Strategies for Taking the Exam

1

Take time to deliberate, but when the time for action comes, stop thinking and go in.
—Napoléon Bonaparte

→ **Strategies for Multiple-Choice Questions**

→ **Strategies for Free-Response Questions**

THE MULTIPLE-CHOICE SECTION

What Is Tested?

The questions in the multiple-choice section span the entire AP Java subset, and the types of questions are classified according to the type of content. The table below shows the percentage of questions in each category, as described in the College Board's AP Computer Science A Course Description. Since categories can overlap in a given question (for example, a question can compare a recursive algorithm with a fundamental loop implementation), the percentage total is greater than 100%.

Category	Percentage of Multiple-Choice Questions
Programming Fundamentals	55 – 75%
Data Structures	20 – 40%
Logic	5 – 15%
Algorithms and Problem Solving	25 – 45%
Object-oriented Programming	15 – 25%
Recursion	5 – 15%
Software Engineering	2 – 10%

Time Issues

You have 90 minutes for 40 questions, which means a little more than 2 minutes per question. There are, however, several complicated questions that need to be hand traced, which may take longer than 2 minutes. The bottom line is that you don't have time to waste.

Don't let yourself become bogged down on a question. You know how it goes: You're so close to getting the answer, and you've already put in the time to trace the code, and maybe you made an error in your trace and should do it again . . . Meanwhile the clock is ticking. If a given question stymies you, circle it and move on. You can always come back to it if you have time at the end.

Guessing

There's no penalty for guessing. If you don't know the answer to a question and are ready to move on, eliminate the answer choices that are clearly wrong, and guess one of the remaining choices.

When time is almost up, bubble in an answer for each of your unanswered questions. Remember, you should make random guesses rather than leaving blanks.

The Java Quick Reference

You will have access to the Java Quick Reference for both the multiple-choice and free-response sections of the AP exam. You should familiarize yourself with it ahead of time (see p. xiii on how to find it), so that you don't waste time searching for something that isn't in it.

The quick reference contains specifications for methods and constants from the Java library for the `Object`, `Integer`, `Double`, `String`, and `Math` classes. Also included are methods from the `List<E>` interface. Each of the methods provided may appear in multiple-choice questions on the AP exam.

Additionally, the quick reference provides the following algorithms in their entirety: insertion sort, selection sort, merge sort, sequential search, and binary search. Nevertheless, you should have the mechanics of these algorithms at your fingertips, as well as preconditions, postconditions, and the best and worst cases. The last thing you want is to be peering through the code in the quick reference when you're asked, for example, which sorting algorithm uses a temporary array. (It's merge sort!)

An Active Pencil

You will not be given scratch paper but you may write on the exam booklet. Don't trace tricky algorithms in your head. Here are some active-pencil tips:

- For each iteration of a loop, write down the values of the loop variables and other key variables. Often, just a few values on the page will reveal a pattern and clarify what an algorithm is doing.
- When you trace an algorithm that has a method with parameters, draw memory slots to visualize what's happening to the values. Remember, when a method is called, copies are made of the actual parameters. It's not possible to alter actual values, unless the parameters are references. (See p. 115.)
- To find a value returned by a recursive algorithm, write down the multiple method calls to keep track. (See, e.g., Question 5 on p. 325.)
- In a complicated question about inheritance, sketch a quick UML (inheritance) diagram to reinforce the relationships between classes. (See p. 226.)
- In the multiple-choice section, questions like the following occur frequently:
  - What does this algorithm do?
  - Which could be a postcondition for ...?
  - Which array will result from ...?
  - Which string will be returned?

  The key to solving these easily is to think *small*. Give yourself a 3-element array, or a 2 × 2 matrix, or a 3-character string, and hand execute the code on your manageable

little data structure. Write down values. Keep track of the loop variables. And the answer will be revealed.

Troubleshooting—What's Wrong with This Code?

Some multiple-choice questions tell you that the code doesn't work as intended. You are to identify the incorrect replacement for `/* more code */`, or simply find the error. Here are some quick details you can check:

- If it's a recursive algorithm, does it have a base case?
- Does the base case in a recursive algorithm actually lead to termination?
- In an array algorithm, can the index equal `arr.length`? If so, it is out of bounds.
- Does the algorithm contain `arr[i]` and `arr[i + 1]`? Often `arr[i + 1]` goes out of bounds.
- In a string algorithm that has `str.substring(start)`, is `start` greater than `str.length()`? If so, it is out of bounds.
- In a string algorithm that has `str.substring(start, end)`, is `start` greater than or equal to `end`? If so, it will cause an error. Is `end` greater than `str.length()`? If so, it is out of bounds.
- In an `ArrayList` method other than `add`, is the index value greater than or equal to `list.size()`? If so, it is out of bounds.
- Is a client program of `SomeClass` trying to directly access the private instance variables or private methods of `SomeClass`? It will cause an error.
- Is the keyword `super` being used in a constructor? If so, it had better be in the first line of code; otherwise, it will cause a compile-time error.
- If a value is being inserted into a sorted list, is there a range check to make sure that the algorithm doesn't sail off the end of the list?

Loop Tracing

Here are some tips:

- There are several questions that will ask how many times a loop is executed. This can be phrased in different ways: How many times will a word or phrase be printed? How many times will a method in the loop body be executed? If the numbers are small, write down all of the values of the loop variable for which the loop will be executed, and count them! That's the answer.
- If the answer to a question with a loop depends on a parameter n, try setting n to 2 or 3 to see the pattern.
- Be sure to pay attention to whether the loop test uses < or <= (or > or >=).
- Watch out for how the loop variable is being changed. It is not always `i++` (increment by 1) or `i--` (decrement by 1).

Java Exceptions

Occasionally, one of the answer choices is the following:

The code throws `<some kind of run-time exception>`.

Run your eye down the algorithm and check for:

- An array, `ArrayList`, or `String` going out of bounds. Each situation will throw an `ArrayIndexOutOfBoundsException` or `IndexOutOfBoundsException`. If you find it, you can move on without even glancing at the other answer choices.
- Integer division by zero. If division by zero occurs in the first part of a compound test, an `ArithmeticException` will always be thrown. If the division is in the second part of the test, you won't get the exception if the value of the whole test is already known (short-circuit evaluation). (See p. 76.)

Matrix Manipulation

Suppose you are given a matrix and an algorithm that changes the matrix in some way. These algorithms are often hard to trace, and, to complicate the problem, you don't have much time.

Assume that `row` and `col` are loop variables in the algorithm. For a quick solution, try looking at the element that corresponds to the first `row` value and first `col` value of the loop. Often this is the element in the top left-hand corner. Does it change? Can you eliminate some of the answer choices, based on this one element? Repeat the process for the final `row` and `col` values. Often this is the element in the bottom right-hand corner. Can you eliminate other answer choices?

In other words, you don't always need to trace all of the values to find the answer.

Comparing Algorithms

Several questions may ask you to compare two algorithms that supposedly implement the same algorithm. They may ask you which algorithm is more efficient. Factors that affect efficiency are the number of comparisons and the number of data moves. Check out the body of each loop. Count the comparisons and data movements. Multiply the total by the number of iterations of the loop. The algorithm with the lower answer is the more efficient algorithm.

You may be asked to compare data structures. These are the questions to ask yourself:

- Which involves the least number of comparisons to locate a given element?
- Which involves the least number of data moves to insert or remove an element?
- Does the structure store the entire state of the object?

Mechanics of Answering Questions

Here are three final tips:

- Take care that you bubble in the answer that corresponds to the number of the question you just solved!
- Since the mark-sense sheet is scored by machine, make sure that you erase completely if you change an answer.
- Take a moment at the end to check that you have provided an answer for every question.

THE FREE-RESPONSE SECTION

What Is the Format?

- You will have 90 minutes for 4 free-response questions.

- Each question is worth 9 points. Note that the free-response and multiple-choice sections carry equal weight in determining your AP score.
- The code that you write must be in Java.
- Write your solutions to the questions on the test booklet provided, underneath the specification for each part of a question.
- You should use a No. 2 pencil for your solutions.

What Is Tested?

In theory, the entire AP Java subset is fair game for free-response questions. But you should pay special attention to each of the following:

- List manipulation using both arrays and `ArrayLists`
- String manipulation
- Two-dimensional arrays
- Classes, subclasses, abstract classes, and interfaces

What Types of Questions?

You may be asked to do each of the following:

- Write the body of a specified method
- Write the bodies of methods using code from previous parts of the question
- Write an overloaded method (p. 110)
- Write a constructor (p. 147)
- Provide an overridden method in a subclass (p. 147)
- Write a complete class—abstract or concrete, subclass, or interface
- Make an informal comparison of running times for pieces of code
- Comment on data structures used in a question

The Java Quick Reference

You will continue to have access to the Java Quick Reference for the free-response section of the AP exam. Here are tips for using it in the code you write:

- When you use a library method from the `List<E>` interface, or from the `Object`, `String`, or `Math` classes, a glance at the first page of the quick reference will confirm that you're using the correct method name and also the correct format and ordering of parameters.
- The correct format of the constants for the smallest and largest `Integer` values (`Integer.MIN_VALUE` and `Integer.MAX_VALUE`) are shown on the first page.
- If you want to use a divide-and-conquer algorithm in your solution, you could model it on the binary search algorithm that's provided in the quick reference.
- Similarly, if you want to use an algorithm in your solution that's similar to selection sort, for example, you could model it on the selection sort code that's provided in the quick reference.
- Again, do not study the quick reference for the first time during the AP exam!

Time Issues

Here are some tips for managing your time:

- Just 90 minutes for 4 questions means fewer than 23 minutes per question. Since many of the questions involve up to two pages of reading, you don't have much time. Nevertheless, you should take a minute to read through the whole exam so that you can start with a question you feel confident about. It gives you a psychological boost to have a solid question in the bag.
- Work steadily through the remaining questions. If you get stuck on a question, move on. If you can answer only one part of a question, do it and leave the other parts blank. You can return to them if there's time at the end.
- Don't waste time writing comments: the graders generally ignore them. The occasional comment that clarifies a segment of code is OK. I know this because I graded AP Computer Science exams for many years.

Grading the Free-Response Questions

Be aware that this section is graded by humans. It's in your interest to have graders understand your solutions. With this in mind:

- Use a sharp pencil, write legibly, space your answers, and indent correctly.
- Use self-documenting names for variables, methods, and so on.
- Use the identifiers that are used in the question.
- Write clear, readable code. Avoid obscure, convoluted code.

The graders have a rubric that allocates each of the 9 points for solving the various parts of a problem. If your solution works, you will get full credit. This is irrespective of whether your code is efficient or elegant. You should never, ever take time to rewrite a working piece of code more elegantly.

You will be awarded partial credit if you used the right approach for a question, but didn't quite nail the solution. Each valid piece of your code that is included in the rubric will earn you some credit.

There are certain errors that are not penalized. For example, the graders will look the other way if you omit semi-colons, or misspell identifiers, or use keywords as identifiers, or confuse `length` with `length()`, or use = instead of ==, or use the wrong kind of brackets for an array element.

They will, however, deduct points for each of the following types of errors:

- Including output statements in methods that don't require it.
- Including an output statement in a method, instead of returning a value. (Are you clear on this? No `System.out.print()` or `System.out.println()` statements in methods that didn't ask for output!).
- Using local variables that aren't declared.
- Returning the wrong type of value in a method, or returning a value in a `void` method, or having a constructor return a value.
- Using incorrect syntax for traversing and processing arrays and `ArrayLists`.

Take heart: On any given question, you won't receive a negative score!

Coding Issues

Here are important reminders for the code that you write:

- It's worth repeating: You must use Java for your solutions. No pseudo-code, or C++, or any other programming language.
- If the statement of the question provides an algorithm, you should say thank you and use it. Don't waste time trying to reinvent the wheel.
- Don't omit a question just because you can't come up with a complete solution. Remember, partial credit is awarded. Also, don't omit part (b) of a question just because you couldn't do part (a)—the various parts of a question are graded independently.
- In writing solutions to a question, you must use the public methods of classes provided in that question wherever possible. If you write a significant chunk of code that can be replaced by a call to one of those methods, you will probably not receive full credit for the question.
- If you're writing a subclass of a given superclass, don't rewrite the public methods of that class unless you are overriding them. All public methods are inherited.
- It is fine to use methods from the Java library that are not in the AP Java subset. You will receive full credit if you use those methods correctly. If your usage, however, is incorrect, your solution will be penalized. Note that there is always a solution that uses the subset, and you should try to find it.
- It is fine to write a helper method for your solution. But be aware that usually you can solve the question using methods that are already provided.

Maximizing Your Score

Here are some final general tips for maximizing your free-response score:

- At the start of the free-response section clear the decks! The multiple-choice section is over, and whether you killed it or it killed you, it is past history. Take a deep breath and psych yourself up for the code-writing part of the exam. Now is the time to strut your stuff.
- Don't ever erase a solution. Cross it out. You want to avoid a situation in which you erased some code, but don't have time to write the replacement code.
- If you can't see a solution to a question, but have some understanding of what is required to solve it, write something down. Just showing that you understand that you must loop through all of the elements in a list, for example, may earn you a point from the rubric.
- Don't provide more than one solution for a given part. The exam readers are instructed to grade the first attempt only, and to ignore all subsequent ones. Not even the most warm-hearted among us will go wading through the marshes of your various solutions, searching for the one that solves the problem correctly. This means that you must choose one solution and cross out the others with firm, legible lines.
- One final reminder: Use clear, readable code from the AP Java subset. Avoid obscure, opaque brilliancies. The AP exam is not the place to show the world you're a genius.

Good luck!