COMP9020 20T1 Foundations of Computer Science

Week 7 Problem Set
Induction, Recursion, Complexity Analysis

[Show with no answers] [Show with all answers]

1. (Induction proofs)

   a. Prove by induction that $1 \cdot 1! + 2 \cdot 2! + \ldots + n \cdot n! \ = \ (n+1)! - 1$ for all $n \geq 1$ ($n \in \mathbb{N}$).

   b. Given the definition,

   $$\begin{aligned} s_1 &= 1 \\ s_{n+1} &= \tfrac{1}{1+s_n} \quad (n > 1) \end{aligned}$$

   prove by induction that

   $$s_n = \frac{\mathrm{FIB}(n)}{\mathrm{FIB}(n+1)}$$

   for all $n \geq 1$ ($n \in \mathbb{N}$).

   c. Suppose you would like to conclude that $P(n)$ is true for all $n \geq 0$ ($n \in \mathbb{N}$). For each of the following conditions, determine whether the condition is sufficient to prove this.

      i. $P(0)$ and $\forall n \geq 1\,(P(n-1) \Rightarrow P(n+1) \wedge P(n+2))$

      ii. $P(1)$ and $\forall n \geq 0\,(P(n+1) \Rightarrow P(n) \wedge P(n+2))$

      iii. $P(0)$ and $P(1)$ and $\forall n \geq 1\,(P(n) \wedge P(n+1) \Rightarrow P(n+2))$

      iv. $P(0)$ and $P(1)$ and $\forall n \geq 1\,(P(n) \Rightarrow P(n+2))$

      v. $P(0)$ and $P(1)$ and $\forall n \geq 1\,(P(n) \Rightarrow P(2 \cdot n) \wedge P(2 \cdot n+1))$

      vi. $P(0)$ and $P(1)$ and $\forall n \geq 1\,(P(2 \cdot n) \Rightarrow P(2 \cdot n-1) \wedge P(2 \cdot n+1))$

[hide answer]

a. Base case: $1 \cdot 1! = 1 = (1+1!) - 1$

Inductive step:

$$\begin{aligned} 1 \cdot 1! + 2 \cdot 2! + \ldots + n \cdot n! + (n+1) \cdot (n+1)! &= (n+1)! - 1 + (n+1) \cdot (n+1)! && \text{by induction hypothesis} \\ &= (1 + (n+1)) \cdot (n+1)! - 1 \\ &= (n+2) \cdot (n+1)! - 1 \\ &= (n+2)! - 1 && \text{by definition of } (.)! \end{aligned}$$

b. Base case: $s_1 = 1 = \frac{1}{1} = \frac{\mathrm{FIB}(1)}{\mathrm{FIB}(2)}$

Inductive step:

$$s_{n+1} = \frac{1}{1+s_n} = \frac{1}{1 + \frac{\mathrm{FIB}(n)}{\mathrm{FIB}(n+1)}} = \frac{1}{\frac{\mathrm{FIB}(n+1)+\mathrm{FIB}(n)}{\mathrm{FIB}(n+1)}} = \frac{1}{\frac{\mathrm{FIB}(n+2)}{\mathrm{FIB}(n+1)}} = \frac{\mathrm{FIB}(n+1)}{\mathrm{FIB}(n+2)}$$

c. Only conditions ii. and v. suffice:

   i. $P(0)$ and $\forall n \geq 1\,(P(n-1) \Rightarrow P(n+1) \wedge P(n+2))$ is not sufficient:
   From $P(0)$ it follows that $P(2)$ and $P(3)$, but the case $n = 1$ is not covered.

   ii. $P(1)$ and $\forall n \geq 0\,(P(n+1) \Rightarrow P(n) \wedge P(n+2))$ proves $P(n)$ for all $n \geq 0$:
   $P(1)$ implies $P(0)$ and $P(2)$, then $P(2)$ implies $P(3)$, and so on.

   iii. $P(0)$ and $P(1)$ and $\forall n \geq 1\,(P(n) \wedge P(n+1) \Rightarrow P(n+2))$ is not sufficient:
   The "first" instance of the implication is $P(1) \wedge P(2) \Rightarrow P(3)$, but $P(2)$ is not given.

   iv. $P(0)$ and $P(1)$ and $\forall n \geq 1\,(P(n) \Rightarrow P(n+2))$ is not sufficient:
   The "first" instance of the implication is $P(1) \Rightarrow P(3)$, hence the case $P(2)$ cannot be derived.

   v. $P(0)$ and $P(1)$ and $\forall n \geq 1\,(P(n) \Rightarrow P(2 \cdot n) \wedge P(2 \cdot n+1))$ proves $P(n)$ for all $n \geq 0$:
   $P(1)$ implies $P(2)$ and $P(3)$, $P(2)$ implies $P(4)$ and $P(5)$, and so on.

vi. $P(0)$ and $P(1)$ and $\forall n \geq 1\,(P(2 \cdot n) \Rightarrow P(2 \cdot n - 1) \land P(2 \cdot n + 1))$ is not sufficient:
The "first" instance of the implication is $P(2) \Rightarrow P(1) \land P(3)$, but $P(2)$ is not given.

2. (Recursive definitions)

Recall the recursive definition of a rooted tree:

$$\begin{array}{ll} \langle v;\ \rangle & \text{is a tree consisting only of a root node} \\ \langle r; T_1, T_2, \ldots, T_k \rangle\ \rangle & \text{is a tree with root } r \text{ and subtrees } T_1, T_2, \ldots, T_k \text{ at the root } (k \geq 1) \end{array}$$

Prove that in any rooted tree, the number of leaves is one more than the number of nodes with a right sibling.

Hint: This assumes a given order among the children of every node from left to right; see slide 22 (week 7) for an instance of this theorem.

[hide answer]

For a tree T, let $\ell(T)$ and $r(T)$ denote, respectively, the number of leaves and the number of vertices with a right sibling.

Base case:
A tree consisting of just a root has 1 leaf and no vertex with a right sibling:

$$T = \langle v; \rangle \ \Rightarrow\ \ell(T) = 1 = 0 + 1 = r(T) + 1$$

Inductive step:
Consider the tree $T = \langle r;\ T_1, T_2, \ldots, T_{k-1}, T_k \rangle$.

- The leaves in T are all the leaves of all the subtrees T_i. Hence, $\ell(T) = \sum_{i=1}^{k} \ell(T_i)$.
- The vertices in T with a right sibling are
  - all the vertices with a right sibling in all of the subtrees T_i
  - *and* the roots of all the subtrees $T_1, \ldots, T_{k-1}$ (but not T_k because it is the last child of the root r in tree T).

  Hence, $r(T) = (k-1) + \sum_{i=1}^{k} r(T_i)$.

From the induction hypothesis, $\ell(T_i) = r(T_i) + 1$ for all $1 \leq i \leq k$, it follows that

$$\begin{array}{rcll} \ell(T) & = & \sum_{i=1}^{k} \ell(T_i) & \\ & = & \sum_{i=1}^{k} (r(T_i) + 1) & \text{by induction hypothesis} \\ & = & k + \sum_{i=1}^{k} r(T_i) & \\ & = & r(T) + 1 & \end{array}$$

3. (Recurrences)

Recall the recurrence for Mergesort:

- $T(1) = 0$
- $T(n) = 2T(\frac{n}{2}) + (n-1)$

Prove that $n \cdot (\log_2 n - 1) + 1$ is a valid formula for $T(n)$ for all $n = 2^k$ (with $k \geq 1$).

[hide answer]

Base case: $T(2^1) = 2 \cdot 0 + (2^1 - 1) = 1$; this is the same as $2^1 \cdot (\log_2 2^1 - 1) + 1 = 2 \cdot 0 + 1 = 1$

Inductive step:

$$\begin{array}{rcll} T(2^{k+1}) & = & 2 \cdot T(\frac{2^{k+1}}{2}) + (2^{k+1} - 1) & \text{by the recurrence} \\ & = & 2 \cdot T(2^k) + (2^{k+1} - 1) & \\ & = & 2 \cdot (2^k \cdot (\log_2 2^k - 1) + 1) + (2^{k+1} - 1) & \text{by induction hypothesis} \\ & = & 2^{k+1} \cdot (\log_2 2^k - 1) + 2 + 2^{k+1} - 1 & \\ & = & 2^{k+1} \cdot ((\log_2 2^k - 1) + 1) + 1 & \\ & = & 2^{k+1} \cdot k + 1 & \\ & = & 2^{k+1} \cdot (\log_2 2^{k+1} - 1) + 1 & \end{array}$$

4. (Asymptotic running times)

   a. Suppose you have the choice between three algorithms:

      i. Algorithm A solves your problem by dividing it into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.

ii. Algorithm B solves problems of size n by recursively solving two subproblems of size $n-1$ and then combining the solutions in constant time.

iii. Algorithm C solves problems of size n by dividing them into nine subproblems of size $\frac{n}{3}$, recursively solving each subproblem, and then combining the solutions in $\mathcal{O}(n^2)$ time.

Estimate the running times of each of these algorithms. Which one would you choose?

b. Order the following functions in increasing asymptotic complexity:

i. $(n-1)\cdot(n-2)\cdot\sqrt{n}$

ii. $\frac{3n}{\sqrt{n+1}}$

iii. $\sqrt{7n^3+3n+1}$

iv. $5n^{\log(\log(n))}$

v. $3n\log(n)+2n^2$

vi. $8+\log(n)\cdot(n-1)$

[hide answer]

a. C has the best asymptotic running time:

i. $T(n)=5\cdot T(\frac{n}{2})+\mathcal{O}(n)$. The Master Theorem with $d=2,\ \alpha=\log_2 5,\ \beta=1$ implies, since $\alpha>\beta$, that $T(n)=\mathcal{O}(n^{\log_2 5})=\mathcal{O}(n^{2.322})$.

ii. $T(n)=2\cdot T(n-1)+\mathcal{O}(1)$. The theorem on linear reductions with $c=2,\ k=0$ implies, since $c>1$, that $T(n)=\mathcal{O}(2^n)$.

iii. $T(n)=9\cdot T(\frac{n}{3})+\mathcal{O}(n^2)$. The Master Theorem with $d=3,\ \alpha=2,\ \beta=2$ implies, since $\alpha=\beta$, that $T(n)=\mathcal{O}(n^2\log n)$.

b. The correct ordering is (ii) < (vi) < (iii) < (v) < (i) < (iv), with (ii) having the best and (iv) the worst asymptotic complexity:

i. $(n-1)\cdot(n-2)\cdot\sqrt{n}\in\Theta(n^2\cdot n^{0.5})=\Theta(n^{2.5})$

ii. $\frac{3n}{\sqrt{n+1}}\in\Theta(n\cdot n^{-0.5})=\Theta(n^{0.5})$

iii. $\sqrt{7n^3+3n+1}=(7n^3+3n+1)^{0.5}\in\Theta(n^{1.5})$

iv. $5n^{\log(\log(n))}\in\Theta(n^{\log(\log(n))})$. For sufficiently large n, $\log(\log(n))>k$ for any given $k\in\mathbb{R}^+$

v. $3n\log n+2n^2\in\Theta(n^2)$

vi. $(8+\log(n))\cdot(n-1)=8n+n\log(n)-\log(n)-8\in\Theta(n\cdot\log(n))$

5. (Big-Oh)

a. *Without using the Master Theorem,* give tight big-Oh upper bounds for the divide-and-conquer recurrence $T(1)=1$; $T(n)=T(\frac{n}{2})+g(n)$, for $n>1$, where

i. $g(n)=1$

ii. $g(n)=2n$

iii. $g(n)=n^2$

b. For each of the following functions, use the Master Theorem to determine the best upper bound complexity of $T(n)$.

i. $T(n)=9\cdot T(\frac{n}{3})+3n(n+1)$

ii. $T(n)=8\cdot T(\frac{n}{2})+8n(n+1)$

iii. $T(n)=8\cdot T(\frac{n}{2})+2n^2(n+1)$

iv. $T(n)=6\cdot T(\frac{n}{2})+n^3$

v. $T(n)=6\cdot T(\frac{n}{3})+n^2$

c. Analyse the complexity of the following recursive algorithm to test whether a number x occurs in an *unordered* list $L=[x_1,x_2,\ldots,x_n]$ of size n. Take the cost to be the number of list element comparison operations.

Search$(x, L = [x_1, x_2, \dots, x_n])$:
  if $x_1 = x$ **then return** yes
  else if $n > 1$ **then return** *Search*$(x, [x_2, \dots, x_n])$
  else return no

d. Analyse the complexity of the following recursive algorithm to test whether a number x occurs in an *ordered* list $L = [x_1, x_2, \dots, x_n]$ of size n. Take the cost to be the number of list element comparison operations.

BinarySearch$(x, L = [x_1, x_2, \dots, x_n])$:
  if $n = 0$ **then return** no
  else if $x_{\lceil \frac{n}{2} \rceil} > x$ **then return** *BinarySearch*$(x, [x_1, \dots, x_{\lceil \frac{n}{2} \rceil - 1}])$
  else if $x_{\lceil \frac{n}{2} \rceil} < x$ **then return** *BinarySearch*$(x, [x_{\lceil \frac{n}{2} \rceil + 1}, \dots, x_n])$
  else return yes

[hide answer]

a. Unrolling the recursive definitions:

i. $T(n) = T(\frac{n}{2}) + 1 = T(\frac{n}{4}) + 2 = T(\frac{n}{8}) + 3 = \ldots = T(\frac{n}{n}) + \log_2 n = \mathcal{O}(\log n)$

ii. $T(n) = 2n + \frac{2n}{2} + \frac{2n}{4} + \frac{2n}{8} + \ldots = 2n \cdot (1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots) = \mathcal{O}(4n) = \mathcal{O}(n)$

iii. $T(n) = n^2 + (\frac{n}{2})^2 + (\frac{n}{4})^2 + (\frac{n}{16})^2 + \ldots = n^2 \cdot (1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{32} + \ldots) = \mathcal{O}(\frac{4}{3}n^2) = \mathcal{O}(n^2)$

b. The Master Theorem requires to determine the parameters d, α and β and compare α and β:

i. $d = 3$, $\alpha = 2$ and $\beta = 2$. From $\alpha = \beta$ it follows that the solution is $\mathcal{O}(n^2 \cdot \log(n))$.

ii. $d = 2$, $\alpha = 3$ and $\beta = 2$. From $\alpha > \beta$ it follows that the solution is $\mathcal{O}(n^3)$.

iii. $d = 2$, $\alpha = 3$ and $\beta = 3$. From $\alpha = \beta$ it follows that the solution is $\mathcal{O}(n^3 \cdot \log(n))$.

iv. $d = 2$, $\alpha = 2.585$ and $\beta = 3$. From $\alpha < \beta$ it follows that the solution is $\mathcal{O}(n^3)$.

v. $d = 3$, $\alpha = 1.631$ and $\beta = 2$. From $\alpha < \beta$ it follows that the solution is $\mathcal{O}(n^2)$.

c. The worst case is when the element occurs last in the list (or not at all). Let $T(n)$ be the total cost of running *Search* $(x, [x_1, \dots, x_n])$ in this case.

- if $x_1 = x$ then return yes   cost = 1 (one list element comparison)
- else if $n > 1$ then return *Search*$(x, [x_2, \dots, x_n])$ cost = $T(n-1)$ (recursive call with list size $n-1$)
- else return no   cost = 0

This can be described by the recurrence $T(1) = 1$; $T(n) = 1 + T(n-1)$ with the solution $T(n) = \mathcal{O}(n)$.

d. Again, the worst case is when the element occurs last in the list (or is larger than the last element). Let $T(n)$ be the total cost of running *BinarySearch*$(x, [x_1, \dots, x_n])$ in this case.

- if $n = 0$ then return no   cost = 0
- else if $x_{\lceil \frac{n}{2} \rceil} > x$ then return *BinarySearch*$(x, [x_1, \dots, x_{\lceil \frac{n}{2} \rceil - 1}])$ cost = 1 (one list element comparison; this condition is never satisfied in the assumed worst case that x is the largest element in the list)
- else if $x_{\lceil \frac{n}{2} \rceil} < x$ then return *BinarySearch*$(x, [x_{\lceil \frac{n}{2} \rceil + 1}, \dots, x_n])$ cost = $1 + T(\lfloor \frac{n}{2} \rfloor)$ (one comparison plus cost of recursive call with the second half of the list)
- else return yes   cost = 0

This can be described by the recurrence $T(0) = 0$; $T(n) = 2 + T\left(\frac{n}{2}\right)$ with the solution $T(n) = \mathcal{O}(\log n)$.

6. **Challenge Exercise**

Prove by induction that every connected graph $G = (V, E)$ must satisfy $e(G) \geq v(G) - 1$.

Hint: You can use the fact from a previous lecture that $\sum_{v \in V} deg(v) = 2 \cdot e(G)$.

[hide answer]

Base case: A graph with $v(G) = 1$ node is connected, has $e(G) = 0$ edges and hence satisfies $e(G) \geq v(G) - 1$.

Inductive step (proof by contradiction):

Consider graph G with $v(G) \geq 2$ nodes such that $e(G) < v(G) - 1$. We will show that G is not connected. From the lecture we know that $\sum_{v \in V} deg(v) = 2e(G) < 2v(G) - 2$ (according to the assumption). It follows that there is at least one vertex $v_0 \in V$ with $deg(v_0) \leq 1$, since otherwise $\sum_{v \in V} deg(v) \geq 2v(G)$.

- If $deg(v_0) = 0$ then G is not connected and we are done.
- If $deg(v_0) = 1$, consider the graph G' obtained by removing v_0 and its only connecting edge from G. It follows that $e(G') < v(G') - 1$ since $e(G') = e(G) - 1$, $v(G') = v(G) - 1$ and $e(G) < v(G) - 1$. By the induction hypothesis, G' cannot be connected, since $e(G') \not\geq v(G') - 1$. But then neither can be G: if v and w are vertices with no path between them in G' then adding v_0 doesn't help.

Assessment

After you have solved the exercises, go to COMP9020 20T1 Quiz Week 7 to answer 5 quiz questions on this week's problem set (Exercises 1-5 only) and lecture.

The quiz is worth 2.5 marks.

There is no time limit on the quiz once you have started it, but the deadline for submitting your quiz answers is **Thursday, 9 April 10:00:00am**.

Please continue to respect the **quiz rules**:

Do …

- use your own best judgement to understand & solve a question
- discuss quizzes on the forum only **after** the deadline on Thursday

Do not …

- post specific questions about the quiz **before** the Thursday deadline
- agonise too much about a question that you find too difficult