NAME: NIKHIL SATHYANARAYANA SHASTRY
USC ID: 8464475401
Email: nsathyan@usc.edu

CSCI570 - Analysis of Algorithms (HW1)

Q1. Arrange these functions under the Big-O notation in increasing order of growth rate with g(n) following f(n) in your list if and only if f(n) = 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)).

Solution: In the given question, after careful analysis of all the given growth rates and keeping in mind that the given log functions are in the natural logarithmic form, the increasing orders of the growth rates g(n) under the Big-O notation are as follows:

$$\log(n) = \log(\log(n^n)) < 2^{\log(n)} < \log(n!) = n\log(n^2) < 2^{3n} < 3^{2n} < n^{n\log(n)} < n^{n^2}$$

Q2. Show by induction that for any positive integer k, (k^3 + 5k) is divisible by 6.

Solution: To Prove that $(k^3 + 5k)$ is divisible by 6 using mathematical induction, we need to use the following steps:

a. **Base case:** Let us take an integer $k = 1$ and prove that the above statement holds true for $k = 1$.

   For $k = 1$,

   $(1)^3 + 5(1) = 6$

   Since 6 is divisible by 6, the base case holds true for the given equation.

b. **Induction Hypothesis:** Assume that for some positive integer $k = n$, the expression $(k^3 + 5k)$ is divisible by 6.

   This is going to be our induction hypothesis.

c. **Induction Step:** We want to prove that if the statement holds true for $k = n$, it must also be true for $k = n + 1$

Let $k = n + 1$,

Then we need to simplify the following:

$(n+1)^3 + 5\,(n+1)$
$n^3 + 1 + 3n + 3n^2 + 5n + 5$
$n^3 + 5n + 3n^2 + 3n + 6$

Since we know that $n^3 + 5n$ is divisible by 6 from our induction hypothesis we can simplify the rest of our equation above to check if it is divisible by it is divisible by 6.

By taking 3 as the common factor in the remaining part of our equation we get:

$3(n^2 + n + 2)$

This shows that the above equation is divisible by 3. Now since a number should be divisible both by 2 and 3 for a number to be divisible by 6 we need to check whether $n^2 + n + 2$ is divisible by 2. To do this we will need to account for 2 cases of n.

i. If n is even, it can be written as $n = 2a$ for some positive integer a. In such a case the equation becomes as follows:

$$((2a)^2 + 2a + 2)$$
$$((4a^2 + 2a + 2))$$
$$2(2a^2 + a + 1)$$

This clearly shows that for a case where n is even, it is divisible by 2

ii. If n is odd, it can be written as $n = 2a + 1$ for some positive integer a. In such a case the equation becomes as follows:

$$((2a+1)^2 + 2a + 1 + 2)$$
$$\big((4a^2 + 4a + 1) + 2a + 1 + 2\big)$$
$$4a^2 + 6a + 4$$
$$2(2a^2 + 3a + 2)$$

This clearly shows that for a case where n is odd, it is divisible by 2

Since we have shown that $3(n^2 + n + 2)$ is divisible by 2 and through our induction hypothesis that $n^3 + 5n$ is divisible by 6, it shows that the entire equation $n^3 + 5n + 3n^2 + 3n + 6$ is divisible by 6. This proves that the given equation $(k^3 + 5k)$ is divisible by 6 for $k = n + 1$.

Therefore, it is proved by mathematical induction that the given statement $(k^3 + 5k)$ is divisible by 6 holds true for any positive integer k.

<u>**Q3.**</u> Show that $1^3 + 2^3 + \cdot\cdot\cdot + n^3 = \frac{(n^2(n+1)^2)}{4}$ for every positive integer n using induction.

<u>**Solution:**</u> To Prove that $1^3 + 2^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$ for every positive integer n using mathematical induction, we need to use the following steps:

a. <u>**Base case:**</u> Let us take an integer $n = 1$ and prove that the above statement holds true for $n = 1$.

   For $n = 1$,

   $$1^3 = \frac{1^2(1+1)^2}{4} = \frac{1*4}{4} = 1$$

   Since the LHS is equal to the RHS, the base case holds true for the given equation.

b. <u>**Induction Hypothesis:**</u> Assume that for some positive integer $k = l$, the expression will be:

   $1^3 + 2^3 + \cdots + l^3 = \frac{l^2(l+1)^2}{4}$.

   This is going to be our induction hypothesis.

c. <u>**Induction Step:**</u> We want to prove that if the statement holds true for $k = n$, it must also be true for $k = l + 1$

   Let $k = l + 1$,

   Then we need to simplify the following such that the LHS is the same as the RHS:

   $$1^3 + 2^3 + \cdots + (l+1)^3 = \frac{(l+1)^2(l+2)^2}{4}$$

   Now let us simplify the left-hand side to check whether it equates to the right hand side.

   In the below equation we can write $1^3 + 2^3 + \cdots + l^3\ as\ \frac{l^2(l+1)^2}{4}$ because this is our induction hypothesis. After substituting we will get the following

   $$1^3 + 2^3 + \cdots + l^3 + (l+1)^3 = \frac{l^2(l+1)^2}{4} + (l+1)^3$$
   $$1^3 + 2^3 + \cdots + l^3 + (l+1)^3 = \frac{l^2(l+1)^2 + 4(l+1)^3}{4}$$

$$1^3 + 2^3 + \cdots + l^3 + (l+1)^3 = \frac{l^2(l+1)^2 + 4(l+1)(l+1)^2}{4}$$
$$1^3 + 2^3 + \cdots + l^3 + (l+1)^3 = \frac{(l+1)^2(l^2 + 4(l+1))}{4}$$
$$1^3 + 2^3 + \cdots + l^3 + (l+1)^3 = \frac{(l+1)^2(l^2 + 4l + 4)}{4}$$
$$1^3 + 2^3 + \cdots + l^3 + (l+1)^3 = \frac{(l+1)^2(l+2)^2}{4}$$

Since we were able to show that the given statement holds true for both $k = l$ and $k = l + 1$ we can conclude that for every positive integer n the statement $1^3 + 2^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$ will hold true by mathematical induction.

Q4. 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 $2 \le k \le n$ is a prime if and only if $isPrime(k) = True$).
- Please calculate the time complexity under the Big-O notation.

Algorithm 1 Prime Filtering

```
1: Input: a positive integer $n \ge 2$
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 $i \times j \le n$ then
6:       $isPrime(i \times j) \leftarrow$ False
7:     end if
8:   end for
9: end for
```

Solution:

A. We are required to prove that the prime filtering algorithm is correct.

I. To Prove that if k such that $2 \le k \le n$ is prime, then $isPrime(k) = True$

- Let k be a prime number such that $2 \le k \le n$, then k is divisible by itself and 1.
- In the algorithm we can see that $isPrime(i * j)$ is false only when $i * j \le n$ is true for $i = 2, \ldots, n$ and $j = 2, \ldots, floor(n/i)$.
- The above condition $i * j \le n$ will always be False for a prime number k as it is divisible only by 1 and itself and $(1 * k)$ will not occur in the given algorithm.

Therefore, for a prime number k such that $2 \le k \le n$, $isPrime(k) = True$.

II. To prove that if $isPrime(k) = True$ for k such that $2 \leq k \leq n$, then k is a prime number.

- In the algorithm we can see that $isPrime(i * j)$ is false only when $i * j \leq n$ is true for $i = 2, \ldots, n$ and $j = 2, \ldots, floor(n/i)$.
- Hence, $isPrime$ will be true for a number that is not divisible by i and j where $i = 2, \ldots, n$ and $j = 2, \ldots, floor(n/i)$.
- The number that are not divisible by i and j will have only 1 and itself as its divisors. These numbers are called prime numbers.

Therefore, if $isPrime(k) = True$ for k such that $2 \leq k \leq n$, then k is a prime number.

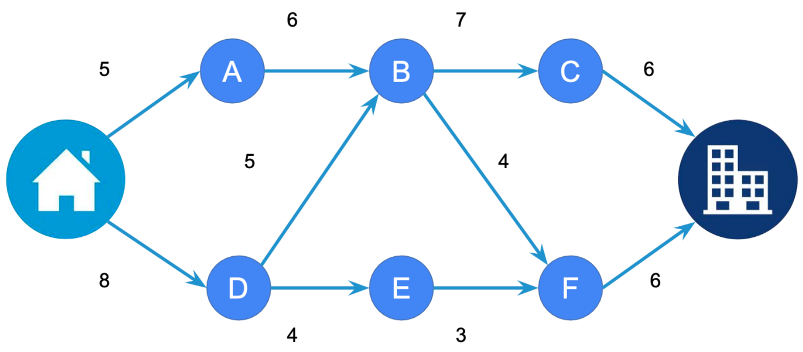
Combining I and II we can prove that a positive integer k such that $2 \leq k \leq n$ is a prime number if and only if $isPrime(k) = True$. This shows that the algorithm is correct.

B. Calculation for the time complexity under the Big-O notation is as follows:

```
1: **Input**: a positive integer $n \geq 2$
2: initialize the Boolean array $isPrime$ such that $isPrime(i) = \textbf{True}$ for $i = 2, \ldots, n$
3: **for** $i = 2 \ldots n$ **do** $-----------\rightarrow \boldsymbol{O(n)}$
4:   **for** $j = 2 \ldots \left\lceil \frac{n}{i} \right\rceil$ **do** $-----------\rightarrow \boldsymbol{O(logn)}$
5:     **if** $i * j \leq n$ **then**
6:       $isPrime(i \times j) \leftarrow$ **False**
7:     end **if**
8:   end **for**
9: end **for**
```

The time complexity of the outer loop is $O(n)$ as it is iterating from $2, \ldots, n$
The time complexity of the inner loop is $O(logn)$ as it iterates approximately $\frac{n}{i}$ $times$ for each i.
Therefore, the total time complexity of the algorithm shown above is $\boldsymbol{O(n * log(n))}$

Q5. 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.

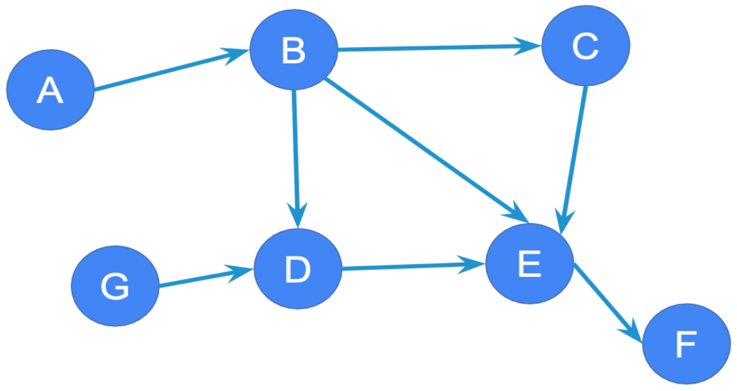


Solution: In the above graph we can observe that there are 5 valid paths that can be followed to reach from Amy's house ("H") to SGM ("S"). They are as follows:

1. H -> A -> B -> F-> S	Cost = (5 + 6 + 4 + 6 = 21)
2. H -> A -> B -> C-> S	Cost = (5 + 6 + 7 + 6 = 24)
3. H -> D -> B -> F-> S	Cost = (8 + 5 + 4 + 6 = 23)
4. H -> D -> E -> F-> S	Cost = (8 + 5 + 3 + 6 = 21)
5. H -> D -> B -> C-> S	Cost = (8 + 5 + 7 + 6 = 26)

As we can see from the above listed valid paths that there are 2 paths that can be followed which are of the same length. They are Path 1 and Path 4 which are of the length 21 which is the shortest path from Amy's house to SGM hall.

Q6. According to the Topological Sort for DAG described in Lecture 1, please find one possible topological order of the graph in Figure 2. In addition, could you find all the possible topological orders?



Solution: From the above graph all the possible topological orders that can be generated using topological sort for the given DAG are as follows:

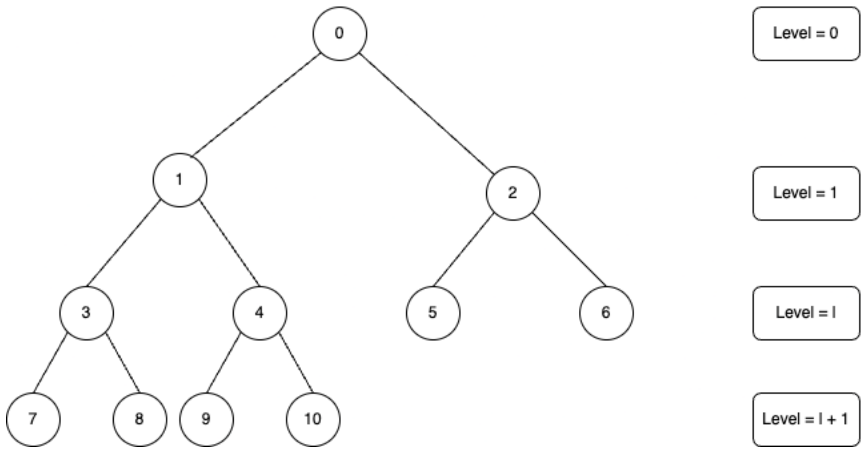
1. A -> G -> B -> C-> D -> E -> F
2. A -> G -> B -> D-> C -> E -> F
3. G -> A -> B -> C-> D -> E -> F
4. G -> A -> B -> D-> C -> E -> F
5. A -> B -> C -> G-> D -> E -> F
6. A -> B -> G -> D-> C -> E -> F
7. A -> B -> G -> C-> D -> E -> F

Q7. A binary tree 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.

Solution:

a.



In a complete binary tree all the levels of the tree are filled completely except the lowest level nodes which are filled from the left to right. Given a complete binary tree, the number of nodes at level l will be 2^l. Now, let us take into consideration the leftmost node at the level l and let it have a label t.

=> We know that the last level l contains 2^l nodes.

=> The position of the leftmost node at level l will be $2^l - 1$

From the above information we can say that $\boldsymbol{t = 2^l - 1}$ where t is the leftmost node at level l.

To find the left child of t we need to move to level $l + 1$

=> The leftmost node at level $l + 1$ will be at the position $2^{(l+1)} - 1$

By substituting this value back into the expression above we get:

=> $2^{(l+1)} - 1 = 2^l * 2 - 1$

This can be written as:

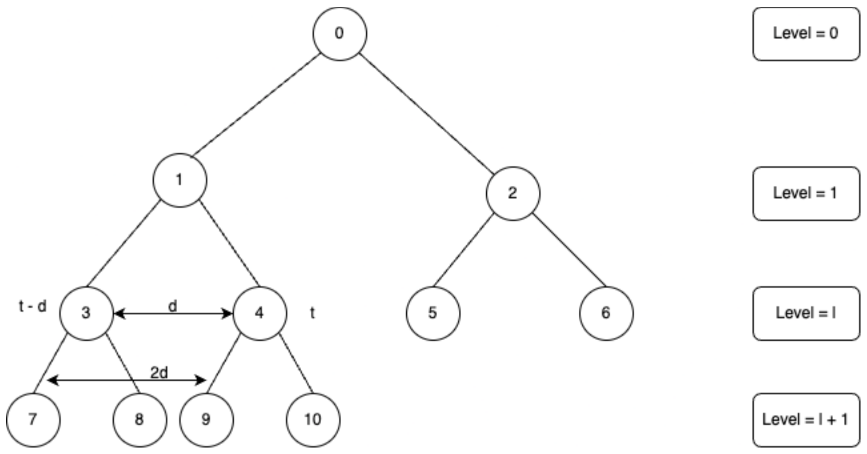
$$=> 2^{(l+1)} - 1 = 2^l * 2 - 2 + 1$$
$$=> 2^{(l+1)} - 1 = 2(2^l - 1) + 1$$

Since we know that $t = 2^l - 1$

$$=> 2^{(l+1)} - 1 = 2t + 1$$

This proves that the left child of the leftmost node t at level l is $2t + 1$.

b.



For a node with label t and suppose it has at least one child, we need to prove that its left child is $2t + 1$.

Let us assume that node t is at level l and is d steps away from the leftmost node in the same level.

- The leftmost node in level l is given by $2^l - 1$.
- We can express the position of the leftmost node in level l with respect to its distance from node t as $\mathbf{2^l - 1 = (t - d)}$

To find the left child of node t, we need to move to the next level which is $(l + 1)$.

Further, at each level l, every node has 2 children. Hence, the left child of t is $2d$ steps away from the leftmost node at level $(l + 1)$.

From our previous proof we have established that for the leftmost node at level l, it's left child is given by $\mathbf{2(t - d) + 1}$.

To find the left child of node t which is $2d$ steps away from the leftmost node at level $l + 1$, we need to add $2d$ steps to the position of the left child of the leftmost node at level l (we need to add $2d$ steps to the equation from the step above):

$$= 2(t - d) + 1 + 2d$$

$$= 2t - 2d + 1 + 2d$$

$$= \mathbf{2t + 1}$$

This proves that the left child of a node with label t at level l is always $2t + 1$.

Q8. 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).

Solution:

As given in the problem statement for a full binary tree with 'k' nodes, to find the time complexity for the two operations removeLastNodes() and addTwoNodes() we need to analyse them separately as 2 different cases:

a. In the case of the removeLastNodes() operation it's function is to remove nodes whose distance equals the largest distance among all nodes to the root node. It is also stated that the cost for removing a node is $O(1)$. The maximum number of nodes at the maximum distance from the root node in a full binary tree is given by $\frac{(k+1)}{2}$.

   Since the cost for removal of each node is 1, the time complexity of this operation is $O(\frac{(k+1)}{2})$ which is equivalent to $\boldsymbol{O(k)}$.

b. In the case of the addTwoNodes() operation it's function is to add two children to all leaf nodes of the full binary tree. It is also given to us in the problem statement that it takes $O(1)$ to find the list of all the leaf nodes of the tree. Further we know that the maximum number of leaf nodes in a full binary tree is $\frac{(k+1)}{2}$.

   Since the cost for the addition of each node is 1 and there are 2 additions done for each leaf node, the time complexity will be $\frac{(k+1)}{2} * 2 = (k + 1)$. Therefore, the time complexity is $O(k + 1)$ which is equivalent to $\boldsymbol{O(k)}$.

Q9. 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).

Solution: As given in the problem statement in a sequence of n operations the cost of the i^{th}operation can be divided into 2 cases:

Case 1: Cost = 2^{j-1} when $i = 2^j$
Case 2: Cost = 1 in all other cases.

The total number of operations = n

The calculation for the total cost can be split into 2 parts:

a. If the i^{th} operation is a power of 2, then the cost of such an operation is 2^{j-1} . Here j ranges from 0 to $log_2(n) - 1$. The total cost for *case a* is as follows:

$$(2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{\log_2(n)-1}) = \sum_{j=0}^{\log_2(n)-1} 2^{j-1}$$

b. For n-1 operations the cost of each operation is 1. This is the case in which the i^th operation is not a power of 2. The total cost for *case b* is as follows:

$$(1 + 1 + 1 \ldots + 1) * n = n$$

Finally, the total cost for both cases a & b combined is as follows:

$$\text{Total cost} \leq\ n\ +\ \sum_{j=0}^{j=log_2(n)} 2^{(j-1)}$$
$$\text{Total cost} \leq\ n + \frac{2^{\log_2(n)+1} - 1}{2}$$
$$\text{Total cost} \leq\ n + \frac{2^{\log_2(n)} * 2 - 1}{2}$$
$$\text{Total cost} \leq\ n + 2^{\log_2(n)} - \frac{1}{2}$$
$$\textbf{Total cost} \leq\ \boldsymbol{n + n - \frac{1}{2} \approx 2n}$$

$$\text{Finally, Amortized cost } = \frac{Total\ Cost\ of\ operations}{Number\ of\ operations}$$

$$\text{Amortized cost } = \frac{2n}{n} \leq 2$$

As the value of n increases the amortized cost per operation is approximately 2 which is a constant. This proves that for the given problem statement the amortized cost per operation is constant i.e. $O(1)$.

Q10. 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?

Solution: From the given information in the problem statement, we can calculate the total number of operations to be as follows:

Number of insertions to the linked list = n

Number of lookup operations on the list = 1

Hence, the total number of operations = $n + 1$

Now, Let us calculate the total cost of the operations:

a. Since we are inserting at the beginning of the list, the total cost of each insertion will be $O(1)$. Finally, since we have to perform n such insertions to the linked list, the total cost of inserting into the list will be $O(1) * n = O(n)$
b. Now, for the lookup operation, we can consider the worst case in which we will need to traverse through the entire linked list to find the required element, which will cost us $O(n)$. Since we run the lookup operation only once, the total cost for performing one lookup will be $O(n) * 1 = O(n)$

Therefore, our total cost for n insert operations and 1 lookup operation will be:

$$= O(n) + O(n)$$

$$= O(n)$$

Finally, $\text{Amortized cost} = \frac{Total\ Cost\ of\ operations}{Number\ of\ operations}$

$$\text{Amortized cost} = \frac{O(n)}{n+1}$$

$$\text{Amortized cost} = O(1)$$