CS 570 HW 1

Mounika Mukkamalla - 1001219359

September 7 2023

1

Time complexity ascending order:

$O(log(n)) = O(log(log(n^n))), 2^{log(n)}, O(log(n!)) = O(nlog(n^2)), 2^{3n}, 3^{2n}, n^{nlog(n)}, n^{n^2}$

2

Let's prove the given statement by induction.

Base Case: For $k = 1$, We need to prove $k^3 + 5k$ is divisible by 6.
By substituting $k = 1$, $k3 + 5k = 1 + 5 = 6$
6 is divisible by 6.
Hence base case is proved.

Inductive Hypothesis: Assume ∃ r for which given statement is true.

$$\exists r, p \in N \ni r^3 + 5r = 6p$$

Inductive Step: We need to prove the given statement holds true for $r + 1$
By substituting $r + 1$ in given expression:
$(r + 1)^3 + 5(r + 1) = r^3 + 3r^2 + 3r + 1 + 5r + 5 = (r^3 + 5r) + (3r^2 + 3r^2 + 6)$
$= 6p + 3r(r + 1) + 6$————————- Eq 1
Here, consider $r(r + 1)$, We know that in a pair of consecutive integers, exact one integer is divisible by 2.
Hence $\exists c \in N \ni r(r + 1) = 2c$
Therefor, Eq 1 becomes: $6p + 3(2c) + 6 = 6(p + c + 1)$ implies divisible by 6
Hence proving the given statement is true for $r + 1$.
Inductive Step is proved.
Given statement holds true $\forall r \in N$

3

Given equation: $1^3 + 2^3 + ... + n^3 = n^2(n+1)^2/4$
Base Case: for $n = 1$,
$LHS = 1^3 = 1$
$RHS = 1^2(1+1)^2/4 = 1$
LHS = RHS.
Hence given equation is true for base case.

Inductive Hypothesis: Assume $\exists r \in N$ for which given equation is true.
$1^3 + 2^3 + ... + r^3 = r^2(r+1)^2/4$

Inductive Step: We need to prove the given equation is true for $r + 1$.
LHS: By substituting $(r+1) : (1^3 + 2^3 + ... + r^3 + (r+1)^3)$
$= r^2(r+1)^2/4 + (r+1)^3$
RHS: By substituting $(r+1)$:
$((r+1)^2(r+2)^2)/4 = ((r^2)(r+1)^2 + (4r+4)(r+1)^2)/4$
$= r^2(r+1)^2/4 + (r+1)(r+1)^2$
$= r^2(r+1)^2/4 + (r+1)^3$
LHS = RHS
Inductive Step is true.
Hence given equation is true $\forall n \in N$

4

1. Let's prove by contradiction.

Case1: Assume that there is prime number $p <= n$, for which $isPrime()$ is false. We need to prove that $isPrime$ can't be false.
According to our algorithm, $isPrime()$ is marked as false when $\exists$, 2 integers i, j for which $2 <= i * j <= n$
But since p is prime, there are only 2 multiples of p which is 1 and p. and 1 is not greater than 2. Hence contradicting the assumption and algorithm.

Case2: Assume that there is composite number $c <= n$ for which $isPrime()$ is not marked false. We need to prove $isPrime()$ is computed as false
Since c is composite,
$\exists$ 2 integers, $2 <= k, l <= \sqrt{c} <= n \ni c = k * l$

Consider in the algorithm, $i = k$, inner loop will run from $j = 2$ to $j = [n/k]$
We know that $k * l = c <= n$ and hence $l <= (n/k)$
Since j loop will run till (n/k), and l is an integer lying in the range of j, we will reach a point where j=l and since $i * j <= n$, algorithm goes to next step and marks $isPrime(i * j)$ as false implying $isPrime(k * l)$ as false.
Hence algorithm marks $isPrime(c)$ as false, contradicting our assumption.

So, by contradiction, we proved Prime Filtering algorithm correctness.

2. Time complexity of given prime filtering algorithm is number of times we enter the inner loop and perform if condition.
for $i = 2$, j runs from 2 to $n/2$;
for $i = 3$, j runs from 2 to $n/3$;
.
.
.
for $i = n$, j runs from 2 to n/n;
Sum of number of times we reach inner loop =
$n/2 + n/3 + ... + n/n = n(1/2 + 1/3 + ...1/n) < n(1 + 1/2 + ... + 1/n)$
Sum of harmonic series, $1, 1/2, ..., 1/n = O(log(n))$
Hence time complexity = n(Sum of harmonic series) = O(nlogn)

5

There are total 5 paths in which Amy can go from Home(H) to SGM(S). Out of which 2 paths are shortest paths with 21 time cost. Below is the list of all 5 paths with path cost.
1. $H- > A- > B- > F- > S$; Path Cost: 21
2. $H- > A- > B- > C- > S$; Path Cost: 24
3. $H- > D- > B- > C- > S$; Path Cost: 26
4. $H- > D- > B- > F- > S$; Path Cost: 30
5. $H- > D- > E- > F- > S$; Path Cost: 21

1 and 5 are the shortest paths for Amy.

6

There are total 7 topological sorting paths that can be done:
1. A, B, C, G, D, E, F
2. A, B, G, C, D, E, F
3. A, B, G, D, C, E, F
4. A, G, B, D, C, E, F
5. A, G, B, C, D, E, F
6. G, A, B, C, D, E, F
7. G, A, B, D, C, E, F

7

In order to prove the given 2 statements, we need a prove:
St 0: Number of nodes in one level = Sum of all the nodes in the previous levels + 1
Assume the given graph contains l number of levels.
Since it is complete binary graph, every node has 2 children, hence number of nodes at a level l_1 is twice the number of nodes in the previous level except the last level.
So, number of nodes in each level are: $1, 2, 4, ...2^{l_1}$
Number of nodes in $l_1 + 1$ level, $N_{l_1+1} = 2^{l_1+1}$
Sum of all the nodes till l_1 : $S_{l_1} = 1 + 2 + 4 + ... + 2^{l_1} = 2^{l_1+1} - 1$
$S_{l_1} = N_{l_1+1} - 1$
Hence Statement 0 is true.

St 1:
Left most node is labeled as t.
According to the numbering given in the question, Sum of all the nodes in the previous levels would be t.
Based on our previous proof, number of nodes in current level = $t + 1$
Hence, last node of the current level = left most node + number of nodes in the level $-1 = t + (t + 1) - 1 = 2t$
Left most child node of node $'t'$ is next node of right most rode in the current level.
Hence Left most child node = right most node +1
Left most child node = 2t+1.

St 2:
Suppose we are at a level l, and consider node t somewhere in the mid of the level.
Let r be the left most node of level l.
Number of nodes from node r to node t excluding t including r = $t - r$
Since graph is complete binary and from question we definitely know that node r has left child, all the nodes before t from r have 2 children.
Number of children for all the nodes before t = 2(t-r)
From St 1, We know that left most child of node r is $2r + 1$
Hence left most child of node t, C_t = left most child of node r, C_r + number of nodes in between C_t and C_r
$C_t = (2r + 1) + 2(t - r)$
Hence, $C_t = 2t + 1$
Proved!

8

1) Given the tree is full binary tree.
$removeLastNodes()$ removes all the nodes whose distance is largest from the root node.
Nodes in the last level are the farthest from the root node.
Each remove operation is O(1)
Time complexity of $removeLastNodes()$ would be number of remove operations implies number of nodes in the last node.
Since the given tree is full binary tree, last level can be fully filled or half filled.
Let's consider the worst case complexity of full binary tree, which is complete binary tree with last level full.
For a complete binary tree, every node has 2 children, hence number of nodes at l_1 is twice the number of nodes in previous level.
So, in level 1 , number of nodes, $N_{l_1} = 1;$
$N_{l_2} = 2^{2-1}, N_{l_3} = 2^{3-1}, ..., N_{l_t} = 2^{t-1}$
Given number of nodes in whole tree, $S = k$
S = sum of all nodes $= N_{l_1} + N_{l_2} + ... + N_{l_t} = 2^t - 1$
$k = 2^t - 1; k + 1 = 2^t; 2^{t-1} = (k + 1)/2; N_{l_t} = (k + 1)/2 = O(k)$
Worst case time complexity of $removeLastNodes()$ is Number of nodes in the last node of complete binary tree which is N_{l_t} which is $O(k)$

2)$addTwoNodes()$ complexity is number of nodes to be added.
In the worst case, number of nodes to add would be filling an entire last level.
In case 1, we proved $N_{l_1} = 2^{t-1}$, adding children to all nodes in l_t will fill l_{t+1}.
Implying number of addition operations to be done $= N_{l_{t+1}} = 2^t$
Again from case 1, we know that $k = 2^t - 1; 2^t = k + 1 = O(k)$
Hence time complexity for $addTwoNodes()$ operation $= N_{l_{t+1}} = O(k)$

9

$\exists k \ni 2^k < n < 2^{k+1}$
Let's denote operation cost of i^{th} operation as O_i
Sum of all operations in a sequence of n operations, $S_n = O_1 + O_2 + ...O_n$
Given $O_{2^r} = 2^r$, and for all other i, $O_i = 1$
Sum of all $(2^j)^{th}$ operations for integer $0 <= j <= k$:
$S_j = O_{2^0} + O_{2^1} + ...O_{2^k} = 1 + 2 + 4 + ...2^k = 2^{k+1} - 1$
Sum of all the remaining operations, $S_r = n - k - 1$
$S_n = S_j + S_r = 2^{k+1} - 1 + n - k - 1$
$= 2(2^k) + n - k - 2 < 2(n) + n - k - 2 < 2(n) + n = 3n$
Hence, $S_n < 3n$
Amortised cost = Sum of all operations cost / number of all operations.
$=S_n/n < 3 = O(1)$.
Hence Amortised cost per operation is constant, O(1).

10

Operation cost of insert operation, O_i: constant = O(1)
$O_i < c_1$
After n insert operations,
Operation cost of lookup operation, O_l: linear time = O(n)
$O_l < c_2n$ Total operation cost = (Sum of all insert operations cost + lookup operation cost)
$= (n(O_i) + O_l) < nc_1 + c_2n = (c_1 + c_2)n$
Total Operation cost = $O(n) < cn$
Amortised cost = Total Operation cost/ Number of operations
$= cn/(n+1) =$ O(1)
Amortised cost per operation = O(1)