Problem Set One

Zoe Farmer

February 26, 2024

1. *For each claim, determine whether the statement is* ***True*** *or* ***False****. Justify your answer.*

   (a) $n + 3 = O(n^3) \rightarrow$ **True**. According to the definition of Big-O notation, $f = O(g)$ if

   $$(\exists c, k > 0, x > k)\left[|f(x)| \leq c\,|g(x)|\right]$$

   Therefore

   $$\boxed{|n + 3| \leq c\left|n^3\right|}$$

   and the statement is valid.

   (b) $3^{2n} = O(3^n) \rightarrow$ **False**. Again, using the previous definition of Big-O notation we see that

   $$\boxed{\left|3^{2n}\right| \not\leq c\,|3^n|}$$

   (c) $n^n = o(n!) \rightarrow$ **False**. We can use the definition of little-o notation which states that f is little-o of g if

   $$\lim_{x \to \infty} \frac{f(x)}{g(x)} = 0$$

   Therefore this statement turns to

   $$\boxed{\lim_{x \to \infty} \frac{n^n}{n!} = \infty}$$

   Therefore the statement is invalid.

   (d) $\frac{1}{3n} = o(1) \rightarrow$ **True**. We then use the above definition again and apply L'Hospital's Rule to determine the value of the limit.

   $$\lim_{x \to \infty} \frac{1}{3n} \rightarrow \boxed{\lim_{x \to \infty} \frac{0}{3} = 0}$$

   (e) $\ln^3(n) = \Theta(\log_2^3(n)) \rightarrow$ **False**[1] We can use the definition of Big-O notation to determine that

   $$\boxed{\ln^3(n) = O(\log_2^3(n))}$$

[1] This is assuming that $lg(x)$ refers to the base-2 logarithm, $\log_2(x)$.

because $\left|\ln^3(n)\right| \leq c\left|\log_2^3(n)\right|$, however

$$\boxed{\log_2^3(n) \neq O(\ln^3(n))}$$

because $\left|\log_2^3(n)\right| \not\leq c\left|\ln^3(n)\right|$. Therefore the statement is false.

---

2. *Simplify each of the following expressions.*

   (a)
   $$\frac{d}{dt}(3t^4 + 1/3t^3 - 7) \rightarrow \boxed{12t^3 + t^2}$$

   (b)
   $$\sum_{i=0}^{k} 2^i \rightarrow 1 + 2 + 4 + \cdots + 2^k \rightarrow \boxed{2^{k+1} - 1}$$

   (c)
   $$\Theta\left(\sum_{k=1}^{n} \frac{1}{k}\right) \rightarrow \boxed{H_n}$$

   Where H_n is the n^{th} Harmonic number.

---

3. *T is a balanced binary search tree storing n values. Describe an $O(n)$-time algorithm that takes input T and returns an array containing the same values in ascending order.*

   (a) Below is the code to perform this operation.

Balanced Binary Search Tree to Ascending Array

```
asc = []                               # List to populate
class Node:                            # The structure of any given node
    left = None                        # Class object of left node
    right = None                       # Class object of right node
    value = None                       # Value of node
def tree_to_array(head):               # Function to scrape in asc order
    if head.left != None:              # If left is node
        tree_to_array(head.left)       # Take left
        head.left = None               # Destroy traversed result
    if head.right != None:             # Else take right
        asc.append(head.value)         # Take next smallest val
        tree_to_array(head.right)      # Go right
        head.right = None              # Destroy traversed result
    if head.left is None and           # If both sides are empty
            head.right is None:
        try:
            if (head.value >=
                asc[len(asc) - 1]):    # If larger than prev
                asc.append(head.value) # This value is our next smallest
        except IndexError:             # Only enter if list is empty
            asc.append(head.value)     # This value is our next smallest
head = construct_tree(random=true)     # Create a random balanced tree
print(tree_to_array(head))             # Print our end array
```

---

4. *Acme Corp. has asked Professor Flitwick to develop a faster algorithm for their core business. The current algorithm runs in $f(n)$ time. (For concreteness, assume it takes $f(n)$ microseconds to solve a problem of size exactly n.) Flitwick believes he can develop a faster algorithm, which takes only $g(n)$ time, but developing it will take t days. Acme only needs to solve a problem of size n once. Should Acme pay Flitwick to develop the faster algorithm or should they stick with their current algorithm? Explain.*

   (a) Let $n = 41, f(n) = 1.99^n, g(n) = n^3$ and $t = 17$ days.

      i. The time it will take the original algorithm to complete is

      $$1.99^n \text{ where } n = 41 \to 1790507451731.9128ms \to 20.7235d$$

      Flitwick can complete and run his algorithm in

      $$17 + n^3 \text{ where } n = 41 \to 17d + 68921ms \to 17.0000007977d$$

      Therefore the company *should* pay him to develop the better algorithm as it will save them 3 days time.

   (b) Let $n = 10^6, f(n) = n^{2.00}, g(n) = n^{1.99}$ and $t = 2$ days.

      i. The time it will take the original algorithm to complete is

      $$n^{2.00} \text{ where } n = 10^6 \to 1000000000000ms \to 11.5741d$$

      Flitwick can complete and run his algorithm in

      $$2 + n^{1.99} \text{ where } n = 10^6 \to 2d + 870963589956.0806ms \to 12.0806d$$

Therefore the company *should not* pay him to develop the better algorithm as it will take an extra day and a half to complete.

---

5. *Using the mathematical definition of Big-O, answer the following. Show your work.*
   (a) Is $2^{nk} = O(2^n)$ for $k > 1$?
      i. No. 2^{nk} will always grow faster that 2^n.

$$2^{nk} \to (2^n)^k \to \left|(2^n)^k\right| \not\leq c\,|2^n|$$

   (b) Is $2^{n+k} = O(2^n)$, for $k = O(1)$?
      i. Yes. 2^k is constant, therefore

$$2^{n+k} \to 2^n 2^k \to |2^n \cancelto{c}{2^k}| \leq c\,|2^n|$$

---

6. *Is an array that is in sorted order also a min-heap? Justify.*
   (a) Technically no, they are not the same. They have differing data structures, however they are more similar than not upon further inspection. A sorted array has the form `[1, 2, 3, 4, 5]` while a min-heap has the form

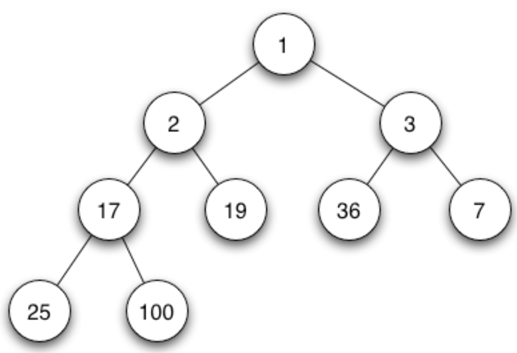

Figure 1: A Sample Min-Heap

with a corresponding data structure similar to the sample code below.

Sample Min-Heap Data Structure
```python
class Node:
    left = left_node_class_object   # Must be greater than Node
    right = right_node_class_object # Must be greater than Node
    value = node_value
```

As is evident the fundamental data structures expressing the two are not similar in the slightest. This being said however, a sorted array will correspond the following min-heap

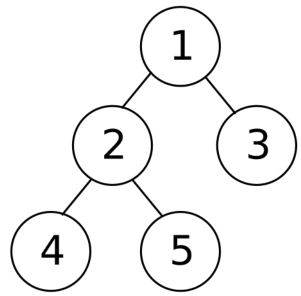

Figure 2: The Min-Heap for our Array

When the min-heap is accessed top-down, left-to-right it will have a one-to-one correspondence to our array. So to put it succinctly, the two data structures are not the same, however they have similar appearance and behavior.