Applications of STACKS

Traversal Algorithm of stack

Traversal(STACK,N,TOP, ,MAXSTACK): to traverse the stack

1. Set TOP:=MAXSTACK.

2. Repeat step 3 and 4 While TOP!=NULL

 3. Write: STACK[TOP]

 4. TOP:=TOP-1.

 5.EXIT

Algorithm of searching in stack

Searching(STACK,N,TOP, ITEM,MAXSTACK): To search element ITEM in the stack

1. Set TOP:=MAXSTACK [and Read: ITEM(optional)]
2. Repeat step 3 and 4 While TOP!=NULL
3. IF STACK[TOP]=ITEM,then:

   Write: Element found and Exit

   Else: TOP:=TOP-1.

5.EXIT

QUICK SORT

- One useful application of stack is to sort a number of elements using quick sort which is also known as partition exchange sort.
- It is based on the idea that it is easier and faster to sort two smaller list than one large list.
- It is based on divide-and-conquer strategy.
- In this technique a given problem is divided into a number of more smaller sub problems and so on till a sub problem is not decomposable.
- The general idea of quicksort is to select one element from a list of elements known as pivot element around which other elements will be rearraned.

Partitioning Procedure

- **PARTITION(SA,N,BEG,END,LOC)-**Given SA is an array of N elements BEG and END contains lower and upper bound of the subarray on which partitioning process is applied. We have used local variable LEFT and RIGHT that contain the indices of the end elements of the subarray that are yet to be scanned. This algorithm divides the subarray with indices ranging from BEG to END into subarrays and return the location LOC of pivot element.

1. Set LEFT:=BEG and RIGHT:=END and LOC:=BEG . [Intilization]

2. [Steps 2 to 4 perform scanning from right to left]

(a) Repeat while SA[LOC]<=SA[RIGHT] and LOC!=RIGHT

RIGHT:=RIGHT-1 [Move towards left]

[End of while loop]

(b) If LOC=RIGHT, then: [Pivot element located?]

Write:LOC and Exit [end of if structure]

(c) If SA[LOC]>SA[RIGHT],then:

i) a) TEMP:=SA[LOC] [Swap A[LOC] and A[RIGHT]]

b) SA[LOC]:=SA[RIGHT]

c) SA[RIGHT]:=TEMP

ii) Set LOC:=RIGHT [Set new location LOC of pivot element after swapping]

iii) Go to step 3.

[end of if structure]

3.[Steps 5 to 7 perform scanning from left to right]

(a)Repeat while SA[LOC]>=SA[LEFT] and LOC!=LEFT

LEFT:=LEFT+1 [Move towards right]

[End of while loop]

(b) If LOC=LEFT,then: [Pivot element located?]

Write:LOC and Exit

[End of if structure]

(c) If SA[LOC]<SA[LEFT] then

i) a) TEMP←SA[LOC] [Swap S[LOC] and S[LEFT]]

b) SA[LOC]←SA[LEFT]

c) SA[LEFT]←TEMP

ii) LOC:=LEFT [Update LOC after swapping]

iii) Goto Step2 [Repeat the steps]

[end of if structure]

QUICK SORT using Recursion

- **QUICKSORT(SA,N,BEG,END)**-*Given SA be an array of N elements.This algorithm sorts the array elements in ascending order.BEG represents initial index and END represents last index of the array*
- 1. If BEG<END,then:
- LOC =PARTITION(A, BEG, END)

3. QUICKSORT(A, BEG, LOC-1)

4. QUICKSORT(A, LOC+1, END)

5. Exit

- Worst case
  - $O(n^2)$

- Average case
  - O(n log n)

- Best case

  O(n log n)

Complexity of Quick Sort

- Worst case
  - $O(n^2)$
- Average case
  - $O(n \log n)$
- Best case

  $O(n \log n)$

Merge Sort

- Divide and Conquer
- Recursive in structure
  - ***Divide*** the problem into sub-problems that are similar to the original but smaller in size
  - ***Conquer*** the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.
  - ***Combine*** the solutions to create a solution to the original problem

An Example: Merge Sort

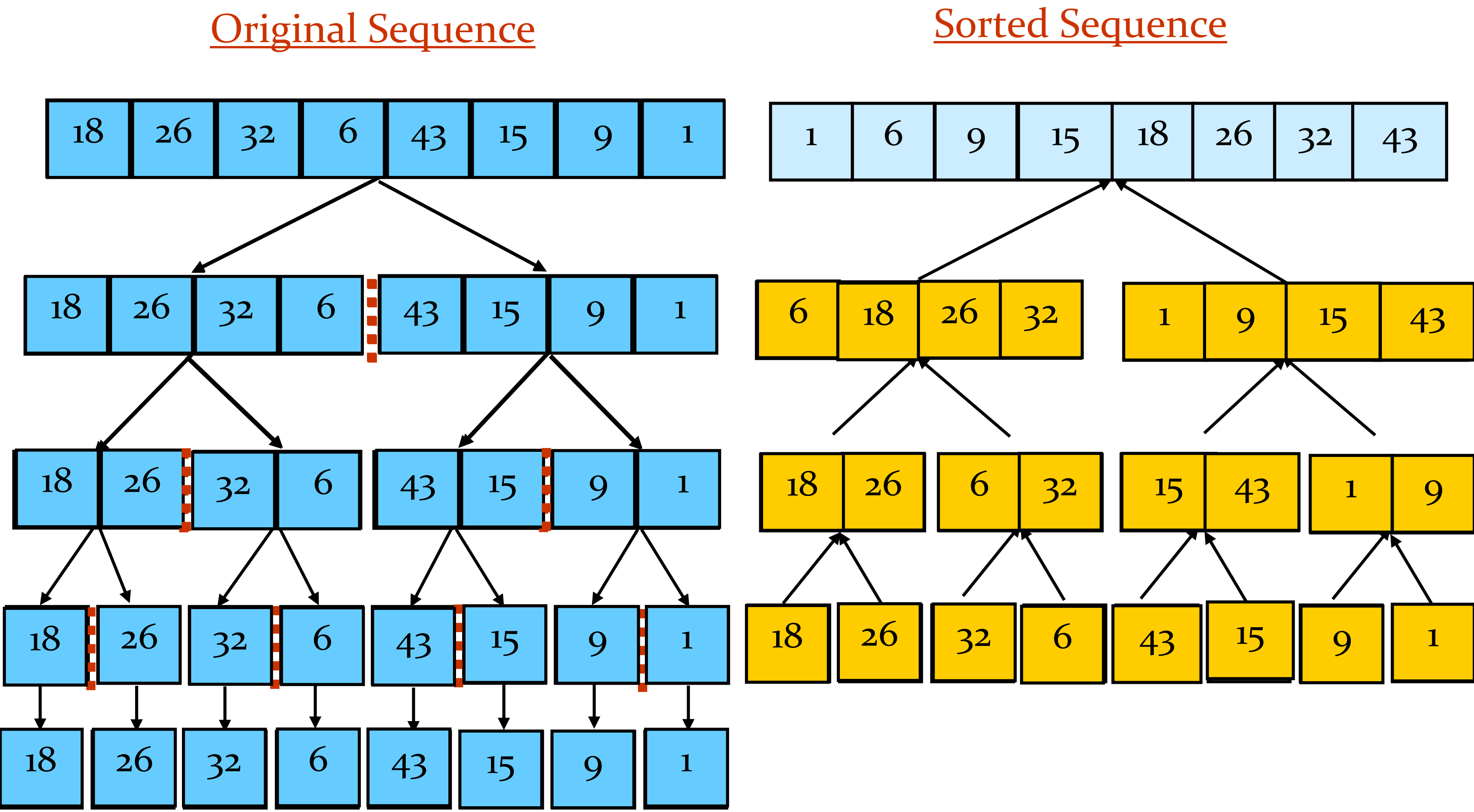
***Sorting Problem*:** Sort a sequence of *n* elements into non-decreasing order.

***Divide*:** Divide the *n*-element sequence to be sorted into two subsequences of *n/2* elements each

Conquer: Sort the two subsequences recursively using merge sort.

***Combine*:** Merge the two sorted subsequences to produce the sorted answer.

Merge Sort – Example



Merge-Sort (A, Beg, End)

INPUT: a sequence of *n* numbers stored in array A
OUTPUT: an ordered sequence of *n* numbers

***MergeSort* (*A, Beg, End*):**sort *A*[*N*] by divide & conquer

if *Beg*< *End,Then:*

Mid := (*Beg*+*End*)/2

MergeSort (*A*, *Beg*, *Mid*)

MergeSort (*A*, *Mid*+1, *End*)

Merge (*A*, *Beg*, *Mid*, *End*) [merges *A*[*beg to mid*] with *A*[*mid +1 to end*]]

Initial Call: MergeSort(A, 1, n)

Merge two arrays

Merge(*A*, *Beg*, *Mid*,*End*)

1 Set $N1$:= Mid-Beg+1

2 *Set N2:= End-Mid*

3. Repear for $I=1$ *to* $N1$

  Set $L[I]:=A[Beg + I - 1]$

4. Repear for $J=1$ *to* $N2$

  Set $R[J]:=A[Mid+J]$

5. Set $L[n_1+1] := \infty$

6. Set $R[n_2+1] := \infty$

7. Set $I:=0$

8. Set $J:=0$

9. Repeat for k =Beg to End

  if $L[I] \leq R[J]$,then:

  $A[k]:= L[\mathrm{I}]$

  Set I:=I+1

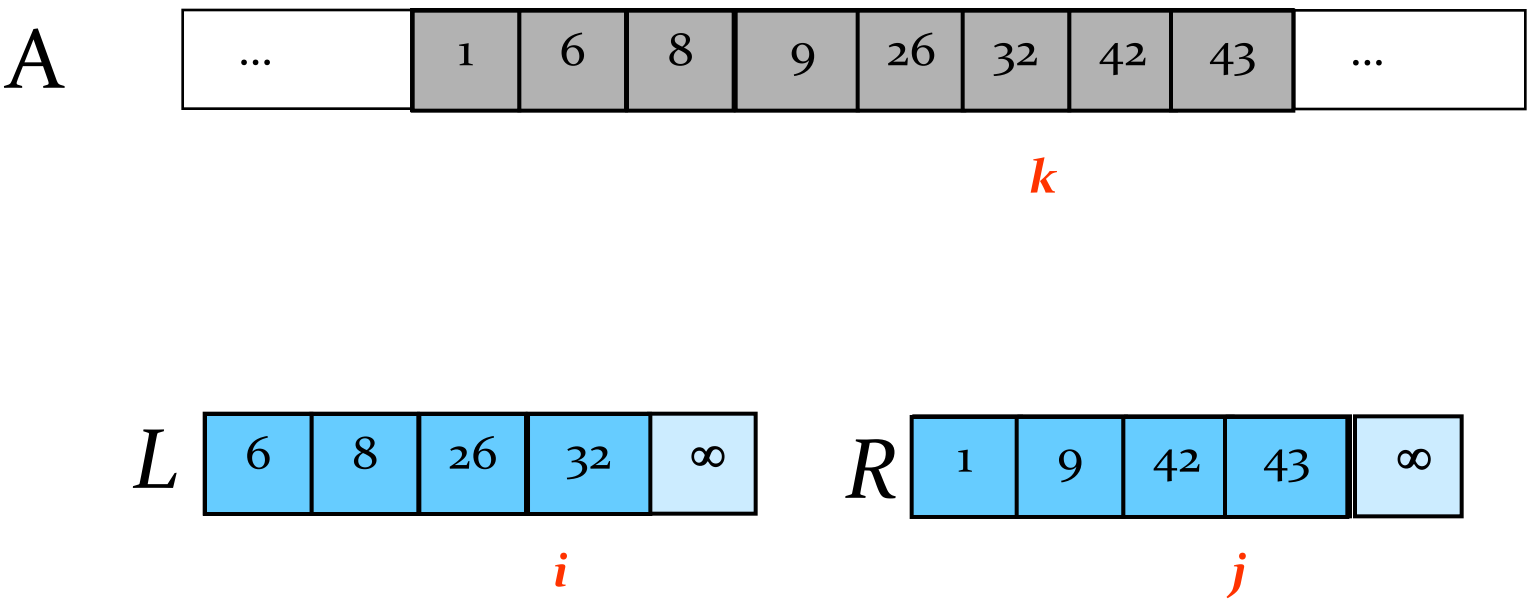
  else:

  Set $A[k] := R[j]$

  Set J:=J+1

10. Exit

Merge – Example



Analysis of Merge Sort

Running time $T(n)$ of Merge Sort:
Divide: computing the middle takes $\Theta(1)$
Conquer: solving 2 sub-problems takes $2T(n/2)$
Combine: merging n elements takes $\Theta(n)$
Total:

$$T(n) = \Theta(1) \quad \text{if } n = 1$$
$$T(n) = 2T(n/2) + \Theta(n) \quad \text{if } n > 1$$

$$\begin{aligned}
T(n) &= 2\,T(n/2) + n \\
&= 2\,((n/2)\log(n/2) + (n/2)) + n \\
&= n\,(\log(n/2)) + 2n \\
&= n \log n - n + 2n \\
&= n \log n + n \\
&= O(n \log n)
\end{aligned}$$

Comparing the Algorithms

	Best Case	**Average Case**	**Worst Case**
Bubble Sort	O(n)	O(n^2)	O(n^2)
Insertion Sort	O(n)	O(n^2)	O(n^2)
Selection Sort	O(n^2)	O(n^2)	O(n^2)
Merge Sort	O($n \log n$)	O($n \log n$)	O($n \log n$)
Quick Sort	O($n \log n$)	O($n \log n$)	O(n^2)
Heap Sort	O($n \log n$)	O($n \log n$)	O($n \log n$)

Thank You

TOWERS OF HANOI

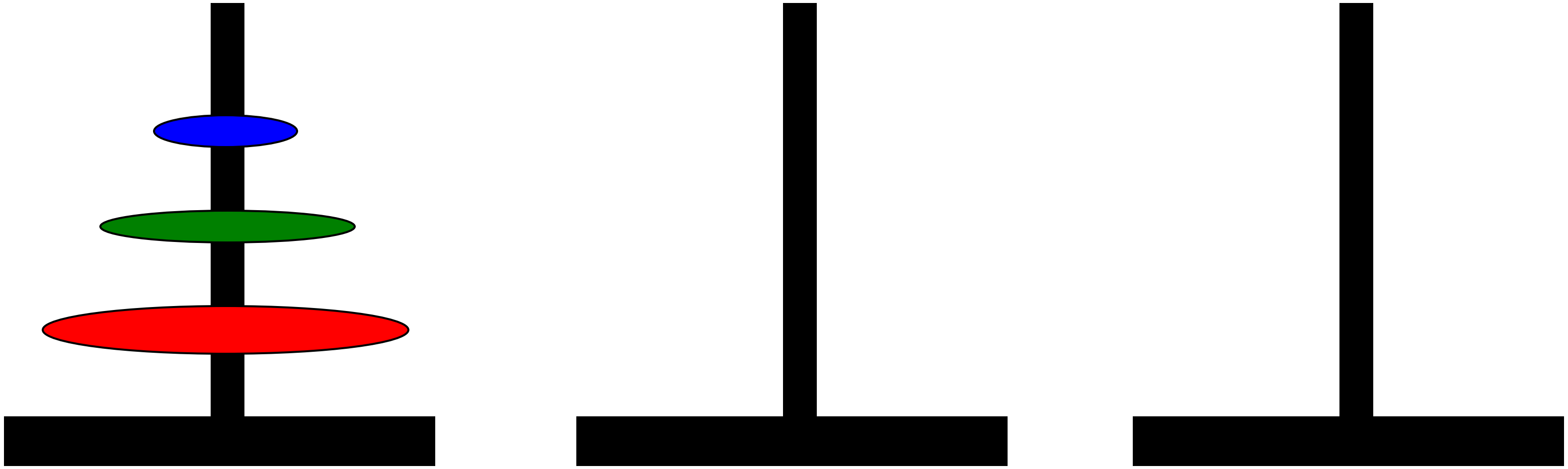
- Tower of Hanoi is the most common recursive problem
- The objective is to shift all disk from peg A to Peg C using Peg B.
- The shifting of disks is restricted by following rules:

i) Only one disk can be shifted at a time.

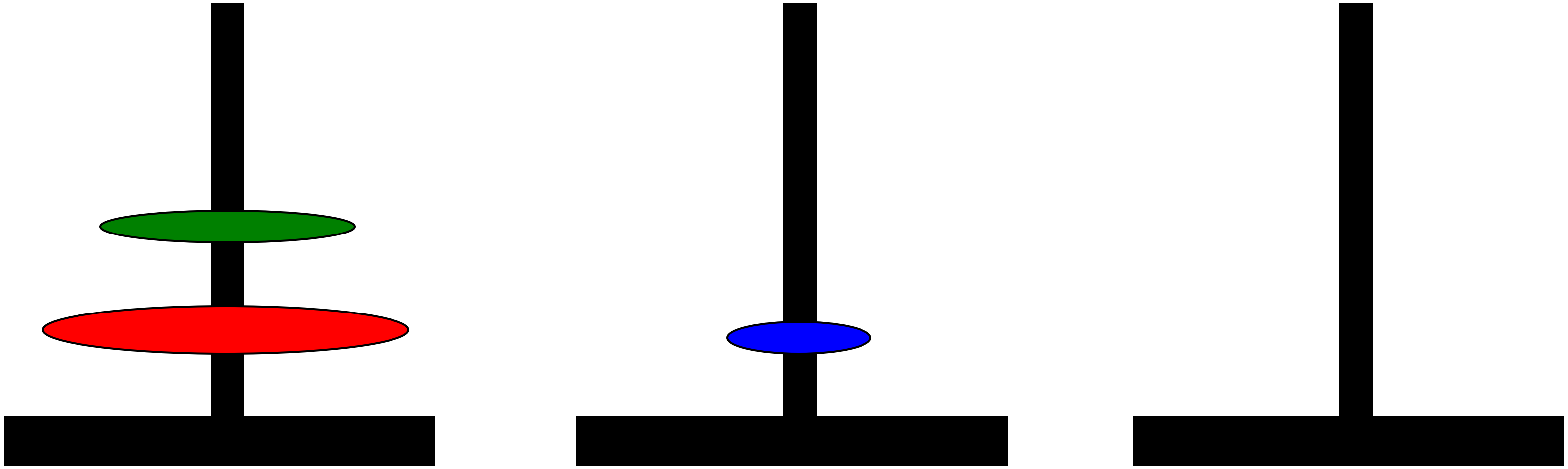
ii) Only top disk on any peg may be shifted to any other peg.

iii) At no time can a larger disk be placed on a smaller disk

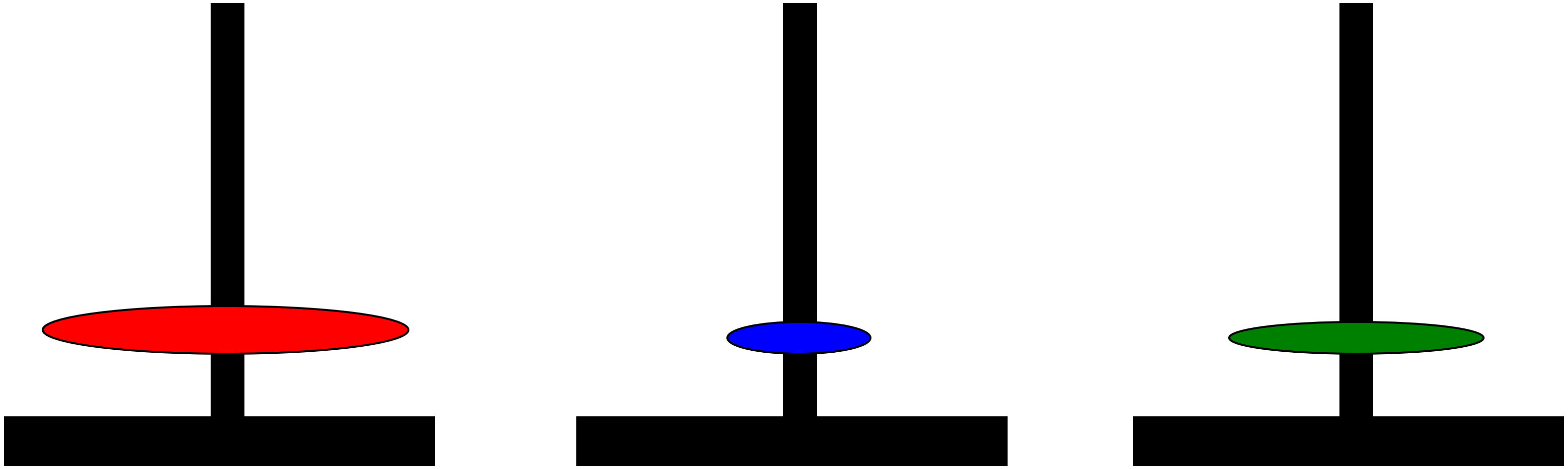
Tower of Hanoi



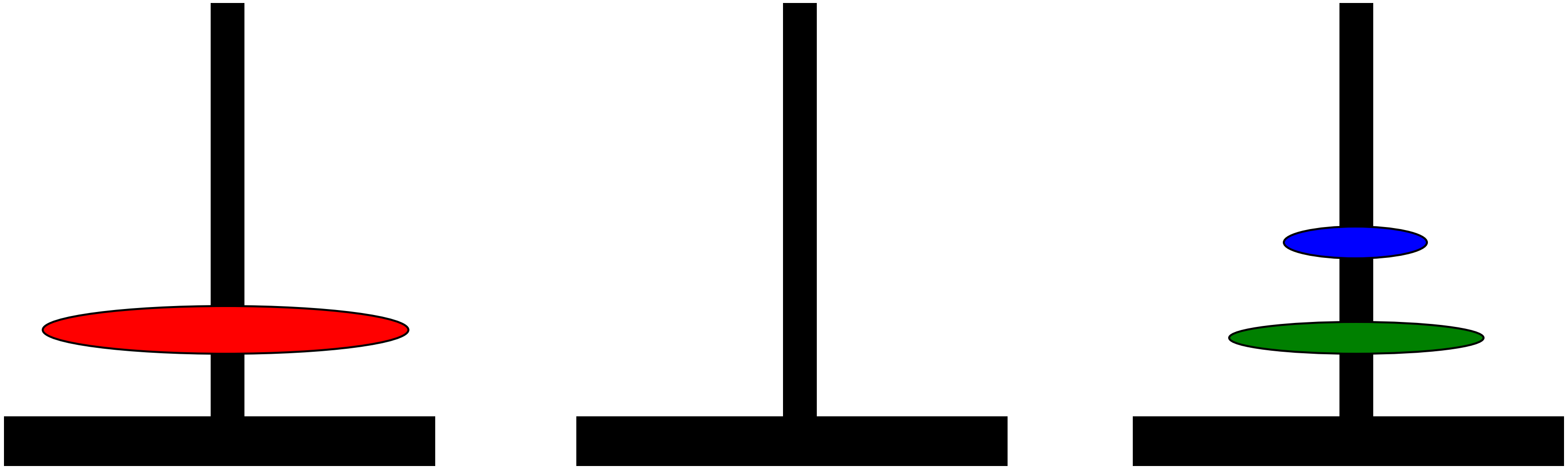
Tower of Hanoi



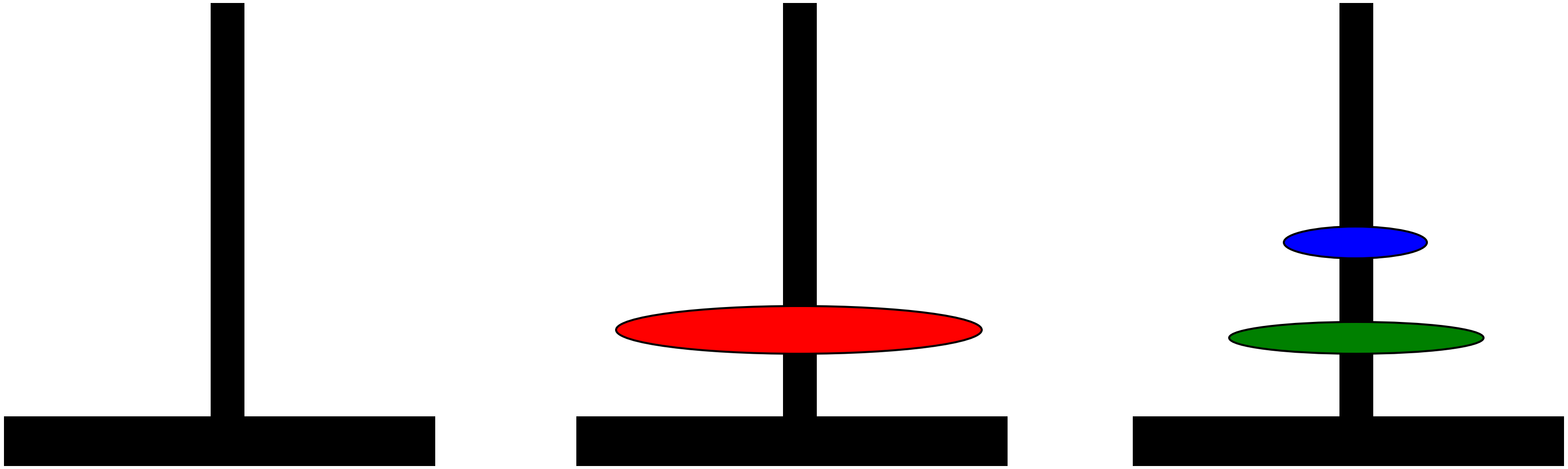
Tower of Hanoi



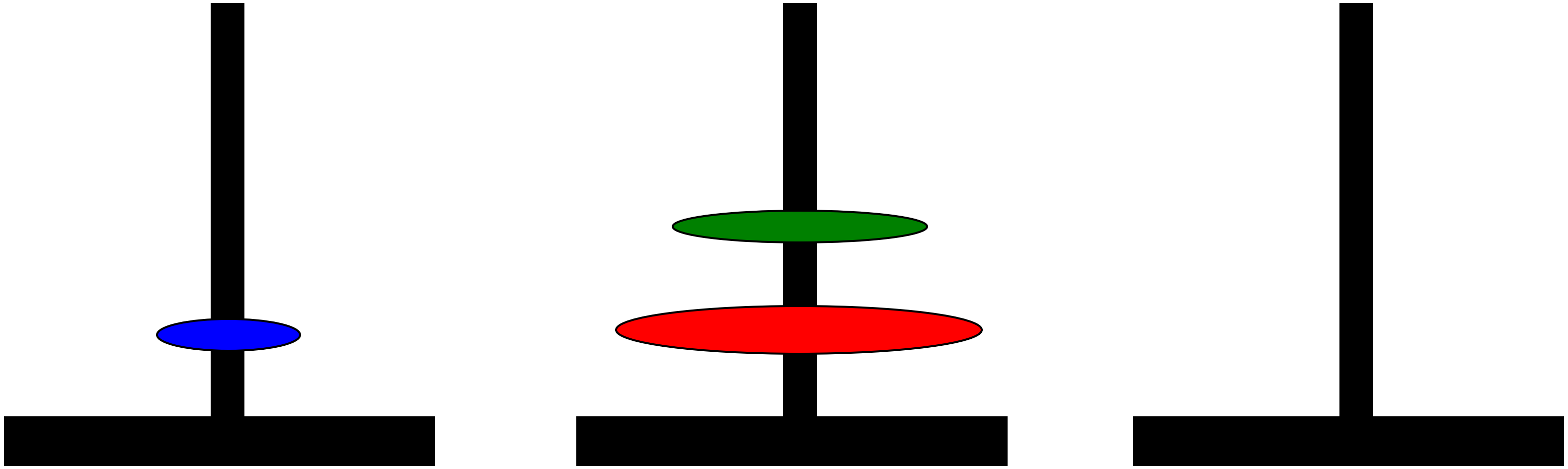
Tower of Hanoi



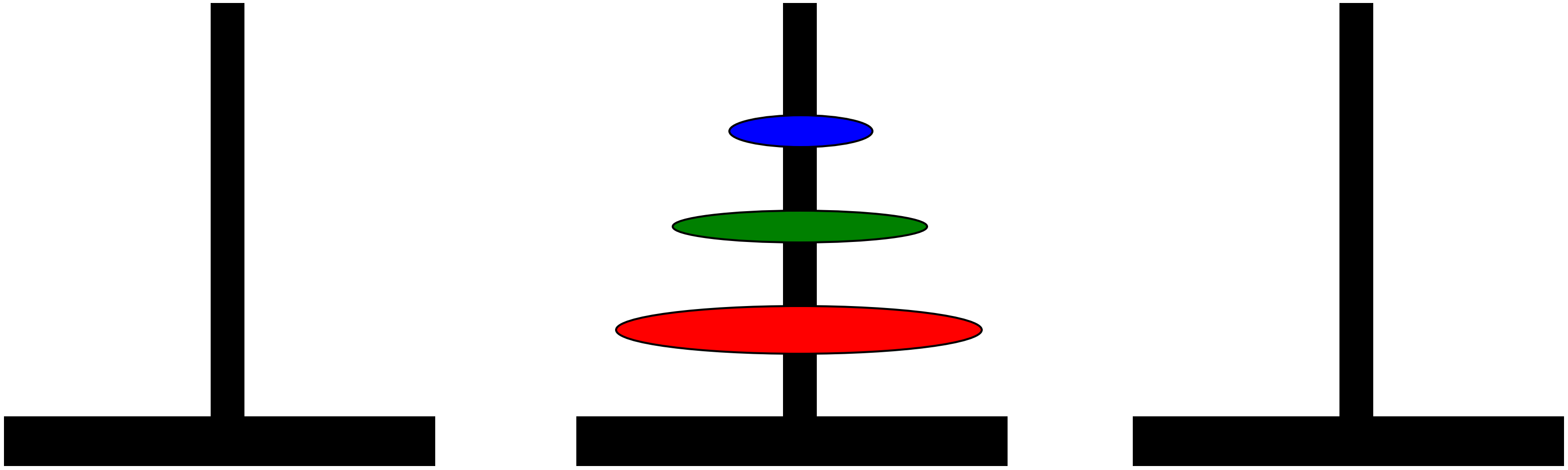
Tower of Hanoi



Tower of Hanoi



Tower of Hanoi



TOWERS of HANOI

- **TOI(N,BEG,END,AUX)**-This algo shifts N(>0) number of disks from the source peg (BEG) to the destination peg(END) using the intermediate peg(AUX).

1.If N=1,the: then [shifting a single disk]

Write : BEG -> End and Exit [shift single disk from “ BEG “to” END]

2.Call TOWER(N – 1, BEG, END, AUX). [Shifting N-1 disk recursively]

3. Write: BEG – END.

4. Call TOWER(N – 1, AUX, BEG, END).

5. Exit

- A->B
- A->C
- B->C
- A->B
- C->A
- C->B
- A->B
- A->C

- B->C
- B->A
- C->A
- B->C
- A->B
- A->C
- B->C

(Complexit=2^n exponential)