Q1
1 Point

What is the best case time complexity of the following code?

```
def insertion_sort(arr):
    """Sort `arr` in ascending order."""
    n = len(arr)
    for i in range(1, n):
        x = arr[i]
        j = i - 1
        # find where to place x
        while j >= 0 and x < arr[j]:
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = x
```

- $\Theta(1)$
- $\Theta(\log n)$
- $\Theta(\sqrt{n})$
- $\Theta(n)$
- $\Theta(n \log n)$
- $\Theta(n\sqrt{n})$
- $\Theta(n^2)$
- $\Theta(n^3)$
- $\Theta(2^n)$

Explanation

The best case occurs when the array is already sorted. Notice that in this case, the `while` loop will never actually execute.

Q2
1 Point

What is the *worst case* time complexity of the following code?

```
def insertion_sort(arr):
    """Sort `arr` in ascending order."""
    n = len(arr)
    for i in range(1, n):
        x = arr[i]
        j = i - 1
        # find where to place x
        while j >= 0 and x < arr[j]:
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = x
```

- $\Theta(1)$
- $\Theta(\log n)$
- $\Theta(\sqrt{n})$
- $\Theta(n)$
- $\Theta(n \log n)$
- $\Theta(n\sqrt{n})$
- $\Theta(n^2)$
- $\Theta(n^3)$
- $\Theta(2^n)$

Explanation

The worst case occurs when the array is sorted in reverse order. In this case, the while-loop executes on every iteration of the for-loop, and must move `x` from its current position all the way to the beginning of the array.

Q3

1 Point

What is the *best case* time complexity of the following function?

```
def foo(arr):
    """arr is an array of size n"""
    for x in arr:
        for y in arr:
            if sum([x,y]) == 5:
```

```
                return sum(arr)
    return False
```

$\Theta(1)$

$\Theta(\log n)$

$\Theta(\sqrt{n})$

$\Theta(n)$

$\Theta(n \log n)$

$\Theta(n\sqrt{n})$

$\Theta(n^2)$

$\Theta(n^3)$

$\Theta(2^n)$

Explanation

In the best case, we `return sum(arr)` on the very first iteration.

Q4

1 Point

What is the *worst case* time complexity of the following function?

```
def foo(arr):
    """arr is an array of size n"""
    for x in arr:
        for y in arr:
            if sum([x,y]) == 5:
                return sum(arr)
    return False
```

$\Theta(1)$

$\Theta(\log n)$

$\Theta(\sqrt{n})$

$\Theta(n)$

$\Theta(n \log n)$

$\Theta(n\sqrt{n})$

$\Theta(n^2)$

$\Theta(n^3)$

$\Theta(2^n)$

Explanation

In the worst case, we never find `x + y == 5`, and go through all $\Theta(n^2)$ iterations just to return `False` at the end.

Q5

1 Point

What is the *expected* time complexity of the following function?

```
import random

def foo(n):
    """arr is an array of size n"""
    # randomly draw a number from 0, 1, 2, ..., n-1
    x = random.randint(0, n-1)
    if x == 1:
        for i in range(n):
            print(i)
    else:
        print("lucky!")

    for i in range(n):
        print(i)
```

- $\Theta(1)$
- $\Theta(\log n)$
- $\Theta(\sqrt{n})$
- $\Theta(n)$
- $\Theta(n \log n)$
- $\Theta(n\sqrt{n})$
- $\Theta(n^2)$
- $\Theta(n^3)$
- $\Theta(2^n)$

Q6
1 Point

Which of the below is a tight theoretical lower bound for the problem of finding
a mode of a collection of n numbers?

- $\Theta(1)$
- $\Theta(\log n)$
- $\Theta(n)$
- $\Theta(n \log n)$
- $\Theta(n^2)$
- $\Theta(n^3)$

Explanation

In the worst case, all numbers are distinct and we must loop through all of them to discover this, taking linear time.

Q7
1 Point

Consider the function $f(n) = 5n^2 - 100n + 100$ Which of the following asymptotic bounds are true? Choose **all** that apply.

- [ ] $\Theta(n)$
- [ ] $O(n)$
- [x] $\Omega(n)$
- [x] $\Theta(n^2)$
- [x] $O(n^2)$
- [x] $\Omega(n^2)$
- [ ] $\Theta(n^3)$
- [x] $O(n^3)$
- [ ] $\Omega(n^3)$

Q8
1 Point

Let
$f(n) = n^{\frac{1}{\log_2(n)}} \times n$
. Which of the following asymptotic bounds on f is true?

Hint: you might want to review your log properties. One useful one may be $\log_b c = 1/\log_c b$.

$\Theta(n)$

$\Theta(3^n)$

$\Theta(n^2)$

$\Theta(n^2 \cdot 3^n)$

$\Theta(2^n)$

Q9
1 Point

Let $f(n) = 12\log_2(3^{(n^2-2n)} + 2^{(\log n)} - 10n^2 - \log_3 n)$. Which of the following asymptotic bounds on f is true?

$\Theta(1)$

$\Theta(\log n)$

$\Theta(n)$

$\Theta(n \log n)$

$\Theta(n^2)$

$\Theta(2^n)$

$\Theta(3^n)$

Q10
1 Point

True or False. If $f_1 = \Theta(g_1(n))$ and $f_2 = O(g_2(n))$ then $\frac{f_1}{f_2} = \Theta(g_1/g_2)$.

True

False

Explanation

Try breaking it with a counterexample. What if $f_1 = n^3$, $f_2 = n^2$, $g_1 = n^3$ and $g_2 = n^3$. All of the conditions are satisfied, but $f_1/f_2 = n$, while $g_1/g_2 = 1$, so f_1/f_2 is not $\Theta(g_1/g_2)$.

Q11
1 Point

Suppose algorithm A takes $\Theta(n^2)$ time, while algorithm B takes $\Theta(n^3)$ time.

True or False: suppose both algorithms are run on the same input of size 100. Algorithm A must complete in less time than algorithm B.

True

False

Explanation

Algorithm A could have really large constants. For example, it could take 1,000,000 n^2 seconds to run, while algorithm B takes $.000n^3$. Algorithm B will still be better for $n = 100$.

Q12
1 Point

True or False. If $f = \Omega(n^5)$ and $f = O(n^7)$ then f must be either $\Theta(n^5)$, or $\Theta(n^6)$, or $\Theta(n^7)$.

True

False

Explanation

Counterexample: $f = n^{6.5}$. This is neither $\Theta(n^5)$ or $\Theta(n^6)$ or $\Theta(n^7)$; it is $\Theta(n^{6.5})$.