DSC 40B - Midterm Review

The questions below are indicative of what you will see on the midterm. However, note that the midterm itself will be longer. The exam will contain around 25 questions, with each taking an average of 2 to 3 minutes. The time limit for the exam will be 90 minutes.

Problem 1.

What is the time complexity of this piece of code?

```
def factorial(n):
    r = 1
    i = n
    while i > 0:
        r = r*i
        i -= 1
    return r
```

- ☐ $\theta(n^2)$
- ☐ $\theta(n^3)$
- ☐ $\theta(nlogn)$
- ☐ $\theta(n)$

Problem 2.

What is the time complexity of this piece of code?

```
def foo(n):
    for i in range(5, 10):
        for j in range(i):
            for k in range(n):
                print(i, j, k)
```

- ☐ $\theta(n^2)$
- ☐ $\theta(n^3)$
- ☐ $\theta(nlogn)$
- ☐ $\theta(n)$

Problem 3.

This piece of code returns the number of "pairs" of the form (x, -x) in a collection of numbers. What is the time complexity of this piece of code if the input is a Python list of size n?

```
def count_pairs(numbers):
    count = 0
    for x in numbers:
        if -x in numbers:
             count += 1/2
    return count
```

- ☐ $\theta(n^2)$

- ☐ $\theta(n^3)$
- ☐ $\theta(nlogn)$
- ☐ $\theta(n)$

Problem 4.

The below code shows the iterative version of binary search.

```
def binary_search(arr, t, start, stop):
    while start < stop:
        middle = start + (stop -start) // 2
        if(arr[middle] == t):
            return middle
        if(arr[middle]<t):
            start = middle + 1
        else:
            stop = middle
```

Let n = stop - start. What is the worst-case time complexity of this version of binary search?

- ☐ $\theta(n^2)$
- ☐ $\theta(logn)$
- ☐ $\theta(nlogn)$
- ☐ $\theta(n)$

Problem 5.

Here is a recursive algorithm for computing the factorial of n:

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n-1)
```

What is the recurrence relation describing this function's run time?

Problem 6.

Suppose $f_1(n) = \Theta(n^3)$ and $f_2(n) = \Omega(n)$. Which is true about the upper bound of $f_1 + f_2$?

- ☐ $f_1(n) + f_2(n) = O(n)$
- ☐ It cannot be determined
- ☐ $f_1(n) + f_2(n) = O(n^3)$

Problem 7.

Suppose $f_1(n) = \Omega(n^2)$ and $O(n^5)$ and $f_2(n) = \Omega(n^3)$ and $O(n^6)$. Which is true about $f_1 + f_2$?

- ☐ It is $O(n^5)$ and $\Omega(n^2)$
- ☐ It is $O(n^6)$ and $\Omega(n^3)$
- ☐ It is $O(n^6)$ and $\Omega(n^2)$
- ☐ It is $O(n^5)$ and $\Omega(n^3)$

Problem 8.

If $f(n) = O(n^2)$, then $f(n) = \Omega(n^2)$

☐ True

☐ False

Problem 9.

If $f(n) = O(n^5)$, and $g(n) = O(n^2)$ then $f(n)/g(n) = O(n^3)$

☐ True

☐ False

Problem 10.

The best case and worst case time complexity of merge sort is $\theta(nlogn)$

☐ True

☐ False

Problem 11.

The recursive calls made by mergesort are always on arrays of strictly smaller size than the input array.

☐ True

☐ False

Problem 12.

Consider the modified mergesort given below:

```
def mergesort(arr):
    if len(arr)>1 :
        middle=math.floor(len(arr)/2)
        left=arr[:middle]
        right=arr[middle:]
        for i in range(len(arr)):
            for j in range(len(arr)):
                print("Mergesort")
        mergesort(left)
        mergesort(right)
        merge(left, right, arr)
```

What is the time complexity of this modified mergesort?

☐ $\theta(n^2)$

☐ $\theta(n^3)$

☐ $\theta(nlogn)$

☐ $\theta(n)$

Problem 13.

Suppose you are given n numbers in a Python list in sorted order. Describe an efficient algorithm for checking to see if there is any number in the list which occurs 42 times. Do not use a dictionary/hash map.

Problem 14.

Consider this version of quicksort given below. It is essentially the same as that given in lecture, except that 1) it always uses the last element of the array as the pivot, and 2) it has a `print` statement inserted at a crucial place.

```
def quicksort(arr, start, stop):
    """Sort arr[start:stop] in-place."""
    if stop - start > 1:
        pivot_ix = partition(arr, start, stop, stop-1)
        quicksort(arr, start, pivot_ix)
        quicksort(arr, pivot_ix+1, stop)


def partition(arr, start, stop, pivot_ix):
    def swap(ix_1, ix_2):
        arr[ix_1], arr[ix_2] = arr[ix_2], arr[ix_1]

    pivot = arr[pivot_ix]
    swap(pivot_ix, stop-1)
    middle_barrier = start
    for end_barrier in range(start, stop - 1):
        if arr[end_barrier] < pivot:
            print('hello')
            swap(middle_barrier, end_barrier)
            middle_barrier += 1
        # else:
            # do nothing
    swap(middle_barrier, stop-1)
    return middle_barrier
```

Suppose **arr** is an array of length n with entries `[1, 2, 3, ..., n]`, where n is some large integer. If `quicksort(arr, 0, n)` is run, exactly how many times will `"hello"` be printed to the screen? Your answer should be an expression involving n, and should not involve $\sum$ or $\ldots$. Show your work.