DSC 40B - Discussion 01

Problem 1.

Suppose $A(n), B(n), C(n)$ are functions describing the runtime of three algorithms. Furthermore, suppose we have the following bounds on each function:

$$A(n) = \Theta(n^3)$$
$$B(n) = O(n^2 \log n)$$
$$C(n) = O(n^5) \text{ and } C(n) = \Omega(n^2)$$

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

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

Example 1: $A(n) + B(n)$.

Solution: $A(n) + B(n)$ is $\Theta(n^3)$.

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

Solution: $f(n) = 2 \cdot C(n)$ is $O(n^5)$ and $\Omega(n^2)$.

a) $A(n) + B(n)$

> **Solution:** $\Theta(n^3)$

b) $B(n) + C(n)$

> **Solution:** $O(n^5)$ and $\Omega(n^2)$

c) $A(n) + C(n)$

> **Solution:** $O(n^5)$ and $\Omega(n^3)$

Problem 2.

Compute the best and worst case time complexity for the following code snippets:

a)
```python
def f_1(arr1, arr2):
    """`arr1` and `arr2` are two arrays each of size n"""
    n = len(arr1)
    for i in range(n):
        for j in range(n):
            if arr1[i] + arr2[j] == 0:
                return (i, j)
```

> **Solution:**
> Best case time complexity is $\Theta(1)$ when `arr1[0] + arr2[0] == 0`
> Worst case time complexity if $\Theta(n^2)$ when no two elements in the array sum to 0.

b)
```python
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
```

> **Solution:**
> Best case time complexity is $\Theta(n)$ when all elements in the array are already sorted in increasing order.
> Worst case time complexity if $\Theta(n^2)$ when all elements in the array are sorted in decreasing order.

Problem 3.

In each of the parts below, compute the *expected running time* of the given code. Also state the *worst-case* running time. Use asymptotic notation.

For full credit, for each part: (1) clearly state the cases you consider, (2) give the probability of each case, (3) give the time taken in each case, and (4) compute the expected time as a probability-weighted sum.

Reminder. In Python, `np.random.randint(n)` picks a random integer uniformly from $\{0, 1, \ldots, n-1\}$.

a)
```python
def alpha(n):
    # pick a random integer from 0 to n-1
    r = np.random.randint(n)
    if r < n/5:
        for i in range(n**3):
            print(i)
    else:
        for i in range(n):
            print(i)
```

b)
```python
def beta(n):
    # pick a random integer from 0 to n-1
    r = np.random.randint(n)
    if r < 20:
        for i in range(n**2):
            print(i)
    else:
        for i in range(n):
            print(i)
```

c)
```python
def gamma(n):
    # pick a random integer from 0 to n-1
    m = np.random.randint(n)
    total = 0
    for i in range(m):
        for j in range(i):
            total += 1
```

d)
```python
def random_search(arr, target):
```

```
n = len(arr)
while True:
    idx = np.random.randint(n)
    if arr[idx] == target:
        return idx
```

Assume all elements in `arr` are distinct and `target` is in the array.

Solution:

- **(a)** There are two cases. If $r < n/5$ (probability $1/5$), the loop runs n^3 times, so the time is $\Theta(n^3)$. Otherwise (probability $4/5$), the loop runs n times, so the time is $\Theta(n)$.

  The expected time is

  $$\mathbb{E}[T] = \frac{1}{5}\Theta(n^3) + \frac{4}{5}\Theta(n) = \Theta(n^3).$$

  The worst-case time complexity is $\Theta(n^3)$.

- **(b)** If $r < 20$ (probability $20/n$), the running time is $\Theta(n^2)$. Otherwise (probability $1 - 20/n$), the running time is $\Theta(n)$.

  The expected time is

  $$\mathbb{E}[T] = \frac{20}{n}\Theta(n^2) + \left(1 - \frac{20}{n}\right)\Theta(n) = \Theta(n).$$

  The worst-case time complexity is $\Theta(n^2)$.

- **(c)** For a fixed value $m = k$, the inner loop executes $1 + 2 + \cdots + (k-1) = \Theta(k^2)$ times. Since m is chosen uniformly from $\{0, \ldots, n-1\}$, each case has probability $1/n$.

  The expected time is

  $$\mathbb{E}[T] = \sum_{k=0}^{n-1} \frac{1}{n}\Theta(k^2) = \frac{1}{n}\Theta(n^3) = \Theta(n^2).$$

  The worst-case time complexity is $\Theta(n^2)$.

- **(d)** On each iteration, the probability of success is $1/n$ and the probability of failure is $1 - 1/n$. Let Case α be the event that the algorithm succeeds on the α-th iteration. Then

  $$\Pr[\text{Case } \alpha] = (1 - 1/n)^{\alpha - 1} \cdot \frac{1}{n},$$

  and the time taken in Case α is $\Theta(\alpha)$.

  The expected running time is therefore $\Theta(n)$. The worst-case running time is unbounded.

Key takeaway: Expected running time is computed by identifying all possible cases, determining their probabilities, and summing the running time of each case weighted by its probability.