DSC 190 - Discussion 01

Problem 1.

In lecture, we analyzed dynamic arrays which had *geometric* growth rate. That is, when a resize occurred, the new size was a constant factor times the old. Now consider a *linear* growth rate, where the new size is the old size plus a constant. Show that the amortized time complexity of the append operation is $\Theta(n)$.

Problem 2.

In each of the problems below state the best case and worst case time complexities of the given piece of code using asymptotic notation. Note that some algorithms may have the same best case and worst case time complexities. If the best and worst case complexities *are* different, identify which inputs result in the best case and worst case. You do not need to show your work for this problem.

Example Algorithm: `linear_search` as given in lecture.

Example Solution: Best case: $\Theta(1)$, when the target is the first element of the array. Worst case: $\Theta(n)$, when the target is not in the array.

a)
```python
def f_1(data):
    """`data` is a two-dimensional array of size n*n"""
    n = len(data)
    for i in range(n):
        for j in range(n):
            if data[i, j] == 42:
                return (i,j)
```

b)
```python
def f_2(data):
    """`data` is an array of n numbers"""
    n = len(numbers)
    swapped = True
    while swapped:
        swapped = False
        for i in range(1,n):
            if numbers[i-1] < numbers[i]:
                # next line swaps elements in Theta(1) time
                numbers[i], numbers[i-1] = numbers[i-1], numbers[i]
                swapped = True
```

c)
```python
def median(numbers):
    """computes the median. `numbers` is an array of n numbers"""
    n = len(numbers)
    for x in numbers:
        less = 0
        more = 0
        for y in numbers:
            if y <= x:
                less += 1
            if y >= x:
                more += 1
        if less >= n/2 and more >= n/2:
```

```
            return x
```

d)
```
def mode(data):
    """computes the mode. `data` is an array of n numbers."""
    mode = None
    largest_frequency = 0
    for x in data:
        count = 0
        for y in data:
            if x == y:
                count += 1
        if count > largest_frequency:
            largest_frequency = count
            mode = x
        if count > n/2:
            return mode
    return mode
```

e)
```
def index_of_median(numbers):
    """`numbers` is an array of size n"""
    # the median() from part c
    m = median(numbers)
    # the linear_search() from lecture
    return linear_search(numbers, m)
```