08 Module Quiz

- Due Oct 25 at 11:59pm
- Points 15
- Questions 10
- Available until Oct 25 at 11:59pm
- Time Limit None
- Allowed Attempts Unlimited

Instructions

- **Purpose:** To *review and reinforce your understanding* of key concepts from the week. They are designed as a low-stakes way to assess your mastery and identify topics to revisit before exams.
- **Task:** At the end of each module, complete the quiz in Canvas. You may take each quiz as many times as you like before the deadline. Each attempt may show different questions, drawn from a larger question bank. Your highest score by the deadline will be recorded.
- **Criteria:** Full credit is earned by completing at least 80% of the weekly quizzes on time. Quizzes are unproctored and open-resource, but intended for individual review. Use them to reflect on what you've learned and where you may need more practice.

This quiz was locked Oct 25 at 11:59pm.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	32 minutes	15 out of 15

ⓘ Correct answers are hidden.

Score for this attempt: 15 out of 15
Submitted Oct 21 at 9:06pm
This attempt took 32 minutes.

Question 1
1 / 1 pts
Which of the following Big-O time complexities corresponds to the **LEAST** efficient algorithm?

- O(N)
- O(N^2)
- O(1)

Question 2
1 / 1 pts

Recursive implementations are frequently **less space-efficient** than their iterative counterparts because:

- every recursive call requires a new ***activation record***/***stack frame*** on the call stack
- all ***data structures*** in a recursive solution are duplicated on the heap

Question 3

1 / 1 pts

The case where a recursive method does NOT call itself is called the _____.

- recursive case
- base case

Question 4

1 / 1 pts

Given that `(char)65` gives `'A'`, what line should be added to the following method so that a call of `recursiveAlpha(65)` returns the String `"ABCDE"`?

```
public static String recursiveAlpha(int n) {
  // ** ADD LINE HERE **
  return "" + (char)n + recursiveAlpha(n+1);
}
```

- `if (n < 70) return "";`
- `if (n == 70) return "";`

Question 5

1 / 1 pts

Given the following recursive method, which value for `ARG` will give a method which eventually terminates (for ***all*** values of n)?

```
public static int recurse(int n) {
  if (n <= 0) return 50;
  return n + recurse( ARG );
}
```

- n - 1
- n + 1

Question 6

2 / 2 pts

In which of the following cases will a StackOverflowError be thrown from a recursive method when it runs? Select all which apply.

- [ ] The recursive case reduces the problem to a smaller version of the original problem.
- [ ] Only one base case is defined.
- [x] No base case is defined.
- [x] The recursive case does not make progress towards at least one base case.

Question 7

2 / 2 pts

What is the runtime complexity of the following ***getMaximum()*** method, assuming that the problem size `N` is the input parameter `size` (the number of elements stored in the array data)?

```
/**
 * Recursive method which returns the maximum element of an oversize array
 * @param data a non empty oversize array
 * @param size number of elements stored in an oversize array
 * @return the maximum element of the oversize array data
 */
public static int getMaximum(int[] data, int size) {
  if (size == 1)
    return data[size-1];
  return Math.max(data[size-1], getMaximum(data, size-1));
}
```

- ( ) O(N^2)
- (•) O(N)
- ( ) O(1)
- ( ) O(2^N)

Question 8

2 / 2 pts

Which of the following statement is **TRUE?**

- ( ) Iteration is ALWAYS **more efficient AND simpler** than recursion.
- (•) It is ALWAYS **possible to write the iterative version** of a recursive algorithm.
- ( ) Recursion is ALWAYS **more efficient** than iteration.
- ( ) Recursion ALWAYS **uses less memory** compared to iteration when it executes.

Question 9

2 / 2 pts

What is the runtime complexity of the following **addUsername()** method, assuming that the problem size N represents the number of elements stored in the array ***users*** provided as input?

```java
  /**
   * Adds a new username to the end of a list of users defined by the oversize array (users, si
ze)
   * @param users an oversize array which stores a set of usernames
   * @param size number of elements stored in the users array
   * @param username to add
   * @return the new size of the list of users after adding a new username
   */
  public static int addUsername(String[] users, int size, String username) {
    if(size < users.length) {
      users[size] = username;
      size++;
    }
    return size;
  }
```

- O(N^2)
- O(1)
- O(N)
- O(log(N))

Question 10

2 / 2 pts

What is the worst-case runtime complexity of the following **addPassword()** method, assuming that the problem size N represents the number of elements stored in the array ***passwords*** provided as input?

```java
  /**
   * Adds a new password to the end of a list of passwords defined by the oversize array (passw
ords, size)
   * @param passwords an oversize array which stores a set of users' passwords
   * @param size number of elements stored in the array passwords
   * @param password to add
   * @return the new size of the list of users' passwords after adding a new password
   */
  public static int addPassword(String[] passwords, int size, String password) {
    for(int i = 0; i < passwords.length; i++) {
      if(passwords[i] == null) {
        passwords[i] = password;
        size++;
        break;
      }
    }
    return size;
  }
```

- O(1)
- O(N^2)
- O(log(N))
- O(N)

Quiz Score: 15 out of 15