**1. Intro**-

(1) Why Study? a. Important for other branches b. driver for innovation c. lens on other science d. be better programmer e. sharpen analytical skills f. think algorithmic

(2) Program Workflow: 5 Stages- 1. Application Stage 2. Design Stage 3. Algorithm Analysis Stage 4. Implementation Stage 5. Experiment Stage.

**2. Algorithm Analysis-**

(1) Quest for Efficiency: a. Computational complexity: not based on speed of computers, but good algorithms and organization b. efficient program design c. Four Principles: 1. Each structure and algorithm has benefits and costs. 2. Tradeoffs 3. Should know enough about common practice 4. Data structure follow application needs d. Measuring performance

(2) Experimental studies: 1. Programming workflow. #1 is inefficient. #2 predicts resources (**time,** space) needed. Our focus is capability of the machine.

(3) Growth Rate Analysis:

- Primitive Operations: Assigning value, following object reference, performing arithmetic, comparing, accessing element, calling method, returning.

**- Speed from slowest to fastest:** Exponential, Cubic, Quadratic, NlogN, Linear, Log, Constant.

- The dominant term is the one that makes the program the slowest.

- Tips to do when analyzing: Go line by line, calculate cost and how many times it is repeated and comment on what it does. Reminder: loops are n+1 (for failure).

(4) Asymptotic analysis: **Big-O notation**- how fast does function grow in relation to input size. Upper bound, the worst case runtime. Max time.

**Properties:** speed from slow to fast: n!, 2n, n3, n2, nlogn, n, logn, 1.

To find constant, divide the two functions and its everything bigger than intersection.

**Big-Omega:** lower bound of an algorithm. Best case runtime

**Big-Theta:** tight bound, average case, between omega and o.

**3. Algorithm Design Patterns:** Recursive calls

**4. Linear Data Structures:** arrays (Static data), singly linked lists(Dynamic data). Static size has to be set and cant be changed, dynamic is free to change.

**5. Sorting Algorithms**:

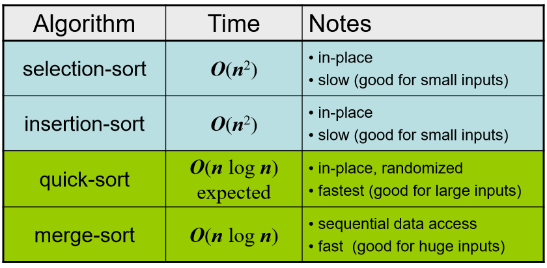
Selection Sort: finds the largest value, puts it in the last element, finds the next largest and puts it in the one before the last and so on. N2 for worst and average scenarios, uses n2/2 comparisons uses n exchanges.

Bubble Sort: bubbles up every two elements and swaps them to be in correct order. Repeats until the array is in order. (remember to have a flag to see if its already sorted). N2 for worst and average scenarios, uses n2/2 comparisons uses n2/2 exchanges.

Insertion Sort: Starts from left and moves the data over if a value on the right side is greater than the on the left. N2 for worst and average scenarios, uses n2/2 comparisons uses n2/2 exchanges for worst. N comparisons and 0 moves for best. uses n2/4 comparisons uses n2/4 exchanges for average

Merge Sort: halves the array into 2 repeatedly until it is left with one element then sorts it one by one. Recursive call. Nlogn complexity.

Quicksort: picks a partition then moves the elements accordingly. Divide and conquer. Recursive. Best case nlogn. Worst case n2(pivot is the smallest number in array)

SUMMARY:

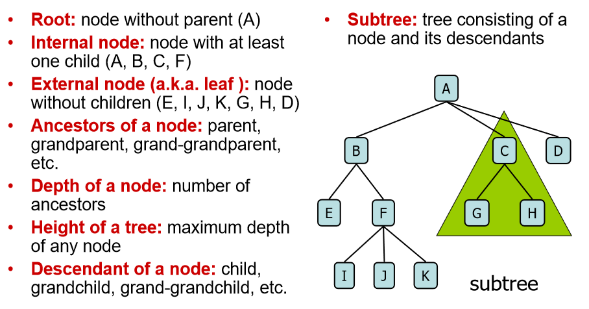
**6. Searching algorithms:**

Sequential: Best case 1, worst case n. average n. simply go one by one until it is found.

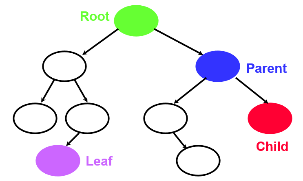
Binary: best case 1, worse case logn. Major assumption, array is sorted.

Take first, mid and last element. Check if what you are looking for is in first have or second half. Recursively call using either the first or second half repeat until found.

**7. Stacks**: First in, Last Out. Last in, First Out. Push inserts, pop removes. Use link stack rather than arrayed stack.



**8. Trees:**

Leaf: no children.

Path: number of edges to get from one node to another.

Root is at depth 0.

Siblings: (B,C,D)  
Height of node is from node to node, height of tree is the deepest path from root to node.