Ozyegin University
CS 321 Programming Languages
Sample Problems on Garbage Collection

1. Which garbage collection algorithm would work the best for the following piece of Java program? Explain why. Remember that Strings in Java are immutable objects (i.e. once created, you may not modify them).

```
String concat(String[] ss) {
  // Assume ss is very long
  String res = "";
  for (int i = 0; i < ss.length; i++)
      res += ss[i];
  return res;
}
```

   Suggest a better (i.e. more efficient) way to implement this code. Explain why it would be more efficient.

   Solution: This program creates a lot of short-lived objects. Each iteration of the loop creates a String object that becomes garbage in the next iteration. For this program, "two space stop and copy" probably works the best because this algorithm does not need to touch the many String objects that die young. Also, these young and dead objects may have caused fragmentation in the heap. "Two space stop and copy" fixes that problem as well.

   (Automatic) reference counting is likely to perform well, too. This is because there are no cyclic links between the dead objects. Furthermore, there are no links to other objects from the String objects; hence, deallocation of a String object does not cause a ripple effect.

2. Write a piece of program (in Java or C++) that would cause a memory leak if reference counting were used as the garbage collection technique.

   Solution:

```
class Node {
  int item;
  Node next;
}
...
Node node1 = new Node();
Node node2 = new Node();
node1.next = node2;
node2.next = node1;
node1 = null;
node2 = null;
...
```

   The code above creates a cycle between two objects, and then loses the pointers to these objects from the stack.

3.

(a) Suppose I have a program where I create a long linked list structure with no cycles, and then set my pointer to the head of the list to null, so the head becomes unreachable. Which garbage collection algorithm would perform the best for this case? Justify by explaining what drawbacks the other algorithms have for this case.

(b) Which garbage collection algorithm is likely to improve cache utilization? Why?

(c) Suppose I have a computer with small heap memory. If I am to make a choice between mark-and-sweep and two-space-stop-and-copy algorithms, which one should I choose? Why?

(d) Suppose I have a program where I create a lot of temporary (i.e. short-lived) objects of various sizes scattered (i.e. *saçılmış, yayılmış*) all over the memory. If I am to make a choice between mark-and-sweep and two-space-stop-and-copy algorithms, which one should I choose? Why?

(e) Suppose I have a program where I create many long-lived and large objects and arrays. If I am to make a choice between mark-and-sweep and two-space-stop-and-copy algorithms, which one should I choose? Why?