CSCI 570 Homework 1

Due Date: Sept. 07, 2023 at 11:59 P.M.

1. Arrange these functions under the Big-$\mathcal{O}$ notation in increasing order of growth rate with $g(n)$ following $f(n)$ in your list if and only if $f(n) = \mathcal{O}(g(n))$ (here, $\log(x)$ is the **natural logarithm**[1] of x, with the base being the Euler's number e) :

$$2^{\log(n)}, 2^{3n}, 3^{2n}, n^{n\log(n)}, \log(n), n\log(n^2), n^{n^2}, \log(n!), \log(\log(n^n)).$$

$$\mathcal{O}(\log(n)) = \mathcal{O}(\log(\log(n^n))) = \mathcal{O}(\log(n) + \log(\log(n))), 2^{\log(n)},$$
$$\mathcal{O}(\log(n!)) = \mathcal{O}(n\log(n^2)), 2^{3n} = 8^n, 3^{2n} = 9^n, n^{n\log(n)}, n^{n^2}.$$

2. Show by induction that for any positive integer k, $(k^3 + 5k)$ is divisible by 6.

Base case: for $k = 1$, $1 + 5 = 6$, which can be divisible by 6.

Induction step: suppose for $k = n$, where n is any integer greater than 0, $n^3 + 5n$ can be divisible by 6. Therefore, we have

$$(k+1)^3 + 5(k+1) = n^3 + 3n^2 + 8n + 6 = (n^3 + 5n + 6) + (3n^2 + 3n)$$
$$= (n^3 + 5n + 6) + 3n(n+1)$$

As $(n^3 + 5n + 6)$ is divisible by 6, we have $3n(n+1)$ remains. We note that $n(n+1)$ is divisible by 2 as either n or $n+1$ is an even number, thus $3n(n+1)$ can be divided by $2 \times 3 = 6$, which concludes our proof.

3. Show that $1^3 + 2^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$ for every positive integer n using induction.

Base case: for $n = 1$, one can verify that $1^3 = 1 = \frac{1^2 2^2}{4}$.
Induction step: Suppose the statement holds true for any positive in-

[1] https://en.wikipedia.org/wiki/Natural_logarithm

teger $m \leq n$. Then, by direction calculation, we have

$$\sum_{k=1}^{n+1} k^3 = \frac{n^2(n+1)^2}{4} + (n+1)^3 = \frac{(n+1)^2}{4} \cdot \left(n^2 + 4(n+1)\right)$$
$$= \frac{(n+1)^2}{4} \cdot (n+2)^2 = \frac{(n+1)^2(n+2)^2}{4},$$

where the first step uses the induction hypothesis, and the third step follows from the fact that $n^2 + 4n + 4 = (n+2)^2$. Therefore, by mathematical induction, we have shown that the statement holds true for every positive integer n.

4. Consider the following prime filtering algorithm that outputs all the prime numbers in $2, \ldots, n$ (the pseudo code is presented in Algorithm 1).

   - Please prove this algorithm is correct (that is, a positive integer k that $2 \leq k \leq n$ is a prime if and only if $isPrime(k) = \textbf{True}$).

     A prime number (or a prime) is a natural number greater than 1 that is not a product of two smaller natural numbers. That is, for any prime number p, there does not exist two positive integers$x, y \geq 2$ such that $p = x \cdot y$. On the other hand, for any non-prime number z, there exists two positive integers $x, y \geq 2$ such that $z = x \cdot y$. Thus, any non-prime number z would be removed by Line 6 of Algorithm 1 when $i = x$ and $j = y$, and the prime numbers would be filtered.

   - Please calculate the time complexity under the Big-$\mathcal{O}$ notation.

     By direct calculation, we have

$$\sum_{i=2}^{n} \left\lfloor \frac{n}{i} \right\rfloor \leq \sum_{i=2}^{n} \frac{n}{i} = n \cdot \left(\sum_{i=1}^{n} \frac{1}{i} - 1 \right) = \mathcal{O}(n \log n),$$

     where the last step follows from the fact that $\sum_{i=1}^{k} \frac{1}{i} = \mathcal{O}(\log(n))$. The sum $\sum_{i=1}^{k} \frac{1}{i}$ is called the k-th Harmonic Number[2], which is important in various branches of number theory. To show that $\sum_{i=1}^{k} \frac{1}{i} =$

[2] https://en.wikipedia.org/wiki/Harmonic_number

$\Theta(\log(k))$, one can simply verify the following inequalities:

$$\int_1^{k+1} \frac{dx}{x} = \sum_{i=1}^{k} \int_i^{i+1} \frac{dx}{x} \leq \sum_{i=1}^{k} \frac{1}{i} \leq 1 + \sum_{i=2}^{k} \int_{i-1}^{i} \frac{dx}{x} = 1 + \int_1^k \frac{dx}{x}.$$

Following the Newton-Leibniz formula, one can show that $\int_1^{k+1} \frac{dx}{x} = \ln(k+1)$, which concludes the proof of $\sum_{i=1}^{k} \frac{1}{i} = \Theta(\log(k))$.

Algorithm 1 Prime Filtering

```
1: Input: a positive integer $n \geq 2$
2: initialize the Boolean array $isPrime$ such that $isPrime(i) = $ True for $i = 2, \ldots, n$
3: for $i = 2 \ldots n$ do
4:   for $j = 2 \ldots \lfloor \frac{n}{i} \rfloor$ do
5:     if $i \times j \leq n$ then
6:       $isPrime(i \times j) \leftarrow$ False
7:     end if
8:   end for
9: end for
```

5. Amy usually walks from Amy's house ("H") to SGM ("S") for CSCI 570. On her way, there are six crossings named from A to F. After taking the first course, Amy denotes the six crossings, the house, and SGM as 8 nodes, and write down the roads together with their time costs (in minutes) in Figure 1. Could you find the shortest path from Amy's house to SGM? You need to calculate the shortest length, and write down all the valid paths.

   There are two shortest paths of 21 minutes: H $\Rightarrow$ A $\Rightarrow$ B $\Rightarrow$ F $\Rightarrow$ S, and H $\Rightarrow$ D $\Rightarrow$ E $\Rightarrow$ F $\Rightarrow$ S.

6. According to the Topological Sort for DAG described in Lecture 1, please find one possible topological order of the graph in Figrue 2. In addition, could you find all the possible topological orders?

   There are 7 possible topological orders: ABCGDEF, ABGCDEF, ABGDCEF, AGBCDEF, AGBDCEF, GABCDEF, GABDCEF. One can start with the longest chain ABCEF, and consider adding D and G sequentially. Clearly, D can only be placed inbetween B and C, or C and E. For ABDCEF, we have three ways of adding G (before A,B,D). For ABCDEF, we have four ways (before A,B,C,D).

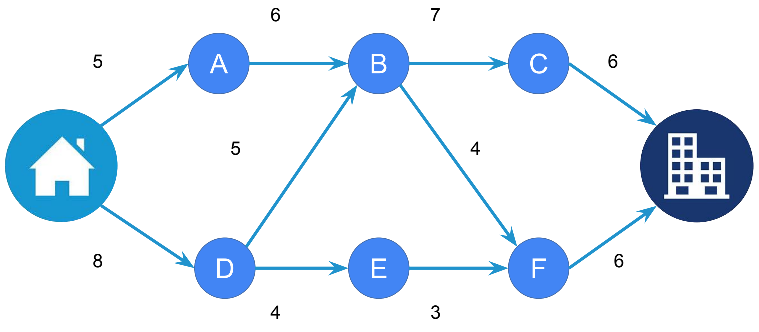

Figure 1: Problem 4's Graph

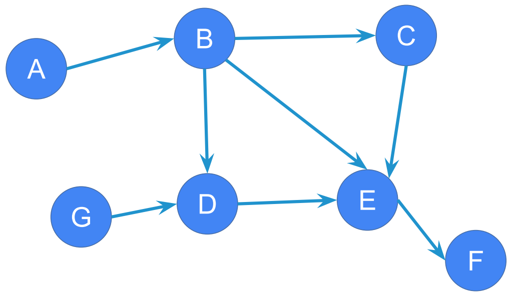

Figure 2: Problem 5's Graph

7. A binary tree[3] is a rooted tree in which each node has two children at most. A complete binary tree is a special type of binary tree where all the levels of the tree are filled completely except the lowest level nodes which are filled from as left as possible. For a complete binary tree T with k nodes, suppose we number the node from top to down, from left to right with 0, 1, 2, ..., $(k-1)$. Please solve the following two questions:

   - For any of the left most node of a layer with label t, suppose it has at least one child, prove that its left child is $2t+1$.
   - For a node with label t and suppose it has at least one child, prove that its left child is $2t+1$.

[3]`https://en.wikipedia.org/wiki/Binary_tree`

Since this is a complete tree, for all the layers before the node t, suppose it is layer $L(L = 1, 2, 3, ...)$, we have its number of node as 2^{L-1}.

- If the node is the root node its label is 0, its left child is 1, which satisfies $1 = 2 \times 0 + 1$.

  For the node with label t, suppose it is on the $L + 1$ layer, we have $\sum_{i=0}^{L} 2^i = t$. Since this layer $L + 1$ has $2^{L+1} - 1$ nodes on its right, following the good perspect $2^{L+1} = \sum_{i=0}^{L} 2^i + 1 = t + 1$, we have the label of its left child as $t + (2^{L+1} - 1) + 1 = t + (t + 1 - 1) + 1 = 2t + 1$.
- For any node that has label t, if it is the root node, we have proved that its left child is $2t + 1$.

  If it not the root node, suppose we have k nodes on the left of this layer, we have the label of the left-most node of this layer as $t - k$. Following the proof of Q1, its left child is $2(t - k) + 1$. Since this is a complete tree, suppose the node with label t has at least one child, its left child should be $2 \times (t - k) + 1 + 2 \times (k) = 2t + 1$.

8. Consider a full binary tree (all nodes have zero or two children) with k nodes. Two operations are defined: 1) *removeLastNodes()*: removes nodes whose distance equals the largest distance among all nodes to the root node; 2) *addTwoNodes()*: adds two children to all leaf nodes. The cost of either adding or removing one node is 1. What is the time complexity of these two operations, respectively? Suppose the time complexity to obtain the list of nodes with the largest distance to the root and the list of leaf nodes is both $O(1)$.

   The time complexity for *removeLastNode()* is $O(k)$, *addTwoNodes()* is $O(k)$. For *removeLastLayer()*, suppose we have $k = 2^L - 1$ nodes as a complete tree, its last layer has $2^{L-1} = \frac{k+1}{2}$ nodes. As each operation cost is 1, $O(\frac{k+1}{2}) = O(k)$. Similarly, the time complexity for *addTwoNodes()* is also $O(k)$, since the number of leaf nodes in a full binary tree = the number of nodes with 2 children + 1. Thus the number of leaf nodes is always $\frac{k+1}{2}$.

9. Given a sequence of n operations, suppose the i-th operation cost 2^{j-1} if $i = 2^j$ for some integer j; otherwise, the cost is 1. Prove that the amortized cost per operation is $O(1)$.

Total cost is

$$\sum_{1\le i\le n, i\ne 2^j, j\in\mathbf{Z}+} 1 + \sum_{1\le i\le n, i=2^j, j\in\mathbf{Z}+} 2^{j-1} = \sum_{1\le i\le n} 1 + \sum_{1\le i\le n, i=2^j, j\in\mathbf{Z}+} (2^{j-1}-1)$$
$$= n - \sum_{i=-1,0,1,2,\ldots,floor(\log n-1)} (2^i - 1) = n - \log n + 2^{floor(\log n)} - 1 \le 2n$$

log in the equation is with base as 2. Thus the amortized cost per operation is $O(2n/n) = O(1)$.

10. Consider a singly linked list as a dictionary that we always insert at the beginning of the list. Now assume that you may perform n insert operations but will only perform one last lookup operation (of a random item in the list after n insert operations). What is the amortized cost per operation?

For each insert operation, the time cost is only $\mathcal{O}(1)$, thus the n insert operations will have $\mathcal{O}(n)$ total time cost. The expected time cost of the look-up operation of a random item in an n-item list is

$$\sum_{k=1}^{n} \Pr(\text{“the target item is the k-th item”}) \cdot k = \sum_{k=1}^{n} \frac{k}{n} = \frac{n+1}{2} = \mathcal{O}(n).$$

So the amortized time cost per operation is $\mathcal{O}(1)$.