Prep 9 Quiz

Quiz Instructions

Readings

Please read the following part of the Course Notes (https://www.teach.cs.toronto.edu/~csc165h/winter/resources/csc165_notes.pdf).

- Chapter 5, pp. (bottom of) 102-107.

General instructions

You can review the general instructions for all prep quizzes at this page (https://www.teach.cs.toronto.edu/~csc165h/winter/homework/index.html). Remember that you can submit multiple times! We have posted a PDF version of the quiz on the course website. You might consider printing this quiz out so that you can work on paper first.

### Question 1

Let $S \subseteq \mathbb{R}$ be a non-empty finite set of real numbers, and let $M \in \mathbb{R}$.

Which of the following expressions is equivalent to the English statement "$M$ is an upper bound on the maximum value of $S$"?

- $S \leq M$
- $\forall x \in S, x < M$
- $\forall x \in S, x \leq M$
- $M \in S \land (\forall x \in S, x \leq M)$
Question 2  

Let $S \subseteq \mathbb{R}$ be a non-empty finite set of real numbers, and let $M \in \mathbb{R}$. Which of the following expressions is equivalent to the English statement "$M$ is a lower bound on the maximum value of $S$"?

- $\exists x \in S, x \leq M$
- $\exists x \in S, x \geq M$
- $M \in S$
- $\forall x \in S, x \geq M$
- $M \leq S$
- $\exists x \in S, x \leq M$

Question 3  

Consider the following function:

```python
def has_duplicates(lst: list) -> bool:
    
    i = 0
    while i < len(lst):
        j = i + 1
        while j < len(lst):
            if lst[i] == lst[j]:
                return True
            j += 1
        i += 1
    return False
```
Select all of the following Big-Oh expressions that are an upper bound on the worst-case running time of `has_duplicates`.

- O(n^3)
- O(n(n + 1))
- O(n)
- O(n^2)
- O(2^n)

Question 4 1 pts

Recall function `has_duplicates` from the previous question:

```python
def has_duplicates(lst: list) -> bool:
    #"Return whether lst contains any duplicate values."
    i = 0
    while i < len(lst):
        j = i + 1
        while j < len(lst):
            if lst[i] == lst[j]:
                return True
            j += 1
        i += 1
    return False
```

Select all of the following Omega expressions that are a lower bound on the worst-case running time of `has_duplicates`.

- Ω(n^3)
- Ω(n^2)
- Ω(n)
Each of the following functions takes in a list of integers. For each function, select whether its running time is asymptotically constant ($\Theta(1)$), depends only on the length of its input list, or depends on both the length of the input list and the values stored in the list.

1. `def has_duplicates(lst: List[int]) -> bool:`
   ```python
   i = 0
   while i < len(lst):
       j = i + 1
       while j < len(lst):
           if lst[i] == lst[j]:
               return True
           j += 1
       i += 1
   return False
   ```

2. `def mod1(lst: List[int]) -> None:`
   ```python
   for i in range(len(lst)):  # Loop 1
       for j in range(i + 1, len(lst)):  # Loop 2
           if (i + j) % 3 == 2:
               # NOTE: this nested break stops Loop 2, but Loop 1 still continues
               break
   ```

3. `def mod2(lst: List[int]) -> None:`
   ```python
   for i in range(len(lst)):
       for j in range(i + 1, len(lst)):
           if (lst[i] + lst[j]) % 3 == 2:
               # NOTE: this nested break stops Loop 2, but Loop 1 still continues
               break
   ```
4. def has_duplicates_bug(lst: List[int]) -> bool:
   i = 0
   while i < len(lst):
      j = i
      while j < len(lst):
         if lst[i] == lst[j]:
            return True
         j += 1
      i += 1

Question 6

One of the algorithms we'll study in lecture this week is string-based, and this question is designed to prepare you for this analysis. Here are two definitions about strings.

**Definition 1 (palindrome).** Let $s$ be a string. We say that $s$ is a palindrome when its reversal (the string obtained by taking the letters of $s$ in reverse order) equals the original string. For example, the string "abcbaba" is a palindrome. The empty string and all strings of length 1 are palindromes.

**Definition 2 (prefix).** Let $r$ and $s$ be strings. We say that $r$ is a prefix of $s$ when $r$ appears at the start of $s$ (in a contiguous block). For example, the string "abc" is a prefix of "abcdef", but it is not a prefix of "abdcef" or "defabc". Every string is a prefix of itself, and the empty string is a prefix of every string.

Now, consider the following algorithm:

def palindrome_prefix_simple(s: str) -> int:
   """Return the length of the longest prefix of s
   that is a palindrome. (Assume s is non-empty.)""
   n = len(s)
   max_length = 0
   for prefix_length in range(1, n + 1):  # goes from 1 to n
      # Check whether s[0:prefix_length] is a palindrome
```python
is_palindrome = True
for i in range(prefix_length):
    if s[i] != s[prefix_length - 1 - i]:
        is_palindrome = False

# If a palindrome prefix is found, update max_length.
if is_palindrome:
    max_length = prefix_length

return max_length
```

Select all of the following Big-Oh expressions that are an upper bound on the worst-case running time of `palindrome_prefix_simple`.

- $O(n^2)$
- $O(n)$
- $O(2^n)$
- $O(n^3)$