Problem Set 1 Solutions

1. **(15 points)** Order of Growth

   For each group of functions, sort the functions in increasing order of asymptotic (big-O) complexity. Partition each group into equivalence classes such that $f_i(n)$ and $f_j(n)$ are in the same class if and only if $f_i(n) = \Theta(f_j(n))$. (You do not need to show your work for this problem.)

   (a) **(5 points)** Group A:

   $$f_1(n) = n \log n$$
   $$f_2(n) = n + 100$$
   $$f_3(n) = 10n$$
   $$f_4(n) = 1.01^n$$
   $$f_5(n) = \sqrt{n} \cdot (\log n)^3$$

   Solution:

   `[[5],[2,3],[1],[4]]`

   We observe that f_5 has the smallest order of growth, since it grows sublinearly. (Recall that $\log n = o(n^\epsilon)$ for any $\epsilon > 0$.) Next, $f_2 = \Theta(f_3)$, since additive and multiplicative constants do not affect asymptotic growth. We know that $n \log n = \omega(n)$, and finally the largest growth is 1.01^n, which grows exponentially.

   (b) **(5 points)** Group B:

   $$f_1(n) = 2^n$$
   $$f_2(n) = 2^{2n}$$
   $$f_3(n) = 2^{n+1}$$
   $$f_4(n) = 10^n$$

   Solution:

   `[[1,3],[2],[4]]`

   We observe that 2^n and 2^{n+1} are in the same equivalence class, since they differ only by a factor of two (and constant multiples do not matter.) Next, we observe that $f_2 = \omega(f_1)$, since

   $$\lim_{n\to\infty} \frac{2^{2n}}{2^n} = \lim_{n\to\infty} 2^n = \infty.$$

Finally, $10^n = \omega(2^{2n})$, since

$$\lim_{n\to\infty} \frac{10^n}{2^{2n}} = \lim_{n\to\infty} \frac{2^n \cdot 5^n}{2^n \cdot 2^n} = \lim_{n\to\infty} \left(\frac{5}{2}\right)^n = \infty.$$

(c) **(5 points)** Group C:

$$f_1(n) = n^n$$
$$f_2(n) = n!$$
$$f_3(n) = 2^n$$
$$f_4(n) = 10^{10^{100}}$$

Solution:

`[[4],[3],[2],[1]]`

The smallest order of growth is clearly f_4, since constant functions are $\Theta(1)$. Next, we observe that $2^n = o(n!)$, since

$$\lim_{n\to\infty} \frac{n!}{2^n} = \lim_{n\to\infty} \frac{1}{2} \cdot \frac{2}{2} \cdot \frac{3}{2} \cdots \frac{n}{2} \geq \lim_{n\to\infty} \frac{1}{2} \cdot \left(\frac{3}{2}\right)^{n-2} = \infty.$$

Finally, the fact that $n! = o(n^n)$ follows from Stirling's approximation that $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$, and noting that

$$\frac{n^n}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n} = \frac{1}{\sqrt{2\pi n}e^{-n}} = \frac{e^n}{\sqrt{2\pi n}}$$

which goes to ∞ as $n \to \infty$. We can also prove that $n! = o(n^n)$ directly by expanding $n!$ and n^n to notice that $n^n/n! \geq \frac{n}{2}$, which goes to ∞ as $n \to \infty$.

2. **(10 points)** Recurrence Relations

(a) **(5 points)** What is the asymptotic complexity of an algorithm with runtime given by the recurrence:

$$T(n) = 4T(n/2) + \log n.$$

1. $\Theta(n)$
2. $\Theta(n \log n)$
3. $\Theta(n^2)$
4. $\Theta(n^2 \log n)$

Solution:

`3`

The easiest way to see this is by the master theorem, since $n^{\log_2 4} = n^2$, and $\log n = o(n^{2-\epsilon})$. We can also solve this problem by expanding the recurrence to obtain a sum, and noting that the largest term in the sum dictates the asymptotic behavior. (Each subsequent term is less than half of the previous term, and therefore the entire sum is bounded by a constant multiple of the first term.)

(b) **(5 points)** What is the asymptotic complexity of an algorithm with runtime given by the recurrence:

$$T(n) = 9T(n/3) + n^2.$$

1. $\Theta(n \log n)$
2. $\Theta(n^2)$
3. $\Theta(n^2 \log n)$
4. $\Theta(n^3)$

Solution:

3

This follows by the master theorem, since $n^{\log_3 9} = n^2$. Since the additive term has the same asymptotic growth as $n^{\log_3 9}$, we gain an extra log factor.

3. **(20 points)** 2D Peak Finding

Consider the following approach for finding a peak in an $(n \times n)$ matrix:

1. Find a maximum element m in the middle column of the matrix.
   - If the the left neighbor of m is greater than it, discard the center column and the right half of the matrix.
   - Else, if the right neighbor of m is greater than it, discard the center column and the left half of the matrix.
   - Otherwise, stop and return m.
2. Find a maximum element m' in the middle row of the remaining matrix.
   - If the the upper neighbor of m' is greater than it, discard the center row and the bottom half of the matrix.
   - Else, if the lower neighbor of m' is greater than it, discard the center row and the top half of the matrix.
   - Otherwise, stop and return m'.
3. Go back to step 1.

(a) **(5 points)** Let the worst-case running time of this algorithm on an $(n \times n)$ matrix be $T(n)$. State a recurrence for $T(n)$. (You may assume that it takes constant time to discard parts of the matrix.)

Solution:

```
T(n) = T(n/2) + \Theta(n)
```

In each iteration, we reduce an $n \times n$ matrix to a $n/2 \times n/2$ submatrix. Doing so requires $\Theta(n)$ work, since we must check all elements in the appropriate row/column.

(b) **(5 points)** Solve this recurrence to find the asymptotic behavior of $T(n)$.

```
T(n) = \Theta(n)
```

This follows by expanding the recurrence. We can bound the $\Theta(n)$ terms above or below by cn, and then bound $T(n)$ by

$$cn + \frac{cn}{4} + \frac{cn}{8} + \frac{cn}{16} + \cdots \leq 2cn.$$

(c) **(10 points)** Prove that this algorithm always finds a peak, or give a small ($n \leq 7$) counterexample on which it does not.

Solution: This algorithm does not always find a peak. One possible counterexample matrix is given below.

```
[[0,0,0,3,2,1,0],
 [0,0,0,0,0,0,0],
 [0,0,0,0,7,0,0],
 [0,0,0,4,6,0,0],
 [0,0,0,0,0,0,0],
 [0,0,0,0,0,0,0],
 [0,0,0,0,0,0,0]]
```

When we run the algorithm, it will return 2 in location $(0, 4)$, which is not a peak. The algorithm will first find that 4 is the maximum element in the middle column, and therefore recurse on the right half of the matrix (since 4 is adjacent to 6). It will then find that 6 is the maximum element in right half of the middle row, and will therefore recurse on the upper right quadrant (since 7 is adjacent to 6). It will then find that 1 is the maximum element in the middle column of the 3×3 upper-right submatrix, and will therefore discard all but the top three elements in the column beginning with "2." Finally, it will return 2 in location $(0, 4)$ as the answer, since 2 is above the 0 in location $(1, 4)$ and is greater than it.

4. **(30 points)** Programming Exercise: Peak In Circle

Write a function `find_peak_in_circle` that efficiently finds a peak value in a circle of integers. This function should take a list of integers as an input. Two elements in this list are *adjacent* if they are consecutive elements of the list or if they are the first and

last element. A peak is an element of the list which is greater than both of its adjacent elements - your goal is to find the value of any peak.

You may assume that the input list is non-empty. However, you may not change the entries of the list, and your function should also accept (immutable) tuples. Here are some example test cases that your function should agree with:

```
# Both 4 and 5 are peaks in the array [1, 2, 5, 3, 4]
find_peak_in_circle([1, 2, 5, 3, 4]) in (4, 5)
# The element 3 is not a peak in [3, 2, 1, 4] because it is adjacent to 4
find_peak_in_circle([3, 2, 1, 4]) == 4
```

Solution: An example algorithm is below. Notice that in this algorithm we keep track of the bounds into the array (instead of copying the array with each recursion) and we recurse on the side containing the maximum element we have seen thus far. This algorithm has worst-case running time $\Theta(\log n)$.

```
def find_peak_in_circle(input):
  max_location = 0
  if input[-1] > input[0]:
    max_location = len(input) - 1

  min_bound = 0
  max_bound = len(input)
  while min_bound < max_bound - 1:
    mid = (min_bound + max_bound) / 2
    if input[mid] < input[max_location]:
      if mid < max_location:
        min_bound = mid + 1
      else:
        max_bound = mid
    elif mid + 1 < max_bound and input[mid + 1] > input[mid]:
      max_location = mid + 1
      min_bound = mid + 1
    elif input[mid - 1] > input[mid]:
      max_location = mid - 1
      max_bound = mid
    else:
      return input[mid]

  return input[max_location]
```