5. (3 points) Consider the function below:

```
void bus(long *i, long *j) {
  // Find the element in the middle of *i and *j in the array.
  long mid = ________________;
}
```

The function is called with

```
long a[100];
bus(&a[0], &a[99]);
```

The variable `mid` should be initialized to the middle element in the array (i.e., `a[49]`). How should `mid` be initialized?

A. `*((i + j) / 2)`
B. `(*i + *j) / 2`
C. `*(i + j) / 2`
D. `*(i + (j - i) / 2)`
E. `*i + (*j - *i) / 2`

Shade X on the answer sheet if none of the choices above is correct.

> **Solution:** D.
>
> This is a challenging question that assesses if students are aware of how pointer arithmetic operates. Not surprisingly, only 23% of students managed to answer this.
>
> Most students picked A. However, `i` and `j` are not indices to an array, but pointers. The addition of two pointers is not a meaningful arithmetic operation. Pointer arithmetic only supports adding offsets to a pointer and calculating offsets between two pointers (which is what Option D does).

6. (3 points) Consider the function below:

```
void doh(long n) {
  long *a;
  for (long i = 0; i < n; i += 1) {
    a = malloc(2 * sizeof(long));
    if (a == NULL) {
      cs1010_println_string("Not enough memory");
    }
    a[0] = 1;
  }
  free(a);
}
```

Which of the following statements about the function above is true? Assume that `n` > 1.

(i) There is a memory leaks if `malloc` is successful.

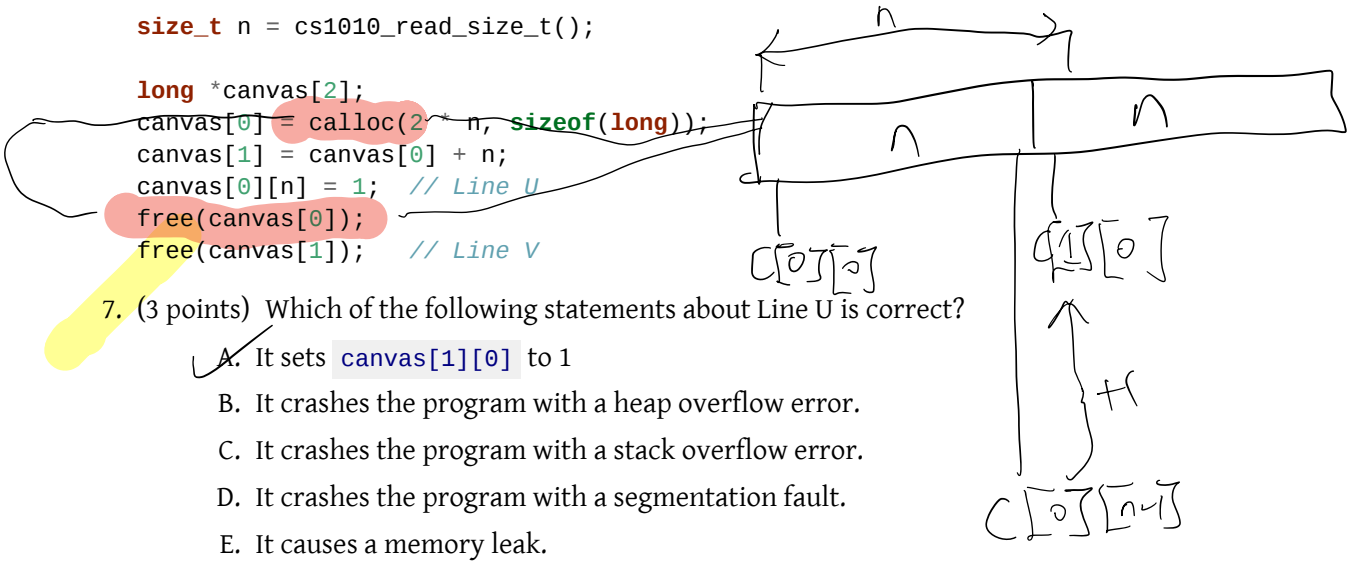
(ii) There is an illegal access to memory if `malloc` is successful.

(iii) There is an illegal access to memory if `malloc` is not successful.

For Questions 7 and 8, consider the code snippet below, which allocates a 2D array with two rows and `n` columns. We assume that `calloc` does not fail and `n` is a positive number.

```
size_t n = cs1010_read_size_t();

long *canvas[2];
canvas[0] = calloc(2 * n, sizeof(long));
canvas[1] = canvas[0] + n;
canvas[0][n] = 1;  // Line U
free(canvas[0]);
free(canvas[1]);   // Line V
```



7. (3 points) Which of the following statements about Line U is correct?

   A. It sets `canvas[1][0]` to 1

   B. It crashes the program with a heap overflow error.

   C. It crashes the program with a stack overflow error.

   D. It crashes the program with a segmentation fault.

   E. It causes a memory leak.

   Shade X on the answer sheet if none of the choices above is correct.

> **Solution:** A.
>
> The given code snippet allocates a continuous region of memory for two rows of integers. Each row contains `n` integers (indexed from `0` to `n - 1`. Line U attempted to update the element at index `n` of Row 0, i.e., `*(canvas[0] + n)`. This deferences `canvas[1][0]` instead.
>
> About a third of students chose the correct answer.

8. (3 points) Which of the following statements about Line V is correct?

   A. It correctly ensures that there is no memory leak.

   B. It causes a memory leak.

   C. It crashes the program with a stack overflow error.

   D. It crashes the program since the memory pointed to by `canvas[1]` is not at the start of a memory region allocated with `calloc`.

   E. It crashes the program since the memory pointed to by `canvas[1]` is allocated on the stack.

   Shade X on the answer sheet if none of the choices above is correct.

> **Solution:** D.
>
> Since `canvas[1]` is only a pointer pointing to a region of memory allocated to `canvas[0]`, we are not supposed to free `canvas[1]`. Doing so would cause the program to crash.
>
> Only 45% of the students chose the correct answer.

14. (14 points) Consider the algorithm below:

```
void swap(long a[], size_t i, size_t j) {
  long temp = a[i];
  a[i] = a[j];
  a[j] = temp;
}

void bar(long a[], size_t n) {
  size_t i = 0;
  while (i < n - 1) {
    // Line P
    if (a[i] > a[i + 1]) {
      swap(a, i, i + 1);
      i = 0;
    } else {
      i += 1;
    }
    // Line Q
  }
}
```

(a) (6 points) Trace through what happens to the array `a[3] = {5, 1, 2};` if we call `bar(a, 3)`, by writing down the values of `i` and content of `a` at the end of each iteration (i.e., at Line Q).

Note that you may not need to fill in all the rows in the given table on the answer sheet.

Solution:

```
i=0, a=5 1 2
i=0, a=1 5 2
i=1, a=1 5 2
i=0, a=1 2 5
i=1, a=1 2 5
i=2, a=1 2 5
```

1 mark for each iteration. The result after the first iteration is already given so you get 1 mark for free!

(b) (4 points) The invariant of the loop is the following: "`a[0]..a[i]` is sorted in non-decreasing order." Assuming that this assertion holds at Line P, explain why it also holds at Line Q.

Solution: if `a[i] < a[i+1]`, then the sequence `a[0]..a[i+1]` is sorted. We increase `i` by 1. So the assertion `a[0]..a[i]` is sorted is true. Otherwise, `i` becomes 0. We have only one element that is trivially sorted.

For marking,

- 2 marks each for explaining why assertion holds for "if" and "else" branch.
- -1 mark if students did not explicitly show that the "if" branch, assertion holds because the new list contains just one element and is trivially sorted.
- -1 mark if students did not show that after i+=1, the assertion reverts from a[0]...a[i+1] is sorted back to a[0]...a[i].

Some common mistakes include (i) not explaining the general case, (ii) saying that because the assertion is true at line P, the "if" loop will never execute.

(c) (4 points) Express the running time of this algorithm using the big-O notation in terms of n. Briefly explain your answer.

Solution: $O(n^3)$. In the worst case, we have $O(n^2)$ pairs of elements not in order. Each scan $O(n)$ fixes one such pair.

To get full marks, students must give the right big O and provide the right explanation. If a student gives the wrong big O, at most 1 mark can be given if the student identifies the worst case correctly as being decreasing/non-increasing/inversely sorted. A common mistake is that most students say is $O(n^2)$, but don't forget this is not exactly like bubble sort because we reset i to 0 after a swap, so that incurs an extra $O(n)$ factor.

9. (3 points) What is the running time of the function below in terms of n, the size of the input array?

```
void waldo(long a[], long n) {
  long k = n;
  while (k > 0) {
    for (long i = 0; i < n; i += 1) {
      a[i] += (k % 10);
    }
    k /= 10;
  }
}
```

A. $O(10^n)$
B. $O(n^{10})$
C. $O(\log n)$
D. $O(n)$
E. $O(n \log n)$

Write X in the answer box if none of the choices above is correct.

Solution: E.

This question assesses if students are able to analyze the big-O running time of a given code.

There is a nested loop here. The inner `for` loop runs $O(n)$ times for each `k`. The outer `while` loop keeps dividing `k` by 10 until it reaches 0, so it runs $O(\log n)$ time. The total running time is thus $O(n \log n)$.

10. (3 points) The running time of a recursive algorithm can be characterized by the following recurrence relation:

$$T(n) = \begin{cases} T\left(\frac{n}{2}\right) + \log n & \text{if } n > 1 \\ 1 & \text{otherwise} \end{cases}$$

What is $T(n)$?

A. $O(1)$
B. $O(\log \log n)$
C. $O(\log n)$
D. $O(\log^2 n) = O(\log n \cdot \log n)$
E. $O(n)$
F. $O(n \log n)$

Write X in the answer box if none of the choices above is correct.

Solution: D.

This question assesses if students are able to solve a recurrence relation. We have

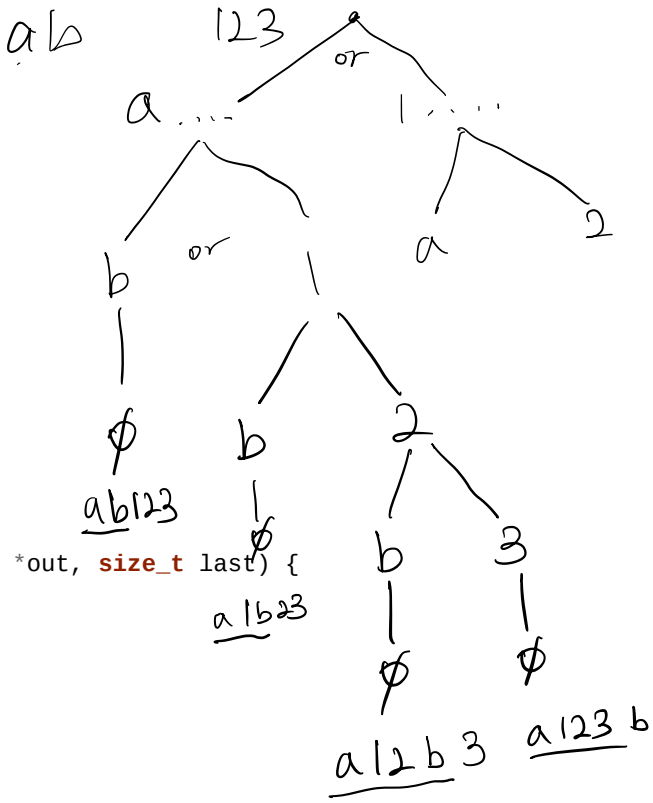
$$\begin{aligned} T(n) &= T(\frac{n}{2}) + \log n \\ &= T(\frac{n}{4}) + \log \frac{n}{2} + \log n \\ &= T(\frac{n}{8}) + \log \frac{n}{4} + \log \frac{n}{2} + \log n \\ &= T(\frac{n}{2^i}) + \log \frac{n}{2^{i-1}} + ... + \log \frac{n}{2} + \log n \end{aligned}$$

When $\frac{n}{2^i}$ becomes 1, i is $\log n$. So we have

$$\begin{aligned} T(n) &= T(1) + \sum_{k=0}^{\log n} \log \frac{n}{2^{k-1}} \\ T(n) &= 1 + \sum_{k=0}^{\log n} (\log n - \log 2^{k-1}) \\ &= \log n \log n - \sum_{k=0}^{\log n} (k-1) \log 2 \\ &= O(\log n \log n) \end{aligned}$$

19. (6 points) In this question, we want to write a function that print out all possible string as the result of interleaving characters from two given strings. The relative order of the characters from the same string must be preserved in the interleaved string. For instance, if the input strings are ab and 123, then our function should print:

```
ab123
a1b23
a12b3
a123b
1ab23
1a2b3
1a23b
12ab3
12a3b
123ab
```



Part of the code of interleave has been given:

```c
#include "cs1010.h"

void interleave(char *s, char *t, char *out, size_t last) {
  if (s[0] == '\0') {
    out[last] = '\0';
    cs1010_print_string(out);
    cs1010_println_string(t);
    return;
  }

  if (t[0] == '\0') {
    out[last] = '\0';
    cs1010_print_string(out);
    cs1010_println_string(s);
    return;
  }

  // COMPLETE THE CODE HERE

}
```

The function `interleave` is a recursive function that takes in two strings `s` and `t`. The string `out` is a string that stores an interleaving of characters from `s` and `t`. You may assume that `out` has been properly allocated. The 4th parameter `last` stores the total number of characters from either `s` or `t` (or both) that has been interleaved and stored in `out`.

The base case has been written for you. Here, we reach the base case if either `s` or `t` is an empty string. Since we are out of characters to interleave, we simply print what we constructed so far (out) followed by the remaining characters from the other string.

The first call to the function interleave looks like this:

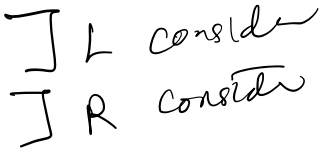
```
interleave(s, t, out, 0);
```

Complete the implementation of the function interleave. There should be at most four lines of code. Write `;` in the blank below if your solution has less than four lines and you want to leave the blanks empty. You may print out the interleaving strings in any order (not necessary the same order as the example above).

Hint: if `s` is a string, then `s+1` is the rest of s without the first character.

Solution:

```
out[last] = s[0];
interleave(s + 1, t, out, last + 1);
out[last] = t[0];
interleave(s, t + 1, out, last + 1);
```



Students did better than I expected here. Bravo!! 73 students scored full marks. 37 students get 5 marks.

Partial marks are given as long as students have the recursion structure: the answer contains two interleave calls but may have other bugs (such as initializing out incorrectly, mixed up +1 with other increments, etc) . Students who included loops, branching, 1/3/4 interleave calls, are given 0 marks.