1. Definition ... a finite set of unambiguous, executable instructions that will ultimately terminate if followed.
   - Imperative languages

2. Algorithmic Discovery
   - Top Down, Stepwise refinement ...
     - do the “clear beepers out of corners of the room” problem
   - Bottom Up ... same problem ...
   - Modular Design ...
     - DEFINE-NEW-INSTRUCTION turnright AS ITERATE 3 TIMES turnleft;
   - Pre and Post Conditions
     - At beginning of cycle, robot is along wall facing next wall
     - At the end of cycle the beeper is cleared, robot has turned the corner to face the next wall

3. Algorithmic Representation
   - Pseudo-Code (go-to-the-wall)
   - Flow Chart
   - Mathematical Statement \[ n_0 = 1; n_i = i \cdot n_{i-1} \ i = 1..50 \]
   - Syntactically correct instructions (WHILE front-is-clear DO move;)
   - Depends on elements of the language
4. Elements of the Language
   - Primitives (move, turnleft, pickbeeper, putbeeper, turnoff)
   - Flow Control
     - Sequence
     - Branching (IF-THEN-ELSE)
       → Testing ... front-is-clear, facing-north, any-beepers-in-beeper-bag, etc
     - Loops (ITERATE n TIMES, WHILE-DO)
   - Block Structuring BEGIN .. END
   - Modularization ... functions, procedures (DEFINE-NEW-INSTRUCTION)

5. Examples ... will be done on blackboard
   - Sorting
     - Bubble sort
     - Selection sort
     - Insertion sort
     - Insertion sort with pointers in graphical form
     - Insertion sort with pointers in table form
   - Searching
     - Linear
   - Binary
Sort Algorithms in pseudo-code

Bubble (list)
   Iterate N-1 times
      For each item in the list, from the first to next-to-last
         If item > next-item-in-list, swap the items

Selection(list)
   Iterate N-1 times
      Select next item in list (if first time, select first)
      Mark item as smallest-item
      For each item in the list, from selected-item-plus-one to last item
         If item < smallest-item, mark item as smallest-item
      Swap selected-item with smallest-item

Insertion(list)
   Take first item from list and insert in newlist
   For each remaining item in the list
      Insert the item in newlist in proper location
   Transfer newlist to list

Insert (item, list)
   // precondition ... assume list is sorted
   Do binary search for item in list
   Put item in that location
6. **Search Algorithms in pseudo code**

   Linear-search(list, target)
   
   For each item in the list
   
   If item == target
   
       then return location of item
   
   else return -1  // not-found

   Binary-search(list, target)  list is an indexed array
   
   If list > 1 item
   
   then
   
       divide list in half  // value of middle = mid
       if target == last item in first-half-of-list
       
           then return location of item in original list
       
       if target < last item in first-half-of-list
       
           then return mid + 1 + Binary-search (first-half, target)
       
       else return mid + Binary-search(last-half, target)
   
   else
   
       return location of item in original list

7. **(informal) Algorithm Analysis**

   - Linear Search ... \( \sim N/2 \) ... \( \bigO(N) \)
   - Binary Search ... \( \sim \log_2 N \) ... \( \bigO(\lg N) \)
   - Bubble and Selection Sorts ... \( \bigO(n^2) \)
   - Insertion Sort ... \( \bigO(n \lg(n)) \)
8. Recursion ... a procedure that calls itself
   - Binary Search (see above)
   - Factorial  \( n! \)
   - Fibonacci Numbers
     - \( F(0) = 1, \)
     - \( F(1) = 1, \)
     - \( F(i) = F(i-1) + F(i-2) \) for \( i \geq 2 \)

   - Towers of Hanoi
     \[
     \text{Hanoi}(n, x, y, z) \\
     \quad \text{if } n == 1 \\
     \quad \quad \text{then move } x \rightarrow y \\
     \quad \text{else} \\
     \quad \quad \text{Hanoi}(n - 1, x, z, y) \\
     \quad \quad \text{move } x \rightarrow y \\
     \quad \quad \text{Hanoi}(n - 1, z, y, x) 
     \]

   - Number of operations = \( 2^n - 1 \) \( \Theta(2^n) \) ... this is exponential growth!
9. Binary Search Tree
   - Placement of items into the tree
   - Traversal of the tree
     - Preorder, inorder, postorder traversals
     - List is sorted in an inorder traversal
     - A recursive process
       
       ```plaintext
       traverse (list)
       if left_child_exists  // not null, not a leaf node
       then traverse (left-child)
       print_data
       if right_child exists  // not null, not a leaf node
       then traverse(right-child)
       ```
10. Compilation process

Role of the Compiler

- C++ Program
- Data for Program

C++ Compiler

"The Computer"

Machine Language Program

"The Computer"

Computer

"The Computer"

Output of C++ program and its data file