Examples = Blue
Title of section = Black
Conclusion = Orange

Mathematical Induction

• Used to prove a sequence of statements (S(1), S(2), ... S(n)) indexed by positive integers:
– Basis step: prove that the statement is true for n = 1 (or higher)
– Inductive step: assume that S(1), S(2), ..., S(n-1) is true and prove that S(n) is true for all n > 1
• Key to proving math induction is to find case S(i) "within" case S(n) where 1 ≤ i ≤ n-1
-Mathematical induction is a powerful proof technique used in mathematics, computer science, and other disciplines to prove a sequence of statements. The Introduction to Algorithms book by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein introduces mathematical induction as a method to prove a sequence of statements indexed by positive integers.
-To use mathematical induction to prove a sequence of statements (S(1), S(2), ... S(n)), two steps are required: the basis step and the inductive step. The basis step involves proving that the statement is true for the first integer in the sequence, often S(1). The inductive step involves assuming that the statement is true for some integer k (1 ≤ k ≤ n-1) and using this assumption to prove that the statement is also true for the next integer, k+1.
-The key to a successful mathematical induction proof is finding the "inner" case S(i) within the larger case S(n), where 1 ≤ i ≤ n-1. This step is critical in showing that the statement is true for all integers in the sequence. Once the basis step and inductive step have been established, it can be concluded that the statement is true for all positive integers.

Example:
Prove that the sum of the first n positive integers is n(n+1)/2, using mathematical induction.
We want to prove that S(n) = 1+2+...+n = n(n+1)/2, for all positive integers n.
Basis step: When n = 1, S(1) = 1, which is equal to 1(1+1)/2 = 1. Hence the statement is true for n = 1.
Inductive step: Assume that S(1), S(2), ..., S(n-1) is true, i.e., 1+2+...+(n-1) = (n-1)n/2. Now we need to prove that S(n) is true, i.e., 1+2+...+n = n(n+1)/2.
1+2+...+n-1+n = (n-1)n/2 + n
= (n^2 - n)/2 + (2n)/2
= n(n+1)/2
Thus, S(n) is true for all positive integers n > 1.
Therefore, by mathematical induction, we have proven that 1+2+...+n = n(n+1)/2 for all positive integers n.

Asymptotic Notation

• "big" O definition (asymptotic "≤", or an upper bound exists for f(n)):
  • O(g(n)) = { f(n) : ∃ constants c > 0, n0 > 0 ∋ 0 ≤ f(n) ≤ cg(n) ∀ n ≥ n0 }

- $f(n) = O(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or below $c \cdot g(n)$. Note: must find n_0 and c.

Example:
Suppose these functions:
- $f(n) = 3n^2 + 5n + 7$
- $g(n) = n^2$

We want to show that $f(n) = O(g(n))$, i.e., that there exist positive constants c and n_0 such that for all $n \geq n_0$, $0 \leq f(n) \leq cg(n)$.
To do this, we need to find suitable values for c and n_0. First, we need to simplify the expression $0 \leq f(n) \leq cg(n)$:
$0 \leq 3n^2 + 5n + 7 \leq cg(n)$
We can simplify the left and right-hand sides of this inequality:
$0 \leq 3n^2 + 5n + 7 \leq cn^2$
Now we need to choose values for c and n_0 that satisfy the above inequality for all $n \geq n_0$. We can do this by selecting c to be a value greater than or equal to 3 (since $3n^2$ is the dominant term in $f(n)$), and selecting n_0 to be 1. Then, we can verify that the inequality holds for all $n \geq n_0$:
$0 \leq 3n^2 + 5n + 7 \leq cn^2$
$0 \leq 3(1)^2 + 5(1) + 7 \leq c(1)^2$ (since $n_0 = 1$)
$0 \leq 15 \leq c$
Therefore, we can choose $c = 15$ and $n_0 = 1$ to show that $f(n) = O(g(n))$. This means that $f(n)$ grows no faster than $g(n)$ as n approaches infinity, or equivalently, $g(n)$ is an upper bound on $f(n)$.

- “big” definition (asymptotic “≥”, or a lower bound exists for f(n)):
  - $\Omega(g(n)) = \{ f(n) : \exists \text{ constants } c > 0, n_0 > 0 \ni 0 \leq cg(n) \leq f(n) \ \forall n \geq n_0 \}$
  - $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or above $c \cdot g(n)$. Note: must find n_0 and c

Example:
Suppose these functions:
$f(n) = 4n^2 - 2n + 1$
$g(n) = n^2$
We want to show that $f(n) = \Omega(g(n))$, i.e., that there exist positive constants c and n_0 such that for all $n \geq n_0$, $0 \leq cg(n) \leq f(n)$.
To do this, we need to find suitable values for c and n_0. First, we need to simplify the expression $0 \leq cg(n) \leq f(n)$:
$0 \leq cg(n) \leq 4n^2 - 2n + 1$
We can simplify the left and right-hand sides of this inequality:
$0 \leq cn^2 \leq 4n^2 - 2n + 1$
Now we need to choose values for c and n_0 that satisfy the above inequality for all $n \geq n_0$. We can do this by selecting c to be a value less than or equal to 4 (since $4n^2$ is the dominant term in $f(n)$), and selecting n_0 to be 1. Then, we can verify that the inequality holds for all $n \geq n_0$:

$0 \leq cn^2 \leq 4n^2 - 2n + 1$
$0 \leq c(1)^2 \leq 4(1)^2 - 2(1) + 1$ (since $n_0 = 1$)
$0 \leq c \leq 3$
Therefore, we can choose $c = 3$ and $n_0 = 1$ to show that $f(n) = \Omega(g(n))$. This means that f(n) grows no slower than g(n) as n approaches infinity, or equivalently, g(n) is a lower bound on f(n).

- definition (asymptotic "=", or a tight bound exists for f(n)):
    - $(g(n)) = \{ f(n) : \exists$ constants $c1 > 0, c2 > 0, n0 > 0 \ni 0 \leq c1g(n) \leq f(n) \leq c2g(n) \ \forall\ n \geq n0 \}$
    - $f(n) = \Theta(g(n))$ if there exist positive constants n0, c1, and c2 such that to the right of n0, the value of f (n) always lies between $c1 \cdot g(n)$ and $c2 \cdot g(n)$ inclusive. Note: must find n0 , c1 and c2

Example:
Suppose these functions $f(n) = 3n^2 + 2n$ and $g(n) = n^2$. We want to show that f(n) is asymptotically equal to g(n), or $f(n) = \Theta(g(n))$.
To do this, we need to find constants c1, c2, and n0 such that for all $n \geq n0$, $c1 \cdot g(n) \leq f(n) \leq c2 \cdot g(n)$.
First, we'll show that f(n) is bounded above by g(n) using the big O notation. We want to find constants c and n0 such that for all $n \geq n0$, $f(n) \leq c \cdot g(n)$.
Let's start with the inequality $f(n) \leq c \cdot g(n)$. We can rewrite this as:
$3n^2 + 2n \leq c \cdot n^2$
Dividing both sides by n^2 (since n is positive for all $n \geq n0$), we get:
$3 + 2/n \leq c$
Now, let's choose $c = 4$ and $n0 = 1$. For all $n \geq n0 = 1$, we have:
$3 + 2/n \leq 4$
Therefore, for all $n \geq 1$, we have:
$3n^2 + 2n \leq 4n^2$
This shows that f(n) is bounded above by g(n) for all $n \geq 1$. So we have shown that $f(n) = O(g(n))$.
Next, we'll show that f(n) is bounded below by g(n) using the big Omega notation. We want to find constants c and n0 such that for all $n \geq n0$, $c \cdot g(n) \leq f(n)$.
Let's start with the inequality $c \cdot g(n) \leq f(n)$. We can rewrite this as:
$c \cdot n^2 \leq 3n^2 + 2n$
Dividing both sides by n^2 (since n is positive for all $n \geq n0$), we get:
$c \leq 3 + 2/n$
Now, let's choose $c = 2$ and $n0 = 1$. For all $n \geq n0 = 1$, we have:
$2 \leq 3 + 2/n$
Therefore, for all $n \geq 1$, we have:
$2n^2 \leq 3n^2 + 2n$
This shows that f(n) is bounded below by g(n) for all $n \geq 1$. So we have shown that $f(n) = \Omega(g(n))$.
Finally, since we have shown that $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, we can conclude that $f(n) = \Theta(g(n))$. In other words, we have found constants $c1 = 2$ and $c2 = 4$ and $n0 = 1$ such that for all $n \geq n0$, $2 \cdot g(n) \leq f(n) \leq 4 \cdot g(n)$.

Asymptotic Notation

• Theorem:

For any two functions f(n) and g(n) we have f(n) = (g(n)) if and only if f(n) = O(g(n)) and f(n) = (g(n).

• Often use f(n) = O(g(n)) and f(n) = (g(n) to show f(n) = (g(n))

Example:
Suppose we have two functions, f(n) = 2n^2 + 3n + 1 and g(n) = n^2. We want to determine whether f(n) = (g(n)), i.e., whether f(n) grows at the same rate or slower than g(n) as n approaches infinity.
To use the theorem mentioned, we need to show that f(n) = O(g(n)) and f(n) = (g(n)).
First, we will show that f(n) = O(g(n)). To do this, we need to find constants c and n0 such that for all n ≥ n0, f(n) ≤ c*g(n).
We can write:
f(n) = 2n^2 + 3n + 1
≤ 2n^2 + 3n^2 + n^2 (since 3n + 1 ≤ 3n^2 for n ≥ 1)
= 6n^2
= 6g(n)
Thus, we can choose c = 6 and n0 = 1 to show that f(n) = O(g(n)).
Next, we will show that f(n) = (g(n)). To do this, we need to find constants c and n0 such that for all n ≥ n0, f(n) ≥ c*g(n).
We can write:
f(n) = 2n^2 + 3n + 1
≥ 2n^2
= 2g(n)
Thus, we can choose c = 2 and n0 = 1 to show that f(n) = (g(n)).
Therefore, by the theorem, we can conclude that f(n) = (g(n)), since f(n) = O(g(n)) and f(n) = (g(n)).
In this case, we often use f(n) = O(g(n)) and f(n) = (g(n)) to show f(n) = (g(n)), which implies that f(n) grows at the same rate or slower than g(n) as n approaches infinity.

Substitution method

• **Background**: The substitution method is a technique used to solve recurrence relations.
• Guess a solution
  – T(n) = O(g(n))
  – Induction goal: apply the definition of the asymptotic notation
• T(n) ≤ d g(n), for some d > 0 and n ≥ n0
  – Induction hypothesis: T(k) ≤ d g(k) for any k < n
• Prove the induction goal
  – Use the induction hypothesis to find some values of the constants d and n0 for which the induction goal holds

Example of the recurrence relation T(n) = 2T(n/2) + n:

1. Guess the solution
We guess that the solution to this recurrence relation is $T(n) = O(n \log n)$.
2. Prove the guess by induction
We will use mathematical induction to prove that $T(n) \leq cn \log n$ for some constant $c > 0$.
Base case: $T(1) = 1 \leq c \log 1 = 0$. This is true for all $c \geq 1$.
Inductive hypothesis: Assume that $T(k) \leq ck \log k$ for all $k < n$.
Inductive step: We need to prove that $T(n) \leq cn \log n$.
$T(n) = 2T(n/2) + n$
$\leq 2c(n/2) \log(n/2) + n$ (by the inductive hypothesis)
$= cn \log n - cn + n$
$\leq cn \log n$ (if we choose $c \geq 1$)
Therefore, by mathematical induction, $T(n) \leq cn \log n$ for all $n \geq 1$ and $c \geq 1$. This proves our guess that $T(n) = O(n \log n)$.
3. Verify the solution
We can verify our solution by substituting it back into the original recurrence relation:
$T(n) \leq 2T(n/2) + n$
$\leq 2c(n/2 \log(n/2)) + n$
$= cn \log n - cn + n$
$= O(n \log n)$
Therefore, our solution $T(n) = O(n \log n)$ is correct.

Example 2:
Sure, here is an example of using the substitution method to prove that $T(n) = 2T(n/2) + n\log n$ is $O(n\log n)$:
First, we guess that $T(n) = O(n\log n)$. This means we assume that there exist constants d and n0 such that $T(n) \leq d * n\log n$ for all $n \geq n0$.
Next, we will use mathematical induction to prove that the assumption is true.
Basis step: We need to show that the assumption is true for some base case. Let's choose n = 2. Then, $T(2) = 2T(1) + 2\log 2 = 4$. We can see that $T(2) \leq d * 2\log 2$ if we choose $d = 2$, so the assumption holds for $n = 2$.
Inductive step: We assume that $T(k) \leq d * k\log k$ for all $k < n$. Now, we need to show that $T(n) \leq d * n\log n$ for some $d > 0$ and $n \geq n0$.
Using the recursive definition of $T(n)$, we can substitute the induction hypothesis into the equation:
$T(n) = 2T(n/2) + n\log n$
$\leq 2d(n/2)\log(n/2) + n\log n$ // using the induction hypothesis
$= dn\log(n/2) + n\log n$
$= dn\log n - dn\log 2 + n\log n$
$= dn\log n - dn + n\log n$
$= (d+1)n\log n - dn$
Since $n\log n >= n$ for all $n >= 1$, we have $T(n) <= (d+1) * n\log n$ for sufficiently large n if we choose $d >= 1$. Therefore, the assumption holds for $n = k+1$. By induction, we have shown that

T(n) = O(nlogn), which means that there exist constants d and n0 such that T(n) ≤ d * nlogn for all n ≥ n0.

Master method

• "Cookbook" for solving recurrences of the form: where, a ≥ 1, b > 1, and f(n) > 0

Case 1: if f(n) = O(n(logba) -) for some > 0, then: T(n) = (nlogba)

Example of the Master Method, let's consider the recurrence relation:

T(n) = 4T(n/2) + n

Here, a = 4, b = 2, and f(n) = n. Therefore, we have:

logba = log24 = 2

f(n) = n = Ω(n1)

Since f(n) is not in the form of O(n(logba)-), we cannot apply Case 1 of the Master Method.

Case 2: if f(n) = (nlogba), then: T(n) = (nlogba)log n

Case 3: if f(n) = Ω(n(logba)+), and if af(n/b) ≤ cf(n) for some constant c < 1 and all sufficiently large n, then: T(n) = (f(n))

Since f(n) = n is not in the form of (n(logba)+), we cannot apply Case 3 of the Master Method.

Therefore, we need to apply Case 2. Here, f(n) = n = (nlog24). Therefore, T(n) = (nlogba)log n = (nlog24)log n.

Hence, the solution to the recurrence relation T(n) = 4T(n/2) + n is T(n) = (nlog24)log n.

Case 2: if f(n) = (nlogba), then: T(n) = (nlogba lg n)

Example: T(n) = 4T(n/2) + nlog2 3

Here, a = 4, b = 2, and f(n) = nlog2 3.

We need to check if f(n) falls into Case 2 of the Master method:

f(n) = nlog2 3 = (nlog24)

logba = log24 = 2

Since f(n) = (nlog24), which is the same as nlogba, we apply Case 2 of the Master method:

T(n) = (nlogba lg n) = (nlog24 lg n)

Therefore, the solution to the recurrence is T(n) = (nlog24 lg n).

Case 3: if f(n) = (n(logba) +) for some > 0, and if af(n/b) ≤ cf(n) for some c < 1 and all sufficiently large n, then: T(n) = (f(n)) regularity condition

Example:

Suppose we have a recurrence relation given by T(n) = 3T(n/2) + n^(1/2)logn. Here, a = 3 and b = 2. Thus, logba = log32 ≈ 1.585. Now, we need to compare f(n) = n^(1/2)logn with n^(logba ±). We can see that f(n) = (n^(1/2))(logn) is in between n^(logba -) and n^(logba +). So, we use case 2 of the master theorem.

Now, f(n) = n^(1/2)logn, and we have a = 3 and b = 2. So, we calculate af(n/b) as follows:

af(n/b) = 3[(n/2)^(1/2)log(n/2)]
        = 3[(n^(1/2))(logn - log2)]
        = 3/2[n^(1/2)logn - n^(1/2)]

Now, we need to compare af(n/b) with cf(n) for some constant c < 1. We can choose c = 3/4. Then,

af(n/b) = 3/2[n^(1/2)logn - n^(1/2)]

≤ 3/2[n^(1/2)logn - n^(1/2)/2]
= 3/2[n^(1/2)logn - n^(1/2)log(n^(1/2))]
= 3/2(n^(1/2))(logn/2)
= (3/2)(n^(1/2))(logn - 1)
≤ (3/4)(n^(1/2))(logn)

Thus, the regularity condition is satisfied. Hence, we use the formula for case 3 of the master theorem, which gives us T(n) = (n^(1/2))(logn).

Asymptotic running times, how it works (visually), and when to use it.

- 1. Insertion Sort
  - The asymptotic running time of an algorithm provides a way to analyze how the algorithm's performance scales with input size. It is expressed using big O notation, which gives an upper bound on the running time. In the case of the Insertion Sort algorithm, its worst-case running time is O(n^2), which means that the running time will not exceed a quadratic function of the input size. This is because for larger input sizes, the number of comparisons and swaps needed to sort the array grows exponentially. Knowing the asymptotic running time of an algorithm is important because it helps in making informed decisions about which algorithm to use for a given problem, especially for larger input sizes.
- 2. Breadth-First Search
  -
- 3. Depth First Search
- 4. Topological Sort
- 5. Minimum Spanning Trees (Kruskal and Prim algorithms)
- 6. Bellman-Ford
- 7. Single source shortest path in DAGs
- 8. Dijkstra’s algorithm
- 9. Edit distance
- 10. 0/1 Knapsack with unlimited quantities
- 11. Horn Formulas
- 12. Huffman codes