DSC 40B - Homework 02

Due: Wednesday, January 21

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 at 11:59 p.m.

Problem 1.

For each of the following functions, express its rate of growth using Θ-notation, and prove your answer by finding constants which satisfy the definition of Θ-notation. Make sure to show your work by writing out the chain of inequalities to prove each bound, like we did in lecture. Note: please do *not* use limits to prove your answer (though you can use them to check your work).

a) $f(n) = 3n^2 - 7n + 10$

Solution: $\Theta(n^2)$

Let $c_1 = 2, c_2 = 3, n_0 = 5$.

Lower bound:

We start with:

$$5 \leq n$$

Adding $3n^2 - 8n + 10$ to both sides, we get

$$3n^2 - 8n + 15 \leq f(n).$$

For $n \geq 5$, we see that the LHS will always be at least $2n^2$, thus $2n^2 \leq f(n)$ and the lower bound is satisfied.

Upper bound:

We start with:

$$5 \leq n$$

Adding $3n^2 - 7n + 5$ to both sides, we have

$$f(n) \leq 3n^2 - 6n + 5$$

For $n \geq 5$, we see that the RHS will always be at most $3n^2$, thus $f(n) \leq 3n^2$ thus the upper bound is satisfied.

If you are unsure, you can check with a graphing calculator.

b) $f(n) = \dfrac{n^2 + 2n - 5}{n + 10}$

Solution: $\Theta(n)$

Let $c_1 = 0.5, c_2 = 1, n_0 = 10$.

Lower bound:

We start with:

$$n \geq 10.$$

Multiply by $0.5n$ on both sides and get:

$$0.5n^2 \geq 5n.$$

Subtract $3n + 5$ from both sides and get:

$$0.5n^2 - 3n - 5 \geq 2n - 5$$

Note that the RHS is at least 0 for $n \geq 10$, so we have:

$$0.5n^2 - 3n - 5 \geq 0.$$

Add $0.5n^2 + 5n$ to both sides to obtain:

$$n^2 + 2n - 5 \geq 0.5n^2 + 5n.$$

Divide by $n + 10$ on both sides to obtain:

$$f(n) \geq 0.5n,$$

thus the lower bound is satisfied.

Upper bound:

We start with:

$$10 \leq n$$

Adding n to both sides, we get:

$$n + 10 \leq 2n.$$

Taking the inverse, we get:

$$\frac{1}{2n} \geq \frac{1}{n+10}.$$

Multiplying by $n^2 + 2n - 5$ on both sides, we get:

$$\frac{n^2 + 2n - 5}{2n} \geq f(n)$$

. Doing polynomial long division on the LHS, we arrive at:

$$\frac{1}{2}n + 1 - \frac{5}{2n} \geq f(n)$$

For $n \geq 10$, we see that the LHS will always be at most n, thus $f(n) \leq n$ and the upper bound is satisfied.

If you are unsure, you can check with a graphing calculator.

c) $f(n) = \begin{cases} 3n^2, & n \text{ is odd,} \\ 2n^2, & n \text{ is even} \end{cases}$

Solution: $\Theta(n^2)$

Let $c_1 = 2, c_2 = 3, n_0 = 1$. For all $n \geq 1$, regardless of whether n is even or odd, we will have $2n^2 \leq f(n) \leq 3n^2$ so both upper and lower bounds are satisfied.

d) State the growth of the function below using Θ notation in as simplest of terms possible, and prove your answer by finding constants which satisfy the definition of Θ notation.

E.g., if $f(n)$ were $3n^2 + 5$, we would write $f(n) = \Theta(n^2)$ and not $\Theta(3n^2)$.

$$f(n) = \frac{n^3 - n^2 + n + 1000}{(n-1)(n+3)}$$

Solution: $\Theta(n)$

Let $c_1 = 0.5, c_2 = 1, n_0 = 20$.

Lower bound:

We start with:

$$n \geq 20.$$

Adding $0.5n^3 - 2n^2 + 1001.5$ to both sides, we get:

$$0.5n^3 - 2n^2 + n + 1001.5 \geq 0.5n^3 - 2n^2 + 1021.5.$$

Note that the RHS is at least 0 for $n \geq 20$. Thus:

$$0.5n^3 - 2n^2 + n + 1001.5 \geq 0.$$

Adding $0.5n^3 + n^2 - 1.5$ to both sides, we get:

$$n^3 - n^2 + n + 1000 \geq 0.5n^3 + n^2 - 1.5$$

Dividing by $n^2 + 2n - 3$ on both sides, we get:

$$f(g) \geq 0.5n,$$

thus the lower bound is satisfied.

Upper bound: We start with:

$$n \geq 20.$$

Adding $3n^2 - 5n - 1000$ to both sides, we get:

$$3n^2 - 4n - 1000 \geq 3n^2 - 5n - 980.$$

Note that the RHS is at least 0 for $n \geq 20$, thus:

$$3n^2 - 4n - 1000 \geq 0.$$

Adding $n^3 - n_n^2 + 1000$ to both sides, we get:

$$n^3 - n^2 + n + 1000 \leq n^3 + 2n^2 - 3n$$

. Dividing by $n^2 + 2n - 3$ on both sides, we get:

$$f(n) \leq n,$$

thus the upper bound is satisfied.

Problem 2.

Suppose $T_1(n), \ldots, T_6(n)$ are functions describing the runtime of six algorithms. Furthermore, suppose we

have the following bounds on each function:

$$T_1(n) = \Theta(n^{2.5})$$
$$T_2(n) = O(n \log n)$$
$$T_3(n) = \Omega(\log n)$$
$$T_4(n) = O(n^4) \text{ and } T_4 = \Omega(n^2)$$
$$T_5(n) = \Theta(n)$$
$$T_6(n) = \Theta(n \log n)$$
$$T_7(n) = O(n^{1.5} \log n) \text{ and } T_7 = \Omega(n \log n)$$

What are the best bounds that can be placed on the following functions?

For this problem, you do not need to show work.

Hint: watch the supplemental lecture at `https://youtu.be/tmR-bIN2qw4`.

Example 1: $T_1(n) + T_2(n)$.

Solution: $T_1(n) + T_2(n)$ is $\Theta(n^{2.5})$.

Example 2: $f(n) = 2 \cdot T_4(n)$.

Solution: $f(n) = 2 \cdot T_4(n)$ is $O(n^4)$ and $\Omega(n^2)$.

a) $T_1(n) + T_5(n)$

> **Solution:** $\Theta(n^{2.5})$
>
> $\Theta(n^{2.5})$ dominates $\Theta(n)$.

b) $T_1(n) + T_7(n)$

> **Solution:** $\Theta(n^{2.5})$
>
> $\Theta(n^{2.5})$ dominates both $O(n^{1.5} \log n)$ and $\Omega(n \log n)$.

c) $T_2(n) + T_6(n)$

> **Solution:** $\Theta(n \log n)$
>
> $O(n\ logn)$ only places an upper bound whereas $\Theta(n \log n)$ places an upper and lower bonud, so it is more precise.

d) $T_2(n) + T_4(n)$

> **Solution:** $O(n^4)$ and $\Omega(n^2)$
>
> The best case for T_4 dominates the worst case for T_2, so the bounds for T_4 dominates.

e) $T_1(n) + T_4(n)$

> **Solution:** $O(n^4)$ and $\Omega(n^{2.5})$
>
> The upper bound for T_5 dominates the upper bound for T_1, and the lower bound for T_1 dominates

the lower bound for T_4.

f) $T_7(n) + T_4(n)$

Solution: $O(n^4)$ and $\Omega(n^2)$ The upper and lower bounds for T_4 dominate the upper and lower bounds for T_7, respectively.

g) $T_3(n) + T_1(n)$

Solution: $\Theta(n^{2.5})$

$\Theta(n1.5)$ provides a stricter lower bound than $\Omega(\log n)$.

h) $T_2(n) + \frac{T_1(n)}{n^2}$

Solution: $O(n \log n)$ and $\Omega(\sqrt{n})$.

$n \log n$ dominates over $\Theta(n^{2.5})$ divided by n^2. We still have a lower bound of $\Omega(\sqrt{n})$ provided by dividing T_1 by n^2.

i) $T_1(n) \times T_4(n)$

Solution: $O(n^{6.5})$ and $\Omega(n^{4.5})$

j) $T_6(n) + \frac{T_4(n)}{T_5(n)}$

Solution: $O(n^3)$ and $\Omega(n \log n)$

$\frac{T_4}{T_5}$ yields a function with time complexity $O(n^3)$ and $\Omega(n)$. Thus, the contribution from T_6 dominates for the lower bound but not the upper bound.

Problem 3.

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)
```
def kth_largest(numbers, k):
    """Finds the k-th largest element in the array.
    `numbers` is an array of n numbers."""
    n = len(numbers)
    for i in range(n):
        count = 0 # Count how many numbers are larger than numbers[i]
        for j in range(n):
            if numbers[j] > numbers[i]:
                count += 1
```

```
        if count == k - 1:
            return numbers[i]
```

Solution: Best case: $\Theta(n)$, when the first element of the array is kth largest.

Worst case: $\Theta(n^2)$, when the last element of the array is kth largest.

b)
```
def index_of_kth_largest(numbers, k):
    """`numbers` is an array of size n"""
    # the kth_largest() from above
    element = kth_largest(numbers, k)
    # the linear_search() from lecture 3 slide 22
    return linear_search(numbers, element)
```

Solution: Best case: $\Theta(n)$.

Worst case: $\Theta(n^2)$.

As `linear_search` takes constant time, and this function calls the function defined in part a, the best case and worst case are the same as the best case and worst case for part a.

Problem 4.

In each of the problems below compute both the expected time and worst time 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)
```
def double_coin_toss(n):
    coin_1 = np.random.rand() > 0.5
    coin_2 = np.random.rand() > 0.5
    if coin_1 and coin_2:
        i = n
        while i > 0:
            print("We got two heads!")
            i -= 2
    else:
        for i in range(n ** 2):
            print("We didn't get two heads.")
```

Solution: Expected case: $\Theta(n^2)$

Worst case: $\Theta(n^2)$

There is a 0.25 probability of getting two heads, which executes a loop with $\Theta(n)$ time complexity, and a 0.75 probability of not getting two heads, which executes a loop with $\Theta(n^2)$ time complexity. Thus, the expected time is $0.25\Theta(n) + 0.75\Theta(n^2)$, resulting in $\Theta(n^2)$.

b)
```
def foo(n):
    # randomly choose a number between 0 and n-1 in constant time
    k = np.random.randint(n)

    if k < np.sqrt(n):
        for i in range(n**2):
            print(i)
    else:
```

```
    j = n
    while j >= 1:
        print(j)
        j = j / 2
```

Solution: Expected case: $\Theta(n^{1.5})$

Worst case: $\Theta(n^2)$

There is a $\frac{1}{\sqrt{n}}$ probability to execute a loop with time complexity n^2 and a $(1 - \frac{1}{\sqrt{n}})$ probability to execute a loop with time complexity $\Theta(\log n)$, therefore the expected probability is $\sqrt{n}\Theta(n^2) + (1 - \frac{1}{\sqrt{n}})\Theta(\log n)$ resulting in $\Theta(n^{1.5})$.

c)
```
def bar (n):
    # randomly choose a number between 0 and n-1 in constant time
    k = np.random.randint(n)
    if k < 100:
        for i in range(n**2):
            print("Small number!")
    else:
        for j in range(n):
            s = n
            while s >= 1:
                print("Large number!")
                s = s / 2
```

Solution: Expected case: $\Theta(n \log n)$

Worst case: $\Theta(n^2)$

There is a $\frac{100}{n}$ probability to execute a loop with time complexity $\Theta(n^2)$ and a $1 - \frac{100}{n}$ probability to execute a loop with time complexity $\Theta(n \log n)$, therefore the expected time complexity is $\frac{100}{n}\Theta(n^2) + (1 - \frac{100}{n})\Theta(n \log n)$. The first term simplifies to $\Theta(n)$ and the second term simplifies to $\Theta(n \log n)$ thus the expected time complexity is $\Theta(n \log n)$.

d)
```
def baz(n):
    # randomly choose a number between 0 and n-1 in constant time
    x = np.random.randint(n)

    for i in range(x):
        for j in range(i):
            print("Hello!")
```

Hint: In class, we saw that the sum of the first n integers is $\frac{n(n+1)}{2}$. It turns out that the sum of the first n integers squared (so, $1^2 + 2^2 + 3^2 + \ldots + n^2$) is $\frac{n(n+1)(2n+1)}{6}$. You can (and should) use this fact, but make sure to point it out when you do.

Solution: $\Theta(n^2)$

Each integer k has a $\frac{1}{n}$ probability of being chosen, for which $\frac{k(k-1)}{2}$ operations will be executed.

Thus, the expected number of operations is:

$$\begin{aligned}\frac{1}{n}\sum_{k=0}^{n-1}\frac{k(k-1)}{2} &= \frac{1}{2n}\sum_{k=0}^{n-1}k^2-k \\ &= \frac{1}{2n}\left(\sum_{k=0}^{n-1}k^2-\sum_{k=0}^{n-1}k\right) \\ &= \frac{1}{2n}\left(\frac{n(n-1)(2n-1)}{6}-\frac{n(n-1)}{2}\right) \\ &= \frac{(n-1)(2n-1)}{12}-\frac{n-1}{4}\end{aligned}$$

The first term has time complexity $\Theta(n^2)$ and the second $\Theta(n)$, so overall the time complexity is $\Theta(n^2)$.

Problem 5.

For each problem below, state the largest theoretical lower bound that you can think of and justify (that is, your bound should be "tight"). Provide justification for this lower bound. You do not need to find an algorithm that satisfies this lower bound.

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.

a) Given a list of size n containing `True`s and `False`s, determine whether `True` or `False` is more common (or if there is a tie).

Solution: $\Omega(n)$

In the best case, the first $\lfloor \frac{n}{2} \rfloor + 1$ elements of the list are the same, thus we would only need to read half the entries. This is still $\Omega(n)$ since this procedure scales linearly with the size of the list.

b) Given a list of n numbers, all assumed to be integers between 1 and 100, sort them.

Solution: $\Omega(n)$

At a minimum, we need to read the list which takes $\Theta(n)$ time. Then you may use your favorite linear sorting algorithm which is also $\Theta(n)$, thus the total time complexity is $\Theta(n)$ in the best case.

c) Given an $\sqrt{n} \times n$ array whose rows are sorted (but whose columns may not be), find the largest overall entry in the array.

For example, the array could look like:

$$\begin{pmatrix} -2 & 4 & 7 & 8 & 10 & 12 & 20 & 21 & 50 \\ -30 & -20 & -10 & 0 & 1 & 2 & 3 & 21 & 23 \\ -10 & -2 & 0 & 2 & 4 & 6 & 30 & 31 & 35 \end{pmatrix}$$

This is an $\sqrt{n} \times n$ array, with $n = 9$ (there are 3 rows and 9 columns). Each row is sorted, but the columns aren't.

Solution: $\Omega(\sqrt{n})$

Since the rows are sorted, we need to read the last element of each row. There are $\sqrt{n}$ rows, so this has time complexity $\Theta(\sqrt{n})$ in the best case.