
 *      0 <= from <= to <= elements.length) and
 *     elements.length == temp.length
 * Postcondition: elements contains its original items and the items in elements
 *                [from] ... <= elements[to] are sorted in ascending order.
 */
private static void mergeSortHelper(int[] elements,
                                    int from, int to, int[] temp)
{
  if (from < to)
  {
    int middle = (from + to) / 2;
    mergeSortHelper(elements, from, middle, temp);
    mergeSortHelper(elements, middle + 1, to, temp);
    merge(elements, from, middle, to, temp);
  }
}
```


© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

```
/**
 * Merges two adjacent array parts, each of which has been sorted into ascending
 * order, into one array part that is sorted into ascending order.
 *
 * @param elements an array containing the parts to be merged.
 * @param from the beginning index in elements of the first part.
 * @param mid the ending index in elements of the first part.
 *        mid+1 is the beginning index in elements of the second part.
 * @param to the ending index in elements of the second part.
 * @param temp a temporary array to use during the merge process.
 *
 * Precondition: 0 <= from <= mid <= to <= elements.length and
 *   elements[from] ... <= elements[mid] are sorted in ascending order and
 *   elements[mid + 1] ... <= elements[to] are sorted in ascending order and
 *   elements.length == temp.length
 * Postcondition: elements contains its original items and
 *   elements[from] ... <= elements[to] are sorted in ascending order and
 *   elements[0] ... elements[from - 1] are in original order and
 *   elements[to + 1] ... elements[elements.length - 1] are in original order.
 */
private static void merge(int[] elements,
                          int from, int mid, int to, int[] temp)
{
  int i = from;
  int j = mid + 1;
  int k = from;

  while (i <= mid && j <= to)
  {
    if (elements[i] < elements[j])
    {
      temp[k] = elements[i];
      i++;
    }
    else
    {
      temp[k] = elements[j];
      j++;
    }
    k++;
  }
```

© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.

```
  while (i <= mid)
  {
    temp[k] = elements[i];
    i++;
    k++;
  }

  while (j <= to)
  {
    temp[k] = elements[j];
    j++;
    k++;
  }

  for (k = from; k <= to; k++)
  {
    elements[k] = temp[k];
  }
}
```


© 2014 The College Board. Visit the College Board on the Web: www.collegeboard.org.