CS577: Introduction to Algorithms (Summer 2023)

Discussion 1: Discrete Review, Asymptotic Analysis and Graphs

Recap

Solving Recurrences

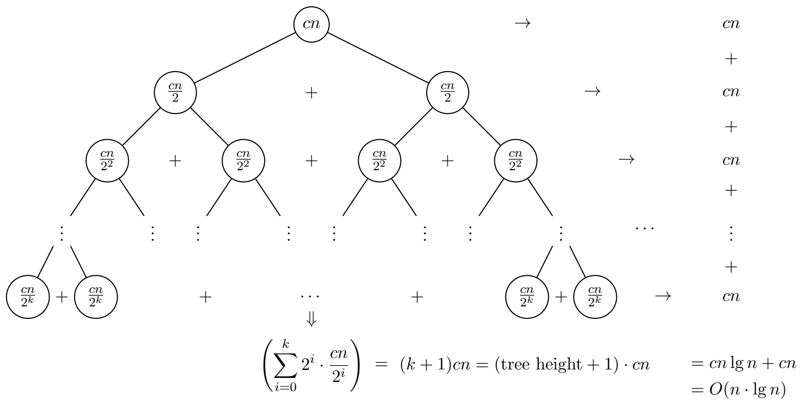
$$T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + cn; T(1) \leq c$$

Unrolling / unwinding:

$$\begin{aligned}
T(n) &\leq 2T\left(\frac{n}{2}\right) + cn \\
&\leq 2\left(2T\left(\frac{n}{4}\right) + c\frac{n}{2}\right) + cn \\
&\leq 2\left(2\left(2T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}\right) + c\frac{n}{2}\right) + cn \\
&\vdots \\
&\leq 2^k T\left(\frac{n}{2^k}\right) + kcn \\
&= nT(1) + cn\log(n), \text{ using (1)} \\
&= cn + cn\log n \\
&= O(n\log(n))
\end{aligned}$$

$$\begin{aligned}
& 1 = \frac{n}{2^k} \\
\iff & 2^k = n \\
\iff & k = \log_2(n) \qquad (1)
\end{aligned}$$

Recursion Tree:



cn
$\frac{cn}{2}$ + $\frac{cn}{2}$
$\frac{cn}{2^2}$ + $\frac{cn}{2^2}$ + $\frac{cn}{2^2}$ + $\frac{cn}{2^2}$
$\frac{cn}{2^k}$ + $\frac{cn}{2^k}$ + … + $\frac{cn}{2^k}$ + $\frac{cn}{2^k}$
$\to$ cn + cn + cn + $\vdots$ + cn
$\Downarrow$
$\left(\sum_{i=0}^{k} 2^i \cdot \frac{cn}{2^i}\right) = (k+1)cn = (\text{tree height} + 1) \cdot cn$
$= cn\lg n + cn$
$= O(n \cdot \lg n)$

Induction

We want to show that a predicate $P(n)$ holds for $n = \{0, 1, 2, \ldots\}$.

1. Base Case(s): The starting point of the induction. Without a base case, there is not induction.
2. Inductive Hypothesis: We assume that $P(k)$ is true for some k. For *strong* induction, we assume that $P(k)$ is true for all $j \leq k$.
3. Inductive Step: We show that $P(k+1)$ holds, using the (2) inductive hypothesis.

So,

$$\text{Base Case(s)} \rightarrow P(0) \rightarrow P(1) \rightarrow P(2) \rightarrow P(3) \rightarrow P(4) \rightarrow \ldots$$

Program Correctness

A program/algorithm is correct if it is:

- **Sound/partial correctness/correct** (any returned value is true), and
- **Complete/termination** (returns a value for all valid inputs).

Proving correctness requires at least 2 proofs: One for soundness and one for completeness.

Iterative Code:

1. Identify and prove the loop invariants using induction.
2. Using the invariants, prove the soundness.
3. Prove the completeness: Show that the algorithm terminates for all input.

Recursive Code:

1. Prove the soundness via induction:
   (a) Show that the recursive base case is correct.
   (b) Assuming that the recursive calls return the correct value (ind hyp), show that the code returns the correct value (induction step).
2. Prove the completeness: Show that recursive calls make progress towards a base case.

Asymptotic Bounds

Bound	Informal Notion	Formal Definition	Limit Test
$f(n) \in O(g(n))$	'$f(n) \leq g(n)$'	$O(g(n)) = \{f(n) : \exists c, n_0 > 0 \mid 0 \leq f(n) \leq cg(n)\ \forall n \geq n_0\}$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq d$ for $d \in \mathbb{R}_{>0}$
$f(n) \in \Omega(g(n))$	'$f(n) \geq g(n)$'	$\Omega(g(n)) = \{f(n) : \exists c, n_0 > 0 \mid 0 \leq cg(n) \leq f(n)\ \forall n \geq n_0\}$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} \geq c$ for $c \in \mathbb{R}_{>0}$
$f(n) \in \Theta(g(n))$	'$f(n) = g(n)$'	$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 > 0 \mid 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\ \forall n \geq n_0\}$	$c \leq \lim_{n\to\infty} \frac{f(n)}{g(n)} \leq d$ for $c, d \in \mathbb{R}_{>0}$
$f(n) \in o(g(n))$	'$f(n) \ll g(n)$'	$o(g(n)) = \{f(n) : \forall c > 0 \exists n_0 > 0 \mid 0 \leq f(n) < cg(n)\ \forall n \geq n_0\}$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
$f(n) \in \omega(g(n)$	'$f(n) \gg g(n)$'	$\omega(g(n)) = \{f(n) : \forall c > 0 \exists n_0 > 0 \mid 0 \leq cg(n) < f(n)\ \forall n \geq n_0\}$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Graphs

A graph $G = (V, E)$ is a way of encoding pairwise relationship among a set of objects:

- V is a collection of *nodes/vertices*
- E is a collection of *edges.*
  - In an undirected graph edges are a set of two nodes, $e = \{u, v\}$ for $u, v \in V$.
  - In a directed graph, edges are ordered pairs of nodes, $e = (u, v)$ for $u, v \in V$.
  - The notation (u, v) is often used for an edge in an undirected graph often when there is no ambiguity.
- A *path* in an undirected graph is a sequence P of nodes $v_1, v_2, \ldots, v_k$ such that $\{v_i, v_{i+1}\} \in E$ (or $(v_i, v_{i+1}) \in E$ for a directed graph). It is also called $v_1 - v_k$ path for the starting vertex v_1 and the ending vertex v_k.
- A *cycle* is a path $v_1, v_2, \ldots, v_k$ such that $k > 2$, the first $k - 1$ nodes are all distinct, and $v_1 = v_k$.
- An undirected graph is *connected* if there is a path between every pair of vertices.
- The *distance* between two nodes u and v is the minimum number of edges in a $u - v$ path.
- A *tree* is an undirected, connected, and acyclic (contians no cycles) graph.

Problems

1. Solve the following recurrences:
   (a) $T(n) = T(n-1) + T(n-2) + 5$, $T(1) = 1$, and $T(0) = 0$. Use unrolling.
   (b) $T(n) = 3T(n/4) + n^2$ and $T(1) = 1$. Use a recursion tree.
2. Prove the following statements using induction:
   (a) For all $n \in \mathbb{N}$,
   $$1^3 + 2^3 + \cdots + n^3 = \frac{1}{4}n^2(n+1)^2$$
   (b) For all $n \in \mathbb{N}$,
   $$\frac{1}{1\cdot 2} + \frac{1}{2\cdot 3} + \cdots + \frac{1}{n\cdot(n+1)} < 1$$
   (Hint: This is not possible if your inductive hypothesis is $1/(1\cdot 2)+\cdots+1/(k\cdot(k+1)) < 1$. Instead, try to prove an equality of the form $1/(1\cdot 2)+\cdots+1/(n\cdot(n+1)) = f(n)$, where $f(n) < 1$ for all n. Try small values of n to find what $f(n)$ should be.)
3. Prove that the following two programs are correct.

Algorithm 1 `BINARY-SEARCH(A, x, left, right)`

```
if left ≤ right then
    mid = (left + right)/2
    if A[mid] < x then
        return BINARY-SEARCH(A, x, mid+1, right)
    else if A[mid] > x then
        return BINARY-SEARCH(A, x, left, mid-1)
    else
        return True
    end if
else
    return False
end if
```

Algorithm 2 `FIND-MAX(A)`

```
ans = A[0]
n = len(A)
for j = 1...n-1 do
    ans = max(ans, A[j])
end for
return ans
```

4. Arrange the following functions in ascending order of asymptotic growth rate. Assume the base of the logarithm is 2.

   - $\sqrt{5^{\log n}}$
   - $\log(5^n)$
   - 5^n
   - $\sqrt{n}$
   - $n^{\log n}$
   - 3^{n+10}
   - $\log n$
   - $(\log n)^n$
   - $5^{\sqrt{\log n}}$

5. *Kleinberg, Jon. Algorithm Design (p. 110, q. 9).* There's a natural intuition that two nodes that are far apart in a communication network — separated by many hops — have a more tenuous connection than two nodes that are close together. There are a number of algorithmic results that are based to some extent on different ways of making this notion precise. Here's one that involves the susceptibility of paths to the deletion of nodes.

   Suppose that an n-node (connected) undirected graph $G = (V, E)$ contains two nodes s and t such that the distance between s and t is strictly greater than $\frac{n}{2}$.

   (a) Show that there must exist some node v, not equal to either s or t, such that deleting v from G destroys all $s - t$ paths. (In other words, the graph obtained from G by deleting v contains no path from s to t.)

   (b) Give an algorithm with running time $O(m + n)$ to find such a node v.