DSC 40B - Homework 02

Due: Tuesday, January 19

Write your solutions to the following problems by either typing them up or handwriting them on another piece of paper. Unless otherwise noted by the problem's instructions, show your work or provide some justification for your answer. Homeworks are due via Gradescope on Tuesday at 11:59 p.m.

Problem 1. *(Eligible for Redemption)*

Write each of the following functions in Θ-notation in simplest terms. You do not need to show your work or provide justification.

Hint: your answer should have the form $\Theta(n^\alpha)$, $\Theta((\log_\mu(n))^\beta)$, $\Theta(\gamma^{\delta n})$, or $\Theta(1)$, where $\alpha, \beta, \gamma, \delta, \mu$ are constants that depend on the problem. While you *can* find the answer using more formal means (such as limits), you should be able to find the answers rather quickly using informal reasoning. You can always use the formal approach to check your answer, if you wish.

a) $f(n) = \sqrt{n^3 + 2n^2 + 500}$

b) $f(n) = 20 \times 2^{\log_2(n^3+n)}$

c) $f(n) = 2^{10} + 10^2 + \log_{10} 1000$

d) $f(n) = \log_2(3^{n+2} + 5n^3 + 1)$

e) $f(n) = 2^{10n} + 10^{2n}$

f) $f(n) = \log_2(n^2 + 3n + 5) \times \log_2(3n^3)$

Problem 2. *(Eligible for Redemption)*

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 otherwise 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)
```
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)
```

Problem 3. *(Eligible for Redemption)*

For each problem below, state a *tight* theoretical lower bound. Provide justification for this lower bound, and sketch an *optimal* algorithm; that is, an algorithm which has the lower bound as its worst case time complexity.

Example: Given an array of size n and a target t, determine the index of t in the array.

Example Solution: $\Omega(n)$, because in the worst case any algorithm must look through all n numbers to verify that the target is not one of them, taking $\Omega(n)$ time. Algorithm: loop through the array, checking each element against the target.

a) Given an array of n numbers, find the maximum.

b) Given a *sorted* array of n numbers, find a median.

c) Given an array of the heights of n people, determine the height of tallest person you can make by stacking two of them.

Problem 4. *(Eligible for Redemption)*

In discussion, we learned how to express the asymptotic time complexity of algorithms which depend on *multiple* sizes. This problem will give more practice doing this.

State the asymptotic time complexity of each of the operations below using multivariate Θ notation. Your answer should include every size variable used in the problem statement, such as number of points, dimensionality, etc. You do not need to show your work.

Example: compute the distance between a point in $\mathbb{R}^d$ and n other points in $\mathbb{R}^d$.

Example Solution: $\Theta(nd)$.

a) Given a set of n points in $\mathbb{R}^d$, compute the distance between each pair of points.

b) Given a set of b blue points and a set of r red points in $\mathbb{R}^d$, find the smallest distance between a blue point and a red point using the brute force approach.

c) Given a set of n points in $\mathbb{R}^d$, 10% of them red and 90% of them blue, compute the smallest distances between a blue point and a red point.

d) Given a set $\mathcal{D}$ of n data points in $\mathbb{R}^d$ and a new point $\vec{x}$ in $\mathbb{R}^d$, find the point in $\mathcal{D}$ which is kth closest to $\vec{x}$ using the `kth_smallest` function below:

```python
def kth_smallest(numbers, k):
    """Given an array of numbers, find index of kth smallest."""
    numbers = np.copy(numbers)
    for _ in range(k):
        smallest_index = None
        smallest_value = float('inf')
        for i, x in enumerate(numbers):
            if x < smallest_value:
                smallest_index = i
                smallest_value = x
        numbers[smallest_index] = float('inf')
    return smallest_index, smallest_value
```

Problem 5.

In each of the problems below compute the average case time complexity of the given code. State your answer using asymptotic notation. Show your work for this problem by stating what the different cases are, the probability of each case, and how long each case takes. Also show the calculation of the expected time.

a)
```python
def median(numbers):
    """computes the median. `numbers` is an array of n numbers"""
    np.random.shuffle(numbers) # randomly permute `numbers` in linear time
    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
```

Tip: `np.random.shuffle` randomly permutes an array in linear time. In this part, you may assume that the array has a unique median.

b)
```
def f(numbers):
    """search by randomly guessing. `numbers` is an array of n numbers"""
    n = len(numbers)
    # randomly choose a number between 0 and n-1 in constant time
    guess = np.random.randint(n)

    if numbers[guess] == max(numbers):
        for i in range(n**2):
            print(i * numbers[guess])
```

In this part, you may assume that the numbers are distinct. Be careful: does every line take constant time to execute once?