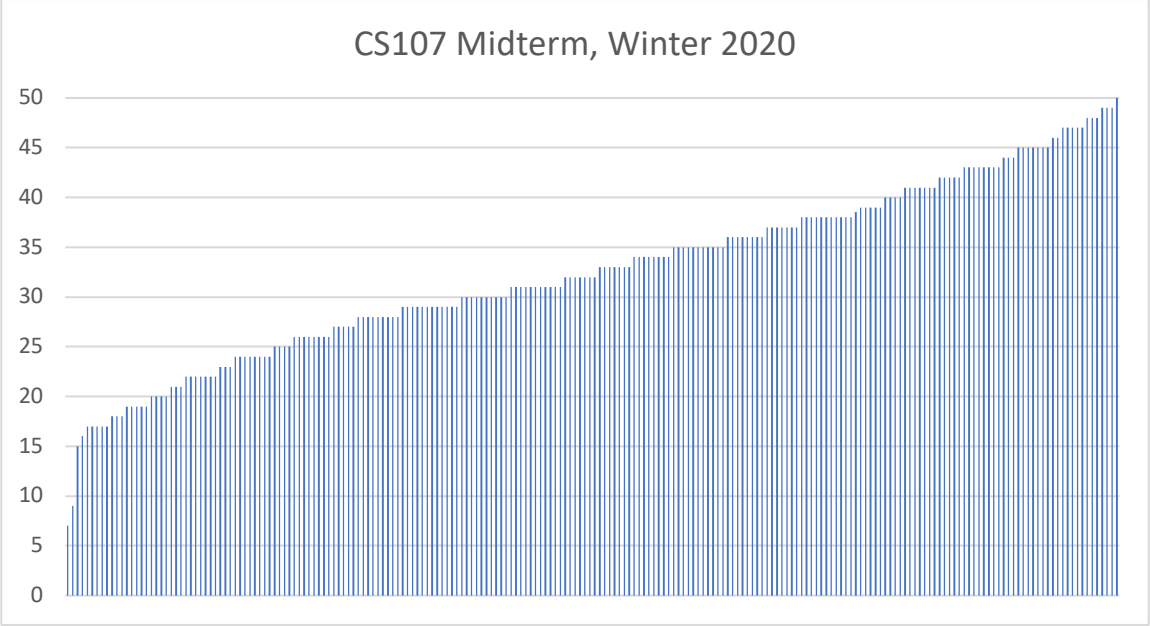
# CS107 Midterm Examination Solution

Your midterms are graded, and we'll be publishing the grades through Gradescope by the end of the day.

The exam was challenging, and every problem was intentionally designed to drill you on something you learned in CS107. That meant that for loops, clever if tests, and recursion didn't come in to play, but C string manipulation, generic programming, and bit-level calisthenics certainly did. The median grade was a 32.0 out of 50, and scores ranged from 7 all the way up to a perfect 50. Here's a graph illustrating how the class did (each vertical bar represents a single student exam, and everything's sorted low to high.




Read over your graded exam sooner than later, compare your answers to our solution, and read through our comments on Gradescope maximize learning. ☺ The grading criteria we used for each of the five problems is exposed on Gradescope, so you should be able to understand why we graded the way we did. If you see something you clearly did right but didn't get recognized for it, submit a regrade request through Gradescope and the person who graded your work will review it.

**Problem 1: Bytes, Bytes, and Numbers**

```
unsigned int amuse(unsigned int v) {
   unsigned int count = 0;
   v ^= v - 1;
   v >>= 1;
   while (v) { v >>= 1; count++; }
   return count;
}
```

a)

3
0
11
5

b) For which values of **v** does **amuse(v)** return zero?

All odd values.

c) When **amuse** returns a nonzero value for nonzero **v**, what is the return value computing?

It returns the number of trailing zeros in the binary representation of **v**.

d) What does **amuse(0)** return?

31

**Problem 2: C Strings**

```
/**
 * Function: parse_tokens
 * ----------------------
 * Assumes the incoming command is a string of one or more
 * tokens, and parses the command in place as described in the
 * problem statement.  At the beginning of the loop, it's assumed
 * that command is addressing the first letter of a token.  We
 * store that address in tokens, advance the count by one, and then
 * skip over all non-space tokens using strcspn.  If after the
 * skip we're staring at a '\0', then we're done.  Otherwise, we're
 * starting at a single space character, which we overwrite with
 * a '\0' before advancing to the next character (which is the
 * first character of the next token).
 *
 * My solution handles the scenario where command is the empty string,
 * but your solution didn't need to worry about that.
 */
```

```
size_t parse_tokens(char command[], char *tokens[]) {
    size_t count = 0;
    while (*command != '\0') {
        tokens[count++] = command;
        size_t n = strcspn(command, " ");
        command += n;
        if (*command == '\0') break;
        *command = '\0';
        command++;
    }
    return count;
}
```

## Problem 3: Serializing String Lists

```
/**
 * Function: serialize
 * -------------------
 * Implements the serialization algorithm as described in the
 * the midterm.  See inline comments for the play-by-play on how
 * to crawl over a list like that described on the midterm.
 */
void *serialize(const void *list) {
    void *memory = malloc(sizeof(size_t));
    assert(memory != NULL);
    size_t length = sizeof(size_t);
    *(size_t *)memory = 0;
    // everythng above is necessary, even when the incoming list is empty

    while (list != NULL) {
        char *chars = (char *)list + sizeof(void *);  // chars are eight bytes in
        size_t num_bytes = strlen(chars) + 1;         // additional bytes needed
        memory = realloc(memory, length + num_bytes); // extend/move memory
        assert(memory != NULL);
        strcpy((char *) memory + length, chars);      // fill in new bytes
        length += num_bytes;                          // compute new length
        (*(size_t *)memory)++;                        // promote size
        list = *(void **)list;                        // drill forward
    }
    return memory;
}
```

## Problem 4: Using `binsert`

```
/**
 * Function: entry_cmp
 * -------------------
 * Accepts two entry *'s disguised as void *s, and compares
 * their strings.
 */

int entry_cmp(const void *vp1, const void *vp2) {
    const entry *ep1 = vp1;
    const entry *ep2 = vp2;
    return strcmp(ep1->word, ep2->word);
}

/**
 * Function: process_word
 * ----------------------
 * process_word is a thin wrapper around binsert, which either inserts a new
 * entry on behalf of the supplied word or surfaces an existing one.  In either
 * case, binsert returns the address of the entry housing word.
 *
 * * If found->word matches e.word (i.e. the pointers match), then binsert
 *   inserted a new record into place.  We need displace the copy of
 *   the word pointer with the address of a heap-based snapshot of word.
 * * If found->word doesn't match e.word, then the word was already present, and
 *   we just need to increment its frequency.
 */
void process_word(entry entries[], size_t *p_nelem, char word[]) {
    entry e = {word, 1};
    entry *found = binsert(&e, entries, p_nelem, sizeof(entry), entry_cmp);
    if (found->word == e.word) {
        found->word = strdup(word); // deep copy displaces shallow one
    } else  {
        found->freq++; // e was not copied in, increment existing freq by 1
    }
}
```

## Problem 5: Function Pointers and Generic Memory Moves

```
void *nth_elem(void *base, size_t n, size_t width) {
    return (char *) base + n * width;
}

size_t uniquify(void *base, int nelems, int width,
                int (*cmp)(const void *, const void *)) {

    int nretained = nelems;
    for (size_t i = nelems - 1; i >= 1; i--) {
        void *p1 = nth_elem(base, i, width);
        for (ssize_t j = i - 1; j >= 0; j--) {
            void *p2 = nth_elem(base, j, width);;
            if (cmp(p1, p2) == 0) {
```

```
                void *dest = p1;
                void *source = (char *) p1 + width;
                size_t num_elems_to_move = nretained - i - 1;
                memmove(dest, source, num_elems_to_move * width);
                nretained--;
                break;
            }
        }
    }

    return nretained;
}
```