



AP[®] Computer Science A

Practice Exam

The questions contained in this AP[®] Computer Science A Practice Exam were written to the content specifications of AP Exams for this subject. Because this practice exam has never been given as an actual AP Exam, statistical data are not available for calculating potential raw scores or conversions into AP scores. Taking this AP Computer Science A Practice Exam is meant to provide students preparing for the AP Exam with an overall sense of their strengths as well as areas where they need to improve.

This AP Computer Science A Practice Exam is provided by the College Board for AP Exam preparation. Teachers are permitted to download the materials and make copies to use with their students in a classroom setting only. To maintain the security of this exam, teachers should collect all materials after their administration and keep them in a secure location. Teachers may not redistribute the files electronically for any reason.

Contents

Directions for Administration.....	ii
Section I: Multiple-Choice Questions.....	1
Section II: Free-Response Questions	42
Student Answer Sheet for Multiple-Choice Section	56
Multiple-Choice Answer Key.....	57
Free-Response Scoring Guidelines.....	58

AP[®] Computer Science A

Directions for Administration

The AP Computer Science A Exam is three hours in length and consists of a multiple-choice section and a free-response section.

- The 1 hour and 30-minute multiple-choice section contains 40 questions and accounts for 50 percent of the final grade.
- The 1 hour and 30-minute free-response section contains 4 questions and accounts for 50 percent of the final grade.

A 10-minute break should be provided after Section I is completed. Students should be given a 10-minute warning prior to the end of Section II.

The actual AP Exam is administered in one session. Students will have the most realistic experience if a complete morning or afternoon is available to administer this practice exam. If a schedule does not permit one time period for the entire practice exam administration, it would be acceptable to administer Section I one day and Section II on a subsequent day.

The total score on the multiple-choice section is based only on the number of questions answered correctly. Points are not deducted for incorrect answers or unanswered questions.

- The use of calculators, or any other electronic devices, is not permitted during the exam.
- It is suggested that the practice exam be completed using a pencil to simulate an actual administration.
- Teachers will need to provide paper for the students to write their free-response answers. Teachers should provide directions to the students indicating how they wish the responses to be labeled so the teacher will be able to associate the student's response with the question the student intended to answer.
- The AP Computer Science A Exam includes a Quick Reference to the Java Library methods that may be tested on the exam. The Java Quick Reference appears at the end of each section of the exam.
- Remember that students are not allowed to remove any materials, including scratch work, from the testing site.

Section I

Multiple-Choice Questions

COMPUTER SCIENCE A

SECTION I

Time—1 hour and 30 minutes

Number of questions—40

Percent of total grade—50

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 box on the student 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.

1. Consider the following method.

```
public static int mystery(int[] arr)
{
    int x = 0;

    for (int k = 0; k < arr.length; k = k + 2)
        x = x + arr[k];

    return x;
}
```

Assume that the array `nums` has been declared and initialized as follows.

```
int[] nums = {3, 6, 1, 0, 1, 4, 2};
```

What value will be returned as a result of the call `mystery(nums)` ?

(A) 5

(B) 6

(C) 7

(D) 10

(E) 17

ok starting out strong I'm feeling confident

Questions 2-3 refer to the following information.

Consider the following partial class declaration.

```
public class SomeClass
{
    private int myA;
    private int myB;
    private int myC;

    // Constructor(s) not shown

    public int getA()
    { return myA; }

    public void setB(int value)
    { myB = value; }
}
```

2. The following declaration appears in another class.

```
SomeClass obj = new SomeClass();
```

Which of the following code segments will compile without error? said out loud: ohhhh fuck

- (A) `int x = obj.getA();`
- (B) `int x;`
`obj.getA(x);` getA() is always called without an arg so it narrows it down to A and D
- (C) `int x = obj.myA;`
- (D) `int x = SomeClass.getA();` obj is the name of the variable so it's got to be A
- (E) `int x = getA(obj);`

3. Which of the following changes to `SomeClass` will allow other classes to access but not modify the value of `myC` ?

(A) Make `myC` public.

(B) Include the method:
`public int getC()`
`{ return myC; }`

(C) Include the method:
`private int getC()`
`{ return myC; }`

(D) Include the method:
`public void getC(int x)`
`{ x = myC; }`

oh no is Java pass by reference or value...

(E) Include the method:
`private void getC(int x)`
`{ x = myC; }`

GO ON TO THE NEXT PAGE.

4. Consider the following code segment.

lol apparently I just skipped this question

```
int x = 7;
int y = 3;

if ((x < 10) && (y < 0))
    System.out.println("Value is: " + x * y);
else
    System.out.println("Value is: " + x / y);
```

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

(A) Value is: 21

(B) Value is: 2.3333333

(C) Value is: 2

(D) Value is: 0

(E) Value is: 1

int 7 / 3 = 2

5 Consider the following method.

```
public ArrayList<Integer> mystery(int n)
{
    ArrayList<Integer> seq = new ArrayList<Integer>();

    for (int k = 1; k <= n; k++)
        seq.add(new Integer(k * k + 3));

    return seq;
}
```

what's an ArrayList ???

oo quick reference has some stuff
seems standard

Which of the following is printed as a result of executing the following statement?

```
System.out.println(mystery(6));
```

(A) [3, 4, 7, 12, 19, 28]

~~(B) [3, 4, 7, 12, 19, 28, 39]~~

(C) [4, 7, 12, 19, 28, 39]

from 1 - 6 should be 6 elements

(D) [39, 28, 19, 12, 7, 4]

$1 * 1 + 3 = 4$

~~(E) [39, 28, 19, 12, 7, 4, 3]~~

6. Consider the following method that is intended to determine if the double values `d1` and `d2` are close enough to be considered equal. For example, given a tolerance of 0.001, the values 54.32271 and 54.32294 would be considered equal.

```
/** @return true if d1 and d2 are within the specified tolerance,
 *     false otherwise
 */
public boolean almostEqual(double d1, double d2, double tolerance)
{
    /* missing code */
}
```

Which of the following should replace `/* missing code */` so that `almostEqual` will work as intended?

- (A) `return (d1 - d2) <= tolerance;`
- (B) `return ((d1 + d2) / 2) <= tolerance;`
- (C) `return (d1 - d2) >= tolerance;`
- (D) `return ((d1 + d2) / 2) >= tolerance;`
- (E) `return Math.abs(d1 - d2) <= tolerance;`

from a math POV you def need
absolute value right

7. Consider the following class declaration.

```
public class Person
{
    private String myName;
    private int myYearOfBirth;

    public Person(String name, int yearOfBirth)
    {
        myName = name;
        myYearOfBirth = yearOfBirth;
    }

    public String getName()
    { return myName; }

    public void setName(String name)
    { myName = name; }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

omg is this what constructors are supposed to look like

Assume that the following declaration has been made.

```
Person student = new Person("Thomas", 1995);
```

Which of the following statements is the most appropriate for changing the name of `student` from "Thomas" to "Tom" ?

- (A) `student = new Person("Tom", 1995);`
- (B) `student.myName = "Tom";`
- (C) `student.getName("Tom");`
- (D) `student.setName("Tom");`
- (E) `Person.setName("Tom");`

it's definitely D but don't see why...
oh I see

myName is private so that's why B doesn't work?

8. Consider the following class declaration.

```
public class Student
{
    private String myName;
    private int myAge;

    public Student()
    { /* implementation not shown */ }

    public Student(String name, int age)
    { /* implementation not shown */ }

    // No other constructors
}
```

Which of the following declarations will compile without error?

- I. `Student a = new Student();`
 - II. `Student b = new Student("Juan", 15);`
 - III. `Student c = new Student("Juan", "15");` **age is an int not string @ III!**
- (A) I only
(B) II only
(C) I and II only
(D) I and III only
(E) I, II, and III

9. Consider the following method that is intended to return the sum of the elements in the array `key`.

```
public static int sumArray(int[] key)
{
    int sum = 0;

    for (int i = 1; i <= key.length; i++)
    {
        /* missing code */
    }

    return sum;
}
```

Which of the following statements should be used to replace `/* missing code */` so that `sumArray` will work as intended?

- (A) `sum = key[i];`
- (B) `sum += key[i - 1];`
- (C) `sum += key[i];`
- (D) `sum += sum + key[i - 1];`
- (E) `sum += sum + key[i];`

Questions 10-11 refer to the following information.

Consider the following instance variable and methods. You may assume that `data` has been initialized with `length > 0`. The methods are intended to return the index of an array element equal to `target`, or -1 if no such element exists.

```
private int[] data;

public int seqSearchRec(int target)
{
    return seqSearchRecHelper(target, data.length - 1);
}

private int seqSearchRecHelper(int target, int last)
{
    // Line 1

    if (data[last] == target)
        return last;
    else
        return seqSearchRecHelper(target, last - 1);
}
```

10. For which of the following test cases will the call `seqSearchRec(5)` always result in an error?

- I. `data` contains only one element.
- II. `data` does not contain the value 5.
- III. `data` contains the value 5 multiple times.

- (A) I only
- (B) II only
- (C) III only
- (D) I and II only
- (E) I, II, and III

11. Which of the following should be used to replace `// Line 1` in `seqSearchRecHelper` so that `seqSearchRec` will work as intended?

- (A) `if (last <= 0)`
 `return -1;`
- (B) `if (last < 0)`
 `return -1;`
- (C) `if (last < data.length)`
 `return -1;`
- (D) `while (last < data.length)`
- (E) `while (last >= 0)`

GO ON TO THE NEXT PAGE.

✓ 12. Consider the following method.

```
public String mystery(String input)
{
    String output = "";

    for (int k = 1; k < input.length(); k = k + 2)
    {
        output += input.substring(k, k + 1);
    }

    return output;
}
```

substring(from, to): Returns the substring beginning at index from and ending at index to - 1

What is returned as a result of the call `mystery("computer")` ?

I wanna say `input.substring(k, k + 1)` is the same as `input[i]/input.charAt(i)`

(A) "computer"

(B) "cmue"

(C) "optr"

(D) "ompute"

(E) Nothing is returned because an `IndexOutOfBoundsException` is thrown.

oooo thought about E but shouldn't be the case?

13. Consider the following code segment.

```
int[] arr = {7, 2, 5, 3, 0, 10};
for (int k = 0; k < arr.length - 1; k++)
{
    if (arr[k] > arr[k + 1])
        System.out.print(k + " " + arr[k] + " ");
}
```

What will be printed as a result of executing the code segment?

- (A) 0 2 2 3 3 0
- (B) 0 7 2 5 3 3
- (C) 0 7 2 5 5 10
- (D) 1 7 3 5 4 3
- (E) 7 2 5 3 3 0

14. Consider the following interface and class declarations.

```
public interface Vehicle
{
    /** @return the mileage traveled by this Vehicle
     */
    double getMileage();
}

public class Fleet
{
    private ArrayList<Vehicle> myVehicles;

    /** @return the mileage traveled by all vehicles in this Fleet
     */
    public double getTotalMileage()
    {
        double sum = 0.0;

        for (Vehicle v : myVehicles)    ooo this for loop is s e x y
        {
            sum += /* expression */ ;
        }

        return sum;
    }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

Which of the following can be used to replace `/* expression */` so that `getTotalMileage` returns the total of the miles traveled for all vehicles in the fleet?

- (A) `getMileage(v)`
- (B) `myVehicles[v].getMileage()`
- (C) `Vehicle.get(v).getMileage()`
- (D) `myVehicles.get(v).getMileage()`
- (E) `v.getMileage()`

15. Consider the following method, `isSorted`, which is intended to return `true` if an array of integers is sorted in nondecreasing order and to return `false` otherwise.

like...ascending???

```
/** @param data an array of integers
 *  @return true if the values in the array appear in sorted (nondecreasing) order
 */
public static boolean isSorted(int[] data)
{
    /* missing code */
}
```

Which of the following can be used to replace `/* missing code */` so that `isSorted` will work as intended?

I.

```
for (int k = 1; k < data.length; k++)
{
    if (data[k - 1] > data[k])
        return false;
}
return true;
```

II.

```
for (int k = 0; k < data.length; k++)
{
    if (data[k] > data[k + 1])
        return false;
}
return true;
```

pretty sure II will throw out of bounds...

III.

```
for (int k = 0; k < data.length - 1; k++)
{
    if (data[k] > data[k + 1])
        return false;
    else
        return true;
}
return true;
```

hmmm doesn't this return on k = 0 regardless of true/false

- (A) I only
(B) II only
(C) III only
(D) I and II only
(E) I and III only

16. Consider the following incomplete method that is intended to return an array that contains the contents of its first array parameter followed by the contents of its second array parameter.

```
public static int[] append(int[] a1, int[] a2)
{
    int[] result = new int[a1.length + a2.length];

    for (int j = 0; j < a1.length; j++)
        result[j] = a1[j];

    for (int k = 0; k < a2.length; k++)
        result[ /* index */ ] = a2[k];

    return result;
}
```

Which of the following expressions can be used to replace `/* index */` so that `append` will work as intended?

(A) `j`

(B) `k`

(C) `k + a1.length - 1`

(D) `k + a1.length`

yeah this was just careless :(

(E) `k + a1.length + 1`



17. Consider the following code segment.

```
int[] arr = {1, 2, 3, 4, 5, 6, 7};  
for (int k = 3; k < arr.length - 1; k++)  
    arr[k] = arr[k + 1];
```

Which of the following represents the contents of `arr` as a result of executing the code segment?

- (A) {1, 2, 3, 4, 5, 6, 7}
- (B) {1, 2, 3, 5, 6, 7}
- (C) {1, 2, 3, 5, 6, 7, 7}
- (D) {1, 2, 3, 5, 6, 7, 8}
- (E) {2, 3, 4, 5, 6, 7, 7}

only arr[3 - 5] should change

✓ 18. Assume that `myList` is an `ArrayList` that has been correctly constructed and populated with objects. Which of the following expressions produces a valid random index for `myList` ?

- (A) `(int)(Math.random() * myList.size()) - 1`
- (B) `(int)(Math.random() * myList.size())`
- (C) `(int)(Math.random() * myList.size()) + 1`
- (D) `(int)(Math.random() * (myList.size() + 1))`
- (E) `Math.random(myList.size())`

oh no
does casting to an int truncate??

I think so..

if not I'm literally inting

19 Assume that `a` and `b` have been defined and initialized as `int` values. The expression

`!(!(a != b) && (b > 7))`

said aloud: oh my god

is equivalent to which of the following?

(A) `(a != b) || (b < 7)`

(B) `(a != b) || (b <= 7)`

(C) `(a == b) || (b <= 7)`

(D) `(a != b) && (b <= 7)`

(E) `(a == b) && (b > 7)`

just wrote out a truth table to confirm that `!&& = ||`

`!(!(a != b) && (b > 7))`

`!(a != b) || (b > 7)`

`(a == b) || (b > 7)`

lmao this isn't ans isn't on there let's try again

ok turns out I read my truth table wrong and `!&& != || :(`

let's try applying the `!` to both sides

`!(!(a != b) && (b > 7))`

`(a != b) && (b <= 7)`

ok at least this answer is on there lmfao

this is exactly why I bombed
PHI 202: Intro to Logic

truth tables what i feel like ur way overcomplicating this

`!(!(a != b) && (b > 7))`

`!(a == b && b > 7)`

`a != b || b <= 7`

going from 2nd last line to last line is like saying "what does it take for not BOTH X and Y to happen? Either X doesn't happen or Y doesn't happen

`!(X && Y) -> !X || !Y`

GO ON TO THE NEXT PAGE.

✓ 20. Consider the following method.

```
public static void arrayMethod(int nums[])
{
    int j = 0;
    int k = nums.length - 1;

    while (j < k)
    {
        int x = nums[j];
        nums[j] = nums[k];
        nums[k] = x;
        j++;
        k--;
    }
}
```

said aloud: ew

Which of the following describes what the method `arrayMethod()` does to the array `nums`?

- (A) The array `nums` is unchanged.
- (B) The first value in `nums` is copied to every location in the array.
- (C) The last value in `nums` is copied to every location in the array.
- (D) The method generates an `ArrayIndexOutOfBoundsException`.
- (E) The contents of the array `nums` are reversed.

21. Consider the following method, which is intended to return the element of a 2-dimensional array that is closest in value to a specified number, `val`.

```
/** @return the element of 2-dimensional array mat whose value is closest to val */
public double findClosest(double[][] mat, double val)
{
    double answer = mat[0][0];
    double minDiff = Math.abs(answer - val);
    for (double[] row : mat)
    {
        for (double num : row)           more sexy for loops 😊
        {
            if ( /* missing code */ )
            {
                answer = num;
                minDiff = Math.abs(num - val);
            }
        }
    }
    return answer;
}
```

Which of the following could be used to replace `/* missing code */` so that `findClosest` will work as intended?

- (A) `val - row[num] < minDiff`
- (B) `Math.abs(num - minDiff) < minDiff`
- (C) `val - num < 0.0` tricky tricky
- (D) `Math.abs(num - val) < minDiff`
- (E) `Math.abs(row[num] - val) < minDiff`

22 Consider the following Book and AudioBook classes.

```
public class Book
{
    private int numPages;
    private String bookTitle;

    public Book(int pages, String title)
    {
        numPages = pages;
        bookTitle = title;
    }

    public String toString()
    {
        return bookTitle + " " + numPages;
    }

    public int length()
    {
        return numPages;
    }
}
```

can you concat with an int?

said aloud: omg so long

```
public class AudioBook extends Book
{
    private int numMinutes;

    public AudioBook(int minutes, int pages, String title)
    {
        super(pages, title);
        numMinutes = minutes;
    }

    public int length()
    {
        return numMinutes;
    }

    public double pagesPerMinute()
    {
        return ((double) super.length()) / numMinutes;
    }
}
```

I can't remember ever learning super in COS 126

but def have seen in JavaScript

uhhhh what happens when
you divide a double by an int...
gets turned into a double?

GO ON TO THE NEXT PAGE.

Consider the following code segment that appears in a class other than `Book` or `AudioBook`.

```
Line 1: Book[] books = new Book[2];
Line 2: books[0] = new AudioBook(100, 300, "The Jungle");
Line 3: books[1] = new Book(400, "Captains Courageous");
Line 4: System.out.println(books[0].pagesPerMinute());
Line 5: System.out.println(books[0].toString());
Line 6: System.out.println(books[0].length());
Line 7: System.out.println(books[1].toString());
```

Which of the following best explains why the code segment will not compile?

- (A) Line 2 will not compile because variables of type `Book` may not refer to variables of type `AudioBook`.
- (B) Line 4 will not compile because variables of type `Book` may only call methods in the `Book` class.
- (C) Line 5 will not compile because the `AudioBook` class does not have a method named `toString` declared or implemented.
- (D) Line 6 will not compile because the statement is ambiguous. The compiler cannot determine which `length` method should be called.
- (E) Line 7 will not compile because the element at index 1 in the array named `books` may not have been initialized.

D seems the most reasonable

wait so even if `books[0]`
is a `AudioBook`, it can't call
`AudioBook` methods because
it's declared as a `Book`? welp

yeah this makes sense though right? You have to explicitly
declare it as `AudioBook` to call `AudioBook` methods

otherwise random non-audio `Books` can try calling `AudioBook`-specific
methods and that's.... nonsense

~~23.~~ Consider the following instance variable and method.

```
private List<String> animals;

public void manipulate()
{
    for (int k = animals.size() - 1; k > 0; k--)
    {
        if (animals.get(k).substring(0, 1).equals("b"))
        {
            animals.add(animals.size() - k, animals.remove(k));
        }
    }
}
```

said outloud: this is so gross

Assume that `animals` has been instantiated and initialized with the following contents.

```
["bear", "zebra", "bass", "cat", "koala", "baboon"]
```

What will the contents of `animals` be as a result of calling `manipulate` ?

- (A) ["baboon", "zebra", "bass", "cat", "bear", "koala"]
- (B) ["bear", "zebra", "bass", "cat", "koala", "baboon"]
- (C) ["baboon", "bear", "zebra", "bass", "cat", "koala"]
- (D) ["bear", "baboon", "zebra", "bass", "cat", "koala"]
- (E) ["zebra", "cat", "koala", "baboon", "bass", "bear"]

baboon, bear, zebra, bass, cat, koala
 baboon, bear, bass, zebra, cat, koala
 baboon, bear, zebra, bass, cat, koala
 baboon, zebra, bass, cat, bear, koala
 zebra, bass, cat, bear, koala, baboon

rip this isn't on there uh

lemme come back to this if I have time

I STILL DO NOT KNOW
WHY THIS IS THE ANSWER

yeah im too lazy to think about this

aight take two:

yeah I realized I thought size() = 5 not 6 oop
 bear zebra bass cat koala baboon
 bear baboon zebra bass cat koala
 bear baboon bass zebra cat koala
 bear baboon zebra cat bass koala
 bear zebra cat bass koala baboon
 zebra cat bass koala baboon bear

FUCK why is this still not an answer

bear zebra bass cat koala baboon
 baboon bear zebra bass cat koala
 baboon bear bass zebra cat koala
 baboon bear zebra bass cat koala
 baboon zebra bass cat bear koala

alright we're just gonna guess

24. Consider the following code segment.

```
int[] oldArray = {1, 2, 3, 4, 5, 6, 7, 8, 9};
int[][] newArray = new int[3][3];

int row = 0;
int col = 0;
for (int value : oldArray)
{
    newArray[row][col] = value;
    row++;
    if ((row % 3) == 0)
    {
        col++;
        row = 0;
    }
}

System.out.println(newArray[0][2]);
```

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

- (A) 3
- (B) 4
- (C) 5
- (D) 7
- (E) 8

25. A rectangular box fits inside another rectangular box if and only if the height, width, and depth of the smaller box are each less than the corresponding values of the larger box. Consider the following three interface declarations that are intended to represent information necessary for rectangular boxes.

```
I. public interface RBox
{
    /** @return the height of this RBox */
    double getHeight();

    /** @return the width of this RBox */
    double getWidth();

    /** @return the depth of this RBox */
    double getDepth();
}
```

wait what's the diff between interface and class oh no

```
II. public interface RBox
{
    /** @return true if the height of this RBox is less than the height of other;
     *      false otherwise
     */
    boolean smallerHeight(RBox other);

    /** @return true if the width of this RBox is less than the width of other;
     *      false otherwise
     */
    boolean smallerWidth(RBox other);

    /** @return true if the depth of this RBox is less than the depth of other;
     *      false otherwise
     */
    boolean smallerDepth(RBox other);
}
```

```
III. public interface RBox
{
    /** @return the surface area of this RBox */
    double getSurfaceArea();

    /** @return the volume of this RBox */
    double getVolume();
}
```

Which of the interfaces, if correctly implemented by a `Box` class, would be sufficient functionality for a user of the `Box` class to determine if one `Box` can fit inside another?

~~(A) I only~~

do we need I??

(B) II only

I'll come back to this welp

~~(C) III only~~

(D) I and II only

~~(E) I, II, and III~~

GO ON TO THE NEXT PAGE.

26. Assume that the array `arr` has been defined and initialized as follows.

```
int[] arr = /* initial values for the array */ ;
```

Which of the following will correctly print all of the odd integers contained in `arr` but none of the even integers contained in `arr` ?

- (A)

```
for (int x : arr)
    if (x % 2 != 0)
        System.out.println(x);
```
- (B)

```
for (int k = 1; k < arr.length; k++)
    if (arr[k] % 2 != 0)
        System.out.println(arr[k]);
```
- (C)

```
for (int x : arr)
    if (x % 2 != 0)
        System.out.println(arr[x]);
```
- (D)

```
for (int k = 0; k < arr.length; k++)
    if (arr[k] % 2 != 0)
        System.out.println(k);
```
- (E)

```
for (int x : arr)
    if (arr[x] % 2 != 0)
        System.out.println(arr[x]);
```

Questions 27-28 refer to the following method.

```
public static int mystery(int n)
{
    int x = 1;
    int y = 1;

    // Point A

    while (n > 2)
    {
        x = x + y;

        // Point B

        y = x - y;
        n--;
    }

    // Point C

    return x;
}
```

27. What value is returned as a result of the call `mystery(6)` ?

- (A) 1
- (B) 5
- (C) 6
- (D) 8
- (E) 13

28. Which of the following is true of method `mystery` ?

- (A) `x` will sometimes be 1 at // Point B.
- (B) `x` will never be 1 at // Point C.
- (C) `n` will never be greater than 2 at // Point A.
- (D) `n` will sometimes be greater than 2 at // Point C.
- (E) `n` will always be greater than 2 at // Point B.

29. Consider the following code segment.

```
for (int k = 1; k <= 100; k++)  
    if ((k % 4) == 0)  
        System.out.println(k);
```

Which of the following code segments will produce the same output as the code segment above?

- (A)

```
for (int k = 1; k <= 25; k++)  
    System.out.println(k);
```
- (B)

```
for (int k = 1; k <= 100; k = k + 4)  
    System.out.println(k);
```
- (C)

```
for (int k = 1; k <= 100; k++)  
    System.out.println(k % 4);
```
- (D)

```
for (int k = 4; k <= 25; k = 4 * k)  
    System.out.println(k);
```
- (E)

```
for (int k = 4; k <= 100; k = k + 4)  
    System.out.println(k);
```

30. Consider the following method.

```
public static String scramble(String word, int howFar)
{
    return word.substring(howFar + 1, word.length()) +
           word.substring(0, howFar);
}
```

What value is returned as a result of the call `scramble("compiler", 3)`?

(A) "compiler"

(B) "pilercom"

(C) "ilercom"

(D) "ilercomp"

(E) No value is returned because an `IndexOutOfBoundsException` will be thrown.

31. Consider the following method.

```
public void mystery(int[] data)
{
    for (int k = 0; k < data.length - 1; k++)
        data[k + 1] = data[k] + data[k + 1];
}
```

The following code segment appears in another method in the same class.

```
int[] values = {5, 2, 1, 3, 8};
mystery(values);
for (int v : values)
    System.out.print(v + " ");
System.out.println();
```

5 7 1 3 8
5 7 8 3 8
5 7 8 11 8
5 7 8 11 19

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

- (A) 5 2 1 3 8
- (B) 5 7 3 4 11
- (C) 5 7 8 11 19
- (D) 7 3 4 11 8
- (E) Nothing is printed because an `ArrayIndexOutOfBoundsException` is thrown during the execution of method `mystery`.

32. Consider the following method.

```
public int compute(int n, int k)
{
    int answer = 1;

    for (int i = 1; i <= k; i++)
        answer *= n;

    return answer;
}
```

Which of the following represents the value returned as a result of the call `compute(n, k)` ?

(A) $n*k$

(B) $n!$

(C) n^k

(D) 2^k

(E) k^n

33. Consider the following code segment.

```
int sum = 0;
int k = 1;
while (sum < 12 || k < 4)
    sum += k;

System.out.println(sum);
```

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

(A) 6

(B) 10

(C) 12

(D) 15

(E) Nothing is printed due to an infinite loop.

k is always 1
while loop statement is an || statement
k < 4 is always true so infinite I think



24. Consider the following class declarations.

```
public class Point
{
    private double x; // x-coordinate
    private double y; // y-coordinate

    public Point()
    {
        x = 0;
        y = 0;
    }

    public Point(double a, double b)
    {
        x = a;
        y = b;
    }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

```
public class Circle
{
    private Point center;
    private double radius;

    /** Constructs a circle where (a, b) is the center and r is the radius.
    */
    public Circle(double a, double b, double r)
    {
        /* missing code */
    }
}
```

Which of the following replacements for `/* missing code */` will correctly implement the `Circle` constructor?

- I. `center = new Point();`
`radius = r;`
- II. `center = new Point(a, b);`
`radius = r;`
- III. `center = new Point();`
`center.x = a;`
`center.y = b;`
`radius = r;`

I def wrong because center will always be (0, 0)
x and y are private variables in Point so it makes me think that III is wrong

- (A) I only
- (B) II only
- (C) III only
- (D) II and III only
- (E) I, II, and III

35. Consider the following code segment.

```
int num = 2574;
int result = 0;

while (num > 0)
{
    result = result * 10 + num % 10;
    num /= 10;
}
System.out.println(result);
```

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

- (A) 2
- (B) 4
- (C) 18
- (D) 2574
- (E) 4752

36. Consider the following method.

```
public void test(int x)
{
    int y;

    if (x % 2 == 0)
        y = 3;
    else if (x > 9)
        y = 5;
    else
        y = 1;

    System.out.println("y = " + y);
}
```

need:
- even,
- greater than 9,
- odd and <= 9

Which of the following test data sets would test each possible output for the method?

~~(A)~~ 8, 9, 12

~~(B)~~ 7, 9, 11

(C) 8, 9, 11

(D) 8, 11, 13

(E) 7, 9, 10

37. Consider the following code segment.

```
int x = 1;
while ( /* missing code */ )
{
    System.out.print(x + " ");
    x = x + 2;
}
```

Consider the following possible replacements for `/* missing code */`.

- I. $x < 6$
- II. $x \neq 6$
- III. $x < 7$

Which of the proposed replacements for `/* missing code */` will cause the code segment to print only the values 1 3 5?

- (A) I only
- (B) II only
- (C) I and II only
- (D) I and III only
- (E) I, II, and III

38. Assume that `x` and `y` have been declared and initialized with `int` values. Consider the following Java expression.

`(y > 10000) || (x > 1000 && x < 1500)`

oh booleans oh no
we struggled with this last time

Which of the following is equivalent to the expression given above?

- (A) `(y > 10000 || x > 1000) && (y > 10000 || x < 1500)`
- (B) `(y > 10000 || x > 1000) || (y > 10000 || x < 1500)`
- (C) `(y > 10000) && (x > 1000 || x < 1500)`
- (D) `(y > 10000 && x > 1000) || (y > 10000 && x < 1500)`
- (E) `(y > 10000 && x > 1000) && (y > 10000 && x < 1500)`

my gut is telling me that `||` is distributive

GO ON TO THE NEXT PAGE.

39. Consider the following recursive method.

```
public int recur(int n)
{
    if (n <= 10)
        return n * 2;
    else
        return recur(recur(n / 3));
}
```

What value is returned as a result of the call `recur(27)` ?

calls to recur:
27, 9, 18, 6, 12, 4, 8

- (A) 8
- (B) 9
- (C) 12
- (D) 16
- (E) 18

40. Consider the following recursive method.

```
public static void whatsItDo(String str)
{
    int len = str.length();
    if (len > 1)
    {
        String temp = str.substring(0, len - 1);
        whatsItDo(temp);
        System.out.println(temp);
    }
}
```

last one woooo 🎉

ahhh more recursion

What is printed as a result of the call `whatsItDo("WATCH")` ?

- (A) WATC
WAT
WA
W
- (B) WATCH
WATC
WAT
WA
- (C) W
WA
WAT
WATC
- (D) W
WA
WAT
WATC
WATCH
- (E) WATCH
WATC
WAT
WA
W
WA
WAT
WATC
WATCH

it's either C or D

think it's C cuz we start with `substring(0, 4) = WATC`

ok I have exactly 15 min to check my work/revisit 3 questions

END OF SECTION I

**IF YOU FINISH BEFORE TIME IS CALLED,
YOU MAY CHECK YOUR WORK ON THIS SECTION.**

DO NOT GO ON TO SECTION II UNTIL YOU ARE TOLD TO DO SO.

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 ≤ index ≤ 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>

Section II

Free-Response Questions

COMPUTER SCIENCE A
SECTION II
Time—1 hour and 30 minutes
Number of questions—4
Percent of total grade—50

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.
1. Consider the following partial declaration for a `WordScrambler` class. The constructor for the `WordScrambler` class takes an even-length array of `String` objects and initializes the instance variable `scrambledWords`.

```
public class WordScrambler                                     said out loud: my brain is fried 🔍
{
    private String[] scrambledWords;

    /** @param wordArr an array of String objects
     *     Precondition: wordArr.length is even
     */
    public WordScrambler(String[] wordArr)
    {
        scrambledWords = mixedWords(wordArr);
    }

    /** @param word1 a String of characters
     *     @param word2 a String of characters
     *     @return a String that contains the first half of word1 and the second half of word2
     */
    private String recombine(String word1, String word2)
    { /* to be implemented in part (a) */ }

    /** @param words an array of String objects
     *     Precondition: words.length is even
     *     @return an array of String objects created by recombining pairs of strings in array words
     *     Postcondition: the length of the returned array is words.length
     */
    private String[] mixedWords(String[] words)
    { /* to be implemented in part (b) */ }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

GO ON TO THE NEXT PAGE.

(a) Write the `WordScrambler` method `recombine`. This method returns a `String` created from its two `String` parameters as follows.

- take the first half of `word1`
- take the second half of `word2`
- concatenate the two halves and return the new string.

said out loud: I genuinely don't know how to write Java

you definitely cannot do nested methods lol that is some JS stuff
Java is nothing like JS

For example, the following table shows some results of calling `recombine`. Note that if a word has an odd number of letters, the second half of the word contains the extra letter.

word1	word2	recombine(word1, word2)
"apple"	"pear"	"applepear"
"apple"	"pearr"	"applepearr"
"apple"	"pearrr"	"applepearrr"

```
private String recombine(String word1, String word2) {
    /* idk if you can do nested methods */
    private int getHalfIndex (String word) {
        return word.length() / 2;
    }
}
```

Oh I got fancy here and used a helper method

Complete method `recombine`

```
int half1 = getHalfIndex(word1);
int half2 = getHalfIndex(word2);

return word1.substring(0, half1) + word2.
        substring(half2);
}

/** @param word1 a St
 * @param word2 a St
 * @return a String
 */
private String recombine(String word1, String word2)
```

(b) Write the `WordScrambler` method `mixedWords`. This method creates and returns a new array of `String` objects as follows.

It takes the first pair of strings in `words` and combines them to produce a pair of strings to be included in the array returned by the method. If this pair of strings consists of `w1` and `w2`, the method should include the result of calling `recombine` with `w1` and `w2` as arguments and should also include the result of calling `recombine` with `w2` and `w1` as arguments. The method should then process the next pair to be processed by this method. This process should be repeated until all pairs have been processed in this way and the new array contains the following elements:

so I did this in place but the scoring guidelines didn't say that's wrong
having the upper bound be `words.length - 1` instead of `words.length` should be fine?

then the call `mixedWords(words)` should return an array containing the following elements:

```
private String[] mixedWords(String[] words) {
    for (int i = 0; i < words.length - 1; i += 2) {
        String word1 = words[i];
        String word2 = words[i + 1];

        words[i] = recombine(word1, word2);
        words[i + 1] = recombine(word2, word1);
    }
}
```

In writing `mixedWords`, you may call `recombine` regardless of what you wrote in part (a).

Complete method `mixedWords` below.

```
return words;
}

/** @param words an array of String objects
 * Precondition: words.length is even
 * @return an array of String objects created by recombining pairs of strings in array words
 * Postcondition: the length of the returned array is words.length
 */
private String[] mixedWords(String[] words)
```

2. An array of positive integer values has the *mountain* property if the elements are ordered such that successive values increase until a maximum value (the peak of the mountain) is reached and then the successive values decrease. The `Mountain` class declaration shown below contains methods that can be used to determine if an array has the mountain property. You will implement two methods in the `Mountain` class.

```
public class Mountain
{
    /** @param array an array of positive integer values
     *   @param stop the last index to check
     *   Precondition:  $0 \leq \text{stop} < \text{array.length}$ 
     *   @return true if for each  $j$  such that  $0 \leq j < \text{stop}$ ,  $\text{array}[j] < \text{array}[j + 1]$  ;
     *   false otherwise
     */
    public static boolean isIncreasing(int[] array, int stop)
    { /* implementation not shown */ }

    /** @param array an array of positive integer values
     *   @param start the first index to check
     *   Precondition:  $0 \leq \text{start} < \text{array.length} - 1$ 
     *   @return true if for each  $j$  such that  $\text{start} \leq j < \text{array.length} - 1$ ,
     *    $\text{array}[j] > \text{array}[j + 1]$ ;
     *   false otherwise
     */
    public static boolean isDecreasing(int[] array, int start)
    { /* implementation not shown */ }

    /** @param array an array of positive integer values
     *   Precondition:  $\text{array.length} > 0$ 
     *   @return the index of the first peak (local maximum) in the array, if it exists;
     *   -1 otherwise
     */
    public static int getPeakIndex(int[] array)
    { /* to be implemented in part (a) */ }

    /** @param array an array of positive integer values
     *   Precondition:  $\text{array.length} > 0$ 
     *   @return true if array contains values ordered as a mountain;
     *   false otherwise
     */
    public static boolean isMountain(int[] array)
    { /* to be implemented in part (b) */ }

    // There may be instance variables, constructors, and methods that are not shown.
}
}
```

- (a) Write the Mountain method `getPeakIndex`. Method `getPeakIndex` returns the index of the first peak found in the parameter `array`, if one exists. A peak is defined as an element whose value is greater than the value of the element immediately before it and is also greater than the value of the element immediately after it. Method `getPeakIndex` starts at the beginning of the array and returns the index of the first peak that is found or -1 if no peak is found.

For example, the following table illustrates the results of several calls to `getPeakIndex`.

arr	getPeakIndex(arr)
{11, 22, 33, 22, 11}	2
{11, 22, 11, 22, 11}	1
{11, 22, 33, 55, 77}	-1
{99, 33, 55, 77, 120}	-1
{99, 33, 55, 77, 55}	3
{33, 22, 11}	-1

Complete method `getPeakIndex` below.

```

/** @param array an array of positive integer values
 *   Precondition: array.length > 0
 *   @return the index of the first peak (local maximum) in the array, if it exists;
 *           -1 otherwise
 */
public static int getPeakIndex(int[] array)

public static int getPeakIndex(int[] array) {
    for (int i = 1; i < array.length - 1; i++) {
        int before = array[i - 1];
        int curr = array[i];
        int after = array[i + 1];

        return (before < curr) && (curr > after);
    }
    return -1;
}

```

cool

- (b) Write the Mountain method `isMountain`. Method `isMountain` returns `true` if the values in the parameter `array` are ordered as a mountain; otherwise, it returns `false`. The values in `array` are ordered as a mountain if all three of the following conditions hold.
- There must be a peak.
 - The array elements with an index smaller than the peak's index must appear in increasing order.
 - The array elements with an index larger than the peak's index must appear in decreasing order.

For example, the following table illustrates the results of several calls to `isMountain`.

<code>arr</code>	<code>isMountain(arr)</code>
{1, 2, 3, 2, 1}	true
{1, 2, 1, 2, 1}	false
{1, 2, 3, 1, 5}	false
{1, 4, 2, 1, 0}	true
{9, 3, 5, 7, 5}	false
{3, 2, 1}	false

In writing `isMountain`, assume that `getPeakIndex` works as specified, regardless of what you wrote in part (a).

Complete method `isMountain` below.

```
/** @param array an array of positive integer values
 *      Precondition: array.length > 0
 *      @return true if array contains values ordered as a mountain;
 *              false otherwise
 */
public static boolean isMountain(int[] array)
```

```
    public static boolean isMountain(int[] array) {
        int peak = getPeakIndex(array);

        return (peak != -1) && isIncreasing(peak, array) &&
            isDecreasing(peak, array);
    }
```

ayyy my solution is char for char the same as theirs

3. A two-dimensional array of temperatures is represented in the following class.

```
public class TemperatureGrid
{
    /** A two-dimensional array of temperature values, initialized in the constructor (not shown)
     *   Guaranteed not to be null
     */
    private double[][] temps;

    /** Computes and returns a new temperature value for the given location.
     *   @param row a valid row index in temps
     *   @param col a valid column index in temps
     *   @return the new temperature for temps[row][col]
     *           - The new temperature for any element in the border of the array is the
     *             same as the old temperature.
     *           - Otherwise, the new temperature is the average of the four adjacent entries.
     *   Postcondition: temps is unchanged.
     */
    private double computeTemp(int row, int col)
    { /* to be implemented in part (a) */ }

    /** Updates all values in temps and returns a boolean that indicates whether or not all the
     *   new values were within tolerance of the original values.
     *   @param tolerance a double value >= 0
     *   @return true if all updated temperatures are within tolerance of the original values;
     *           false otherwise.
     *   Postcondition: Each value in temps has been updated with a new value based on the
     *           corresponding call to computeTemp.
     */
    public boolean updateAllTemps(double tolerance)
    { /* to be implemented in part (b) */ }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

(a) Write method `computeTemp`, which computes and returns the new temperature for a given element of `temps` according to the following rules.

1. If the element is in the border of the array (in the first row or last row or first column or last column), the new temperature is the same as the old temperature.
2. Otherwise, the new temperature is the average (arithmetic mean) of the temperatures of the four adjacent values in the table (located above, below, to the left, and to the right of the element).

If `temps` is the table shown below, `temps.length` is 5 and `temps[0].length` is 6.

	0	1	2	3	4	5
0	95.5	100.0	100.0	100.0	100.0	110.3
1	0.0	50.0	50.0	50.0	50.0	0.0
2	0.0	40.0	40.0	40.0	40.0	0.0
3	0.0	20.0	20.0	20.0	20.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0

The following table shows the results of several calls to `computeTemp`.

Function Call	Result
<code>computeTemp(2, 3)</code>	37.5 (the average of the values 50.0, 20.0, 40.0, and 40.0)
<code>computeTemp(1, 1)</code>	47.5 (the average of the values 100.0, 40.0, 0.0, and 50.0)
<code>computeTemp(0, 2)</code>	100.0 (the same as the old value)
<code>computeTemp(1, 3)</code>	60.0 (the average of the values 100.0, 40.0, 50.0, and 50.0)

Complete method `computeTemp` below.

```

/** Computes and returns a new temperature value for the given location.
 * @param row a valid row index in temps
 * @param col a valid column index in temps
 * @return the new temperature for temps[row][col]
 * - The new temperature for any element in the border of the array is the
 *   same as the old temperature.
 * - Otherwise, the new temperature is the average of the four adjacent entries.
 * Postcondition: temps is unchanged.
 */
private double computeTemp(int row, int col)

private double computeTemp(int row, int col) {
    private boolean isBoundary() {
        return row == 0 || row == temps.length - 1 || col == 0 || col == temps[0].length - 1;
    }

    if (isBoundary()) {
        return temp[row][col];
    } else {
        return (temp[row - 1][col] + temp[row + 1][col] + temp[row][col - 1] + temp[row][col + 1]) / 4;
    }
}

```

-1 pt, should be 4.0

this is a classic

- (b) Write method `updateAllTemps`, which computes the new temperature for every element of `temps`. The new values should be based on the original values, so it will be necessary to create another two-dimensional array in which to store the new values. Once all the computations are complete, the new values should replace the corresponding positions of `temps`. Method `updateAllTemps` also determines whether every new temperature is within `tolerance` of the corresponding old temperature (i.e., the absolute value of the difference between the old temperature and the new temperature is less than or equal to `tolerance`). If so, it returns `true`; otherwise, it returns `false`.

If `temps` contains the values shown in the first table below, then the call `updateAllTemps(0.01)` should update `temps` as shown in the second table.

temps before the call updateAllTemps(0.01)

	0	1	2	3	4	5
0	95.5	100.0	100.0	100.0	100.0	110.3
1	0.0	50.0	50.0	50.0	50.0	0.0
2	0.0	40.0	40.0	40.0	40.0	0.0
3	0.0	20.0	20.0	20.0	20.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0

temps after the call updateAllTemps(0.01)

	0	1	2	3	4	5
0	95.5	100.0	100.0	100.0	100.0	110.3
1	0.0	47.5	60.0	60.0	47.5	0.0
2	0.0	27.5	37.5	37.5	27.5	0.0
3	0.0	15.0	20.0	20.0	15.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0

In the example shown, the call `updateAllTemps(0.01)` should return `false` because there are several new temperatures that are not within the given tolerance of the corresponding old temperature. For example, the updated value in `temps[2][3]` is 37.5, the original value in `temps[2][3]` was 40.0, and the absolute value of $(37.5 - 40.0)$ is greater than the value of `tolerance` (0.01).

Assume that `computeTemp` works as specified, regardless of what you wrote in part (a).

Complete method `updateAllTemps` below.

```
/** Updates all values in temps and returns a boolean that indicates whether or not all the
 * new values were within tolerance of the original values.
 * @param tolerance a double value >= 0
 * @return true if all updated temperatures are within tolerance of the original values;
 *         false otherwise.
 * Postcondition: Each value in temps has been updated with a new value based on the
 *                 corresponding call to computeTemp.
 */
public boolean updateAllTemps(double tolerance)
```

```
public boolean updateAllTemps(double tolerance) {
    double[][] updated = new double[temps.length][temps[0].length];
    boolean isAllWithinTolerance = True;

    private boolean isWithinTolerance(double temp, double newTemp) {
        return Math.abs(temp - newTemp) <= tolerance;
    }

    for (int i = 0; i < updated; i++) {
        for (int j = 0; j < updated[0].length; j++) {
            double newTemp = computeTemp(i, j);

            updated[i][j] = computeTemp(i, j);
            isAllWithinTolerance = isAllWithinTolerance && isWithinTolerance(newTemp, temp[i][j]);
        }
    }

    temps = updated;
    return isAllWithinTolerance;
}
```

lol u really like the nested functions thing
im pretty sure this doesn't even come close to compiling

hmm even though apparently
I suck at boolean operators,
&& should work here

4. A school district would like to get some statistics on its students' standardized test scores. Scores will be represented as objects of the following `ScoreInfo` class. Each `ScoreInfo` object contains a score value and the number of students who earned that score.

```
public class ScoreInfo
{
    private int score;
    private int numStudents;

    public ScoreInfo(int aScore)
    {
        score = aScore;
        numStudents = 1;
    }

    /** adds 1 to the number of students who earned this score
     */
    public void increment()
    { numStudents++; }

    /** @return this score
     */
    public int getScore()
    { return score; }

    /** @return the number of students who earned this score
     */
    public int getFrequency()
    { return numStudents; }
}
```

The following `Stats` class creates and maintains a database of student score information. The scores are stored in sorted order in the database.

I want my python sort method with a nice lambda function 🙄

```
public class Stats
{
    private ArrayList<ScoreInfo> scoreList;
    // listed in increasing score order; no two ScoreInfo objects contain the same score

    /** Records a score in the database, keeping the database in increasing score order. If no other
     * ScoreInfo object represents score, a new ScoreInfo object representing score
     * is added to the database; otherwise, the frequency in the ScoreInfo object representing
     * score is incremented.
     * @param score a score to be recorded in the list
     * @return true if a new ScoreInfo object representing score was added to the list;
     *         false otherwise
     */
    public boolean record(int score)
    { /* to be implemented in part (a) */ }

    /** Records all scores in stuScores in the database, keeping the database in increasing score order
     * @param stuScores an array of student test scores
     */
    public void recordScores(int[] stuScores)
    { /* to be implemented in part (b) */ }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

- (a) Write the `Stats` method `record` that takes a test score and records that score in the database. If the score already exists in the database, the frequency of that score is updated. If the score does not exist in the database, a new `ScoreInfo` object is created and inserted in the appropriate position so that the database is maintained in increasing score order. The method returns `true` if a new `ScoreInfo` object was added to the database; otherwise, it returns `false`.

Complete method `record` below.

```
/** Records a score in the database, keeping the database in increasing score order. If no other
 * ScoreInfo object represents score, a new ScoreInfo object representing score
 * is added to the database; otherwise, the frequency in the ScoreInfo object representing
 * score is incremented.
 * @param score a score to be recorded in the list
 * @return true if a new ScoreInfo object representing score was added to the list;
 *         false otherwise
 */
public boolean record(int score) {
    int i = 0;
    while (i < scoreList.size() && scoreList.get(i).getScore() < score {
        i++;
    }

    boolean exists = i < scoreList.size() && scoreList.get(i).getScore() == score;
    if (exists) {
        scoreList.get(i).increment();
    } else {
        scoreList.add(i, new ScoreInfo(score));
    }

    return !exists;
}
```

NEXT PAGE.

- (b) Write the `Stats` method `recordScores` that takes an array of test scores and records them in the database. The database contains at most one `ScoreInfo` object per unique score value. Each `ScoreInfo` object contains a score and an associated frequency. The database is maintained in increasing order based on the score.

In writing `recordScores`, assume that `record` works as specified, regardless of what you wrote in part (a).

Complete method `recordScores` below.

```
/** Records all scores in stuScores in the database, keeping the database in increasing score order
 * @param stuScores an array of student test scores
 */
public void recordScores(int[] stuScores)
```

```
public void recordScores(int[] stuScores) {
    for (int i = 0; i < stuScores.length; i++) {
        record(stuScore[i]);
    }
}
```

damn the solution used a
sexy for in loop

STOP
END OF EXAM

I finished with 38 minutes to spare
I guess I'll check my work

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 ≤ index ≤ 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>

Name: _____

**AP[®] Computer Science A
Student Answer Sheet
for Multiple-Choice Section**

No.	Answer
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	

No.	Answer
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	

AP[®] Computer Science A
Multiple-Choice Answer Key

No.	Answer
1	C
2	A
3	B
4	C
5	C
6	E
7	D
8	C
9	B
10	B
11	B
12	C
13	B
14	E
15	A
16	D
17	C
18	B
19	B
20	E
21	D
22	B
23	B
24	D
25	D
26	A
27	D
28	E
29	E
30	C

No.	Answer
31	C
32	C
33	E
34	B
35	E
36	C
37	D
38	A
39	D
40	C

AP[®] Computer Science A

Free-Response Scoring Guidelines

Question 1: Word Scrambler

Part A:	<code>recombine</code>	4 points
----------------	------------------------	-----------------

- +1 correctly finds middle index of each word
- +1 correctly forms first half of `word1` and second half of `word2` using identified index (can be off by one)
- +1 first half and second half of each word is correct length (second half of an odd length word is 1 character longer than first half)
- +1 concatenates and returns “combined” string

Part B:	<code>mixedWords</code>	5 points
----------------	-------------------------	-----------------

- +1 creates and returns a result array of correct length
- +2 attempts to loop over all pairs of strings in `words`
 - +1 attempt
 - +1 correct (loses for off-by-one pairing)
- +1 correctly combines pairs of strings (must use `recombine`)
- +1 correctly stores all new string pairs in result array

AP[®] Computer Science A
Free-Response Scoring Guidelines

Question 2: Mountain

Part A:	<code>getPeakIndex</code>	5 points
----------------	---------------------------	-----------------

- +2 loop over array
 - +1 Accesses a consecutive triplet of `array` values (*must be in context of loop*).
 - +1 Accesses all and only triplets of `array` values (*no boundary errors*).
- +2 compare for peak
 - +1 Compares middle element of `array` triplet to both previous and subsequent `array` values.
 - +1 Determines if array element is a peak.
- +1 Returns peak index or -1 if there is no peak.

Part B:	<code>isMountain</code>	4 points
----------------	-------------------------	-----------------

- +1 Calls `getPeakIndex` method.
- +1 Calls `isIncreasing` and `isDecreasing` methods.
- +1 Returns `false` when `getPeakIndex` returns -1 (*must not call `isIncreasing` and `isDecreasing`*).
- +1 Returns `true` if `array` has the mountain property.

AP[®] Computer Science A Free-Response Scoring Guidelines

Question 3: Compute Temperatures

Part A:	<code>computeTemp</code>	4 points
----------------	--------------------------	-----------------

- +2 Returns border element
- +1 Checks border conditions
- +1 Returns element when “on border”
- +1 Computes sum of the four adjacent elements (*must not assign to an int*)
- +1 Returns the average when not “on border”

Part B:	<code>updateAllTemps</code>	5 points
----------------	-----------------------------	-----------------

- +1 Instantiates a new `double` array with same number of rows and columns as `temps`
- +1 Accesses all and only values in `temps` or new array (*no bounds errors*)
- +1 Calls `computeTemp` and stores results in the corresponding elements of the new array
- +1 At exit: `temps` contains all of the updated temperatures
- +1 Returns whether all temp changes are in tolerance

AP[®] Computer Science A

Free-Response Scoring Guidelines

Question 4: Score Statistics

Part A:	<code>record</code>	7 points
----------------	---------------------	-----------------

- +1 Compares `score` value with value retrieved from object in list (must use `getScore`)
- +1 Compares `score` with all appropriate entries in `scoreList`
(*no bounds error, early exit, or infinite loop*)
- +1 Creates a new `ScoreInfo` object containing `score`
- +1 Inserts object into list based on a comparison (other than equality) with object in list
(*point not awarded if inserted more than once*)
- +1 Inserts new `ScoreInfo` object into `scoreList` once and only once in maintaining numerical order and numerical uniqueness (*no destruction of existing data*)
- +1 All `ScoreInfo` objects in `scoreList` have correct frequencies after updates if any.
- +1 correctly return `boolean` value based on whether a new `ScoreInfo` object was added
- 2 Add to `scoreList` inside for-each loop of `scoreList`
(gets `ConcurrentModificationException`).

Part B:	<code>recordScores</code>	2 points
----------------	---------------------------	-----------------

- +1 correctly loop over all scores in `stuScores`
- +1 call `record(someInt)` (*in context of loop*)

AP[®] Computer Science A
Free-Response Canonical Solutions

Question 1: Word Scrambler

PART A:

```
private String recombine(String word1, String word2)
{
    return word1.substring(0, word1.length() / 2) +
           word2.substring(word2.length() / 2);
}
```

PART B:

```
private String[] mixedWords(String[] words)
{
    String[] result = new String[words.length];

    for (int k = 0; k < result.length; k = k + 2)
    {
        result[k] = recombine(words[k], words[k + 1]);
        result[k + 1] = recombine (words[k + 1], words[k]);
    }

    return result;
}
```

AP[®] Computer Science A
Free-Response Canonical Solutions

Question 2: Mountain

PART A:

```
public static int getPeakIndex(int[] array)
{
    for (int k = 1; k < array.length - 1; k++)
    {
        if (array[k - 1] < array[k] && array[k] > array[k + 1])
            return k;
    }

    return -1;
}
```

PART B:

```
public static boolean isMountain(int[] array)
{
    int peak = getPeakIndex(array);
    return (peak != -1) && isIncreasing(array, peak) &&
           isDecreasing(array, peak);
}
```

AP[®] Computer Science A
Free-Response Canonical Solutions

Question 3: Compute Temperatures

PART A:

```
private double computeTemp(int row, int col)
{
    if (row == 0 || row == temps.length - 1
        || col == 0 || col == temps[0].length - 1)
    {
        return temps[row][col];
    }

    double sum = temps[row - 1][col] + temps[row + 1][col]
                + temps[row][col - 1] + temps[row][col + 1];

    return sum / 4.0;
}
```

PART B:

```
public boolean updateAllTemps(double tolerance)
{
    double[][] newTemps = new double[temps.length][temps[0].length];

    boolean within = true;

    for (int r = 0; r < temps.length; r++)
    {
        for (int c = 0; c < temps[0].length; c++)
        {
            newTemps[r][c] = computeTemp(r, c);
            if (Math.abs(newTemps[r][c] - temps[r][c]) > tolerance)
            {
                within = false;
            }
        }
    }

    temps = newTemps;
    return within;
}
```

AP[®] Computer Science A
Free-Response Canonical Solutions

Question 4: Score Statistics

PART A:

```
public boolean record(int score)
{
    int k = 0;
    while (k < scoreList.size() && score > scoreList.get(k).getScore())
    {
        k++;
    }

    boolean found = k < scoreList.size() &&
                    score == scoreList.get(k).getScore();
    if (found)
        scoreList.get(k).increment();
    else
        scoreList.add(k, new ScoreInfo(score));

    return !found;
}
```

Alternate solution

```
public boolean record(int score)
{
    for (int k = 0; k < scoreList.size(); k++)
    {
        if (score < scoreList.get(k).getScore())
        {
            scoreList.add(k, new ScoreInfo(score));
            return true;
        }
        else if (score == scoreList.get(k).getScore())
        {
            scoreList.get(k).increment();
            return false;
        }
    }

    scoreList.add(new ScoreInfo(score));
    return true;
}
```

PART B:

```
public void recordScores(int[] stuScores)
{
    for (int score : stuScores)
        record(score);
}
```