

Design and Analysis of Algorithms

L04: Algorithm Specifications and Analysis

Dr. Ram P Rustagi
Sem IV (2019-H1)
Dept of CSE, KSIT
rprustagi@ksit.edu.in

Resources

- python turtle graphics
 - <https://docs.python.org/3/library/turtle.html>
- Textbook :
 - T2 (Horowitz, Sahani)
 - T1 (Levitin)

Observation about Programs

- Discussed following programs
 - Prime factors
 - Generate harmonic series
 - Integer display in binary
 - Fibonacci series generation
 - Ramanujan number computation
 - Analysis of programs
 - Count number of digits
 - Reverse the digits
 - Use of post and pre-increment operator

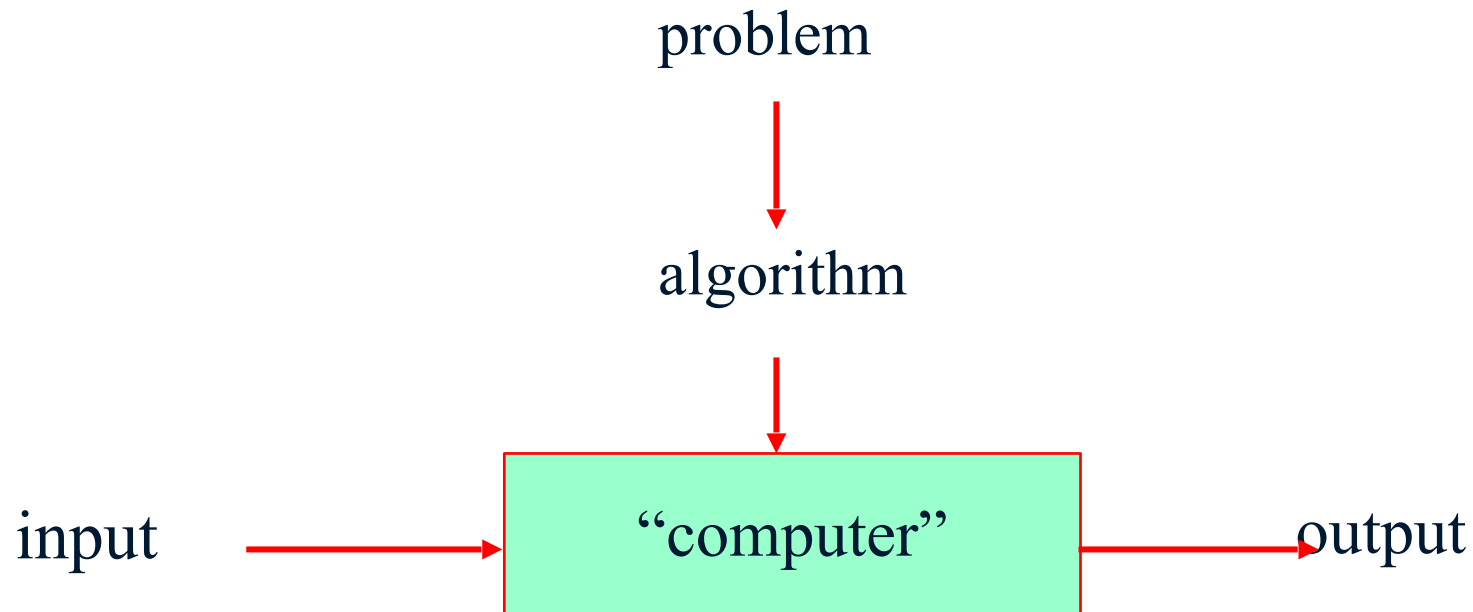
Observation about Programs

- All programs require a computer to run
 - Can we run these with pen and paper
- Input : hard coded or specified as argument
- Each program produces some output
- Program terminates after some time.
 - Does not run for ever.
 - Can we write a program to print all integers?
- Each instruction is unambiguous.
 - There is confusion on how it should be executed
- Each step is executable. It is feasible.
- These are attributes of algorithm

Algorithm

- Defn: A finite set of instructions that when executed (followed) accomplishes a particular task
 - An algorithm is a sequence of unambiguous instructions for solving a problem, i.e. for obtaining a required output for any legitimate input in a finite amount of time.
- Algorithm must satisfy following criteria
 - Input : zero or more
 - Output: At least one output is produced
 - Definite : unambiguity
 - Finite: termination
 - Effectiveness: Feasible to execute

Algorithm



Algorithm Criteria: Examples

- Example for illustrating criteria importance
 - Ambiguity:
 - Go to college (KSIT) or movie theatre
 - Withdraw 100 or 200 rupees from ATM/Bank
 - Infeasible
 - Divide by zero
 - Represent $1/3$ without loss of precision
 - Infinite:
 - Print all integers
 - What about “Count number of hairs on your head”
- Computational procedures: Algorithms that are definite and feasible.
- Program: expression of an algorithm in a prog. lang.

Differences with Data Structures

- All program work with (manipulate) data and requires some form of representation of data
- Data structure is concerned with representation and manipulation of data
- Data manipulation requires an algorithm
- **Study of Data Structures and Algorithm is fundamental to Computer Science**

Algorithms: Areas of Study

- How do **design** algorithms
 - The key focus of this course
 - What approaches to take: e.g. divide and conquer
- Algorithm validation
 - program validation
 - How to ensure it outputs correct value
- **Performance analysis:**
 - How much time it takes, how effective it is.
 - Best case, worst case, average case
 - Example: finding a seat for new student who enters
- How to test the programs
- Optimality: Is the solution optimal

Design Strategies

- Brute force
- Divide and conquer
- Decrease and conquer
- Greedy approach
- Dynamic programming
- Backtracking
- Branch and bound
- Space and time trade offs

Algorithm Specification

- How should an algorithm be specified?
- Can it be described in plain english or any other natural language?
 - Will it be unambiguous?
 - Does it other critieria of finiteness, feasibility etc.
- Should it be defined via a flow chart?
 - Does graphical representation covers all criteria?
 - Works well when it is short and simple/.
- Should we use a programming language? C/C++/Java?
 - What would be difference between program and algorithm?
- A mix and match of all of above?

Example: Algorithm Specification

- Selection sort:
 - Description in english:
 - Sort collection of n unsorted elements. Find the shortest element and place it ahead of sorted list.
 - Issues with this description
 - How are the elements initially stored?
 - In an array, linked list or something else?
 - How is sorted list maintained?
 - Meaning of ahead of sorted list?
 - How the result is placed?

Selection Sort: Specification

- Assumption:
 - Input: All unsorted elements are stored in an array
 - i^{th} element is stored in i^{th} position i.e. $a[i]$
 - Output: Sorted elements will be in the same array
- Specification

```
for i=1 to n do
    examine a[i] to a[n], and
    find the smallest element a[j]
    Interchange a[i] and a[j]
```
- Does it meet all criteria?
 - Input, output, unambiguity, feasibility, finiteness,

Selection Sort: Java Code

- // I/p: array $a[1:n]$ of n elements contains n integers.

```
for (int i=0; i<10; i++) {  
    int j=i;  
    for (int k=i+1; k<10; k++) {  
        if (a[k] < a[j])  
            j = k;  
    }  
    int x = a[i];  
    a[i] = a[j];  
    a[j] = x;  
}
```

- Does it work correctly?

Selection Sort: Example

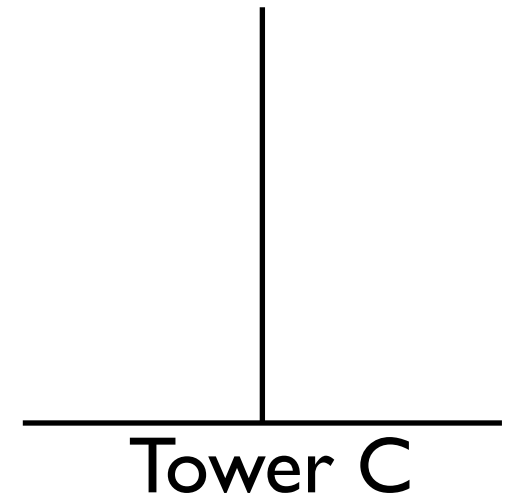
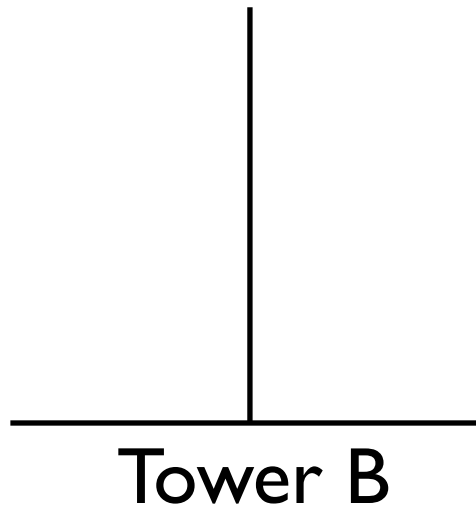
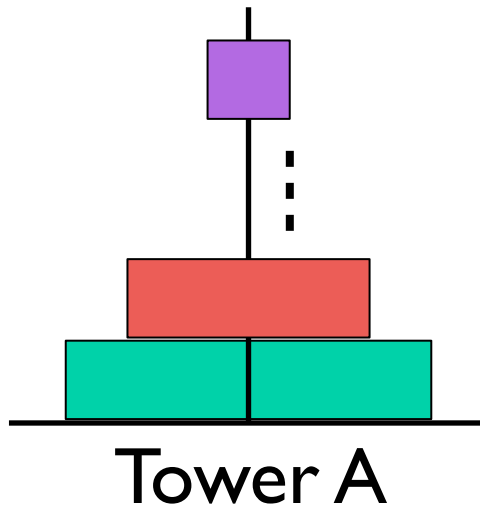
- python turtle graphics
 - Module: `sorting_animate.py`
 - Modified to define number of items to sort
 - Invocation:
 - `python selection_sort <n>`

Recursive Algorithms Specification

- Recursion:
 - Efficient mechanism to specify a good number of problems
 - Factorial: $n! = n * (n-1)!$
 - Binomial coefficient:
$${}^nC_k = n! / k! (n-k)! = {}^{n-1}C_k + {}^nC_{k-1}$$
- Recursion invocation
 - Direct recursion: by a function itself e.g. factorial
 - Indirect recursion: a function calls another function which turn calls the calling function.
- Works well for problem which can be described using recursion

Hanoi Tower

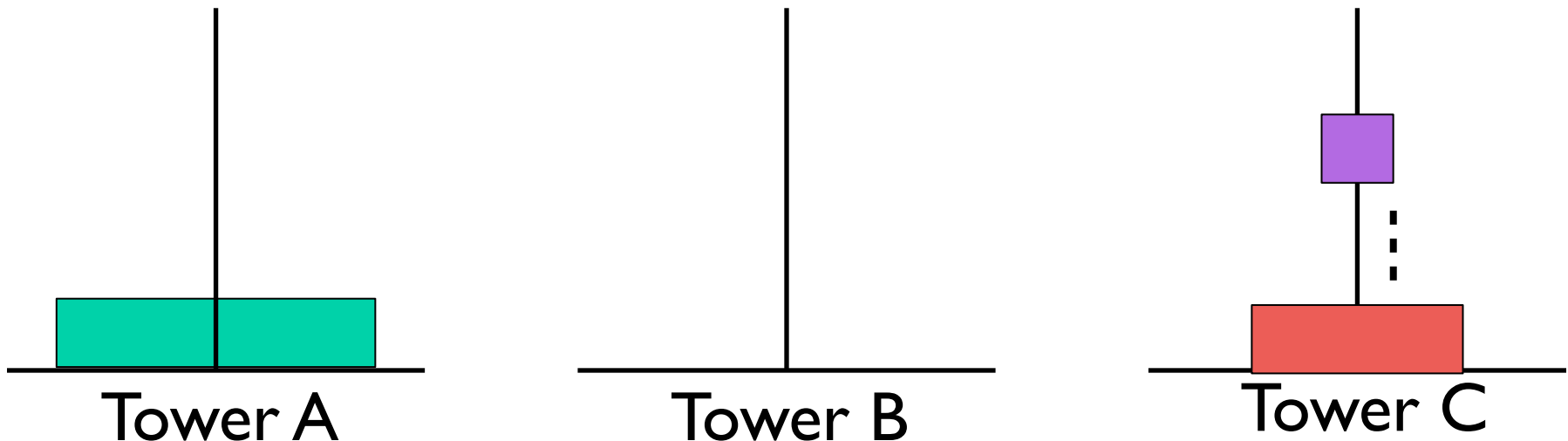
- python turtle graphics
 - Module: minimal_hanoi.py
 - Modified to define number of disks to sort as command line argument
 - Invocation:
 - `python hanoi.py <n>`



Tower of Hanoi

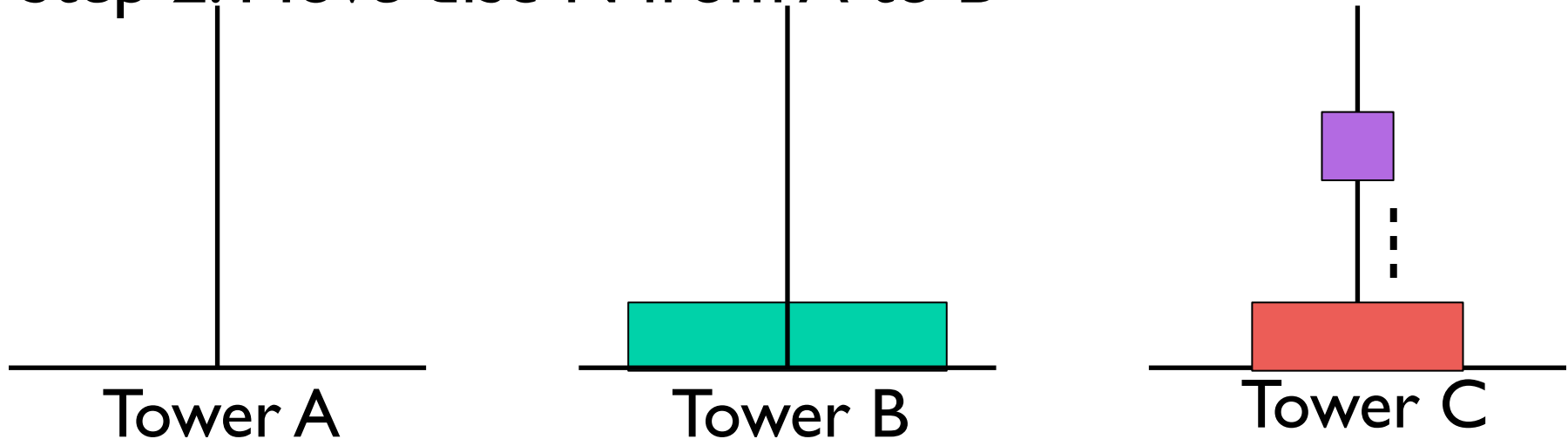
- Task: move N discs from A to B using C
- I/p: N disks on Tower A, o/p: N disks on Tower B
- Algo specification
 - S1: Move top N-1 discs from A to C using B
 - S2: Move largest disc to tower B
 - S3: Move top N-1 discs from C to B using A

Step 1: Move N-1 discs from A to C using B

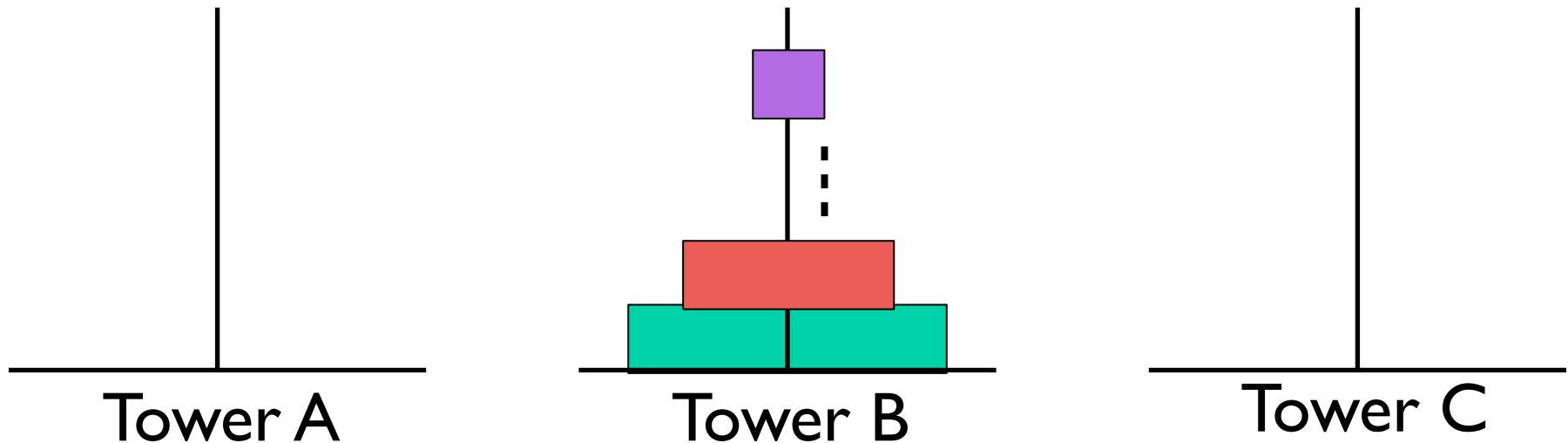


Tower of Hanoi: Algo Specification

- Step 2: Move disc-N from A to B



- Step 3: N-1 discs from C to B using A



Permutation of N number

- Tasks: Given N items, print all of its permutations
- Example: given $S = \{a,b,c\}$
- Output: $\{a,b,c\}, \{a,c,b\}, \{b,a,c\}, \{b,c,a\}, \{c,a,b\}, \{c,b,a\}$
- Algo specification for n items in an array $a[1] \dots a[n]$
 - for $i = 1$ to n
 - print $a[i]$
 - Let $b[1..n-1] = a[1..i-1] + a[i+1..n]$ i.e. excluding $a[i]$
 - print all permutations of $b[1..n-1]$ # recursion part

Exercises Using Recursion

- Generate n sequences for Fibonacci series, given
 $a_1=1, a_2=1$
- Specify Horner's rule in recursive way
– $A(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$
- Given N boolean variables, print all possible combinations of possible truth values
– e.g. for 3 boolean variables, the possible values are
TTT, TTF, TFT, TFF, FTT, FTF, FFT, FFF
- Generate Power set of S
 - e.g. if $S = \{a, b, c\}$, then the power set is
 $\{\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}$

Summary

- Algorithm criteria
- Design strategies
- Algorithm specification
- Sample algorithms (with recursion)
 - Hanoi's tower
 - Permutation of N numbers
 - Fibonacci series
 - Horner's rule
 - All possible truth values for N variables
 - Power set of S.