

Lecture # 6

- Towers of Hanoi
 - Sorting Algorithms
- 
- The background of the slide features several faint, concentric circles in a lighter shade of blue, resembling ripples in water. These circles are positioned in the lower right quadrant of the slide.

Office Hours (Students' Consultation Time)

- DS/Applied Algorithm

Thursday (9:00-12:00 Noon)

- Adv OS

Thursday (9:00-12:00 Noon)

- Analysis of Algorithm

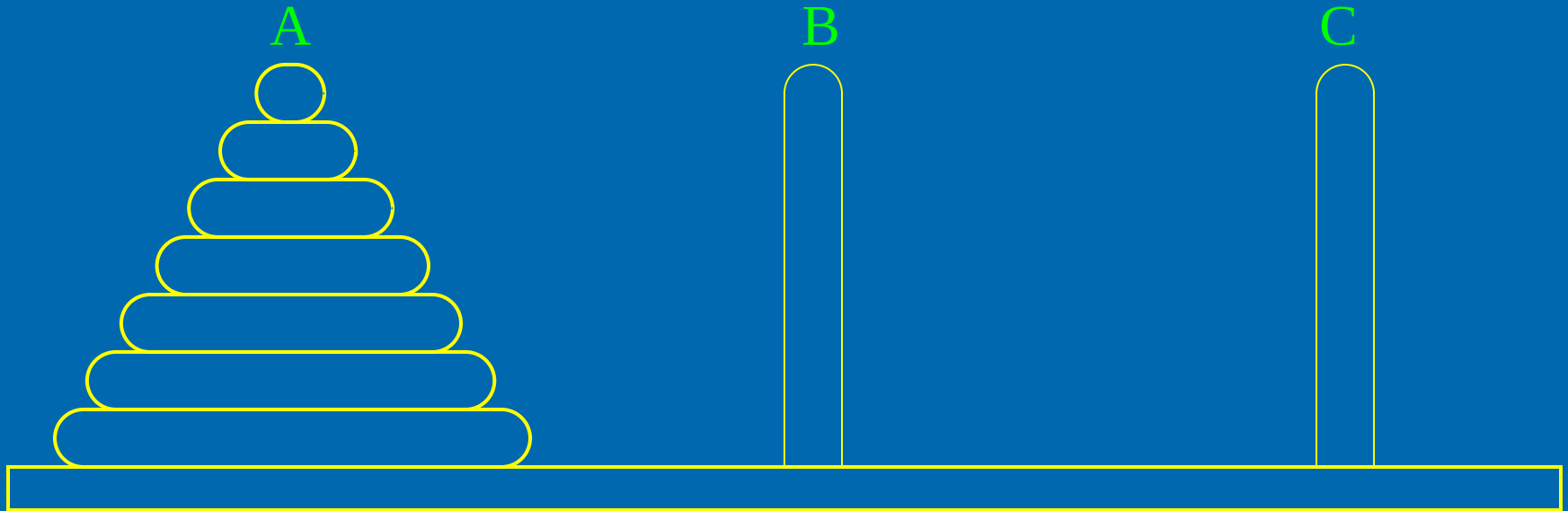
Fri (10:00 – 1:00pm)

- Any other time by appointment

- Please observe office hours for
(academic/nonacademic) consultation/advice

Towers of Hanoi

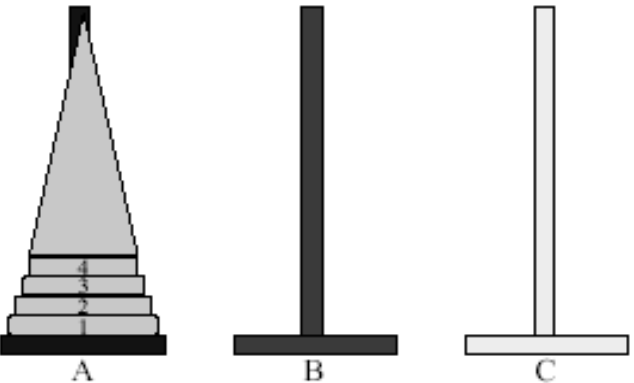
- It provides analysis of recursive algorithm
- If rods = 3 and rings = 64 of different sizes
- All rings are arranged in decreasing size on rod 1 (the largest at the bottom and the smallest at the top)
- Transfer all the rings to the 3rd rod
 - Move one ring in one move
 - No ring is even placed on top of another smaller one on any rod
- *Time required to finish this work for 64 rings is 500,000 million years if one ring is moved in one second*



Initial setup of Towers of Hanoi with $n = 7$

Towers of Hanoi

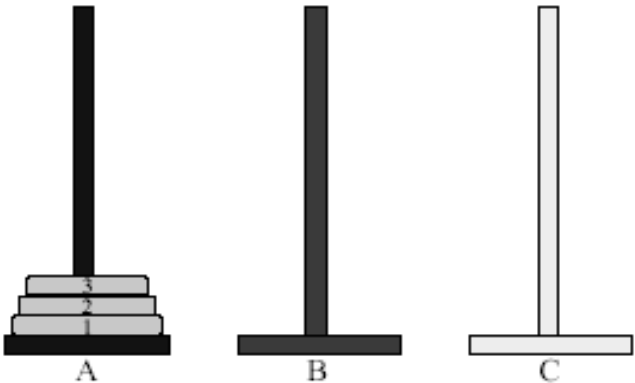
Towers Of Hanoi



The diagram shows three vertical towers labeled A, B, and C. Tower A is on the left and is covered by a large, light-gray cone, with only the top four disks visible. Tower B is in the middle and is empty. Tower C is on the right and is empty. The disks on tower A are of increasing size from bottom to top, with the top disk being the smallest.

- 64 gold disks to be moved from tower A to tower C
- each tower operates as a stack
- cannot place big disk on top of a smaller one

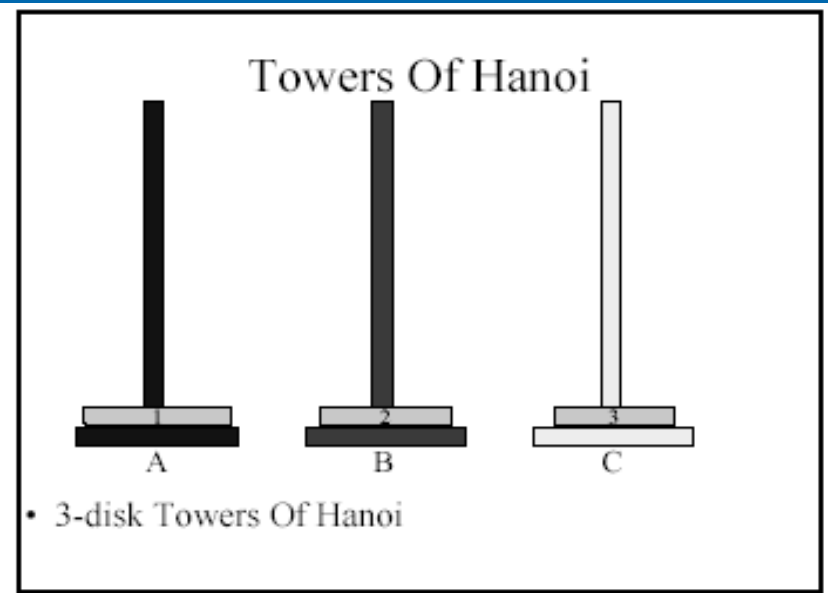
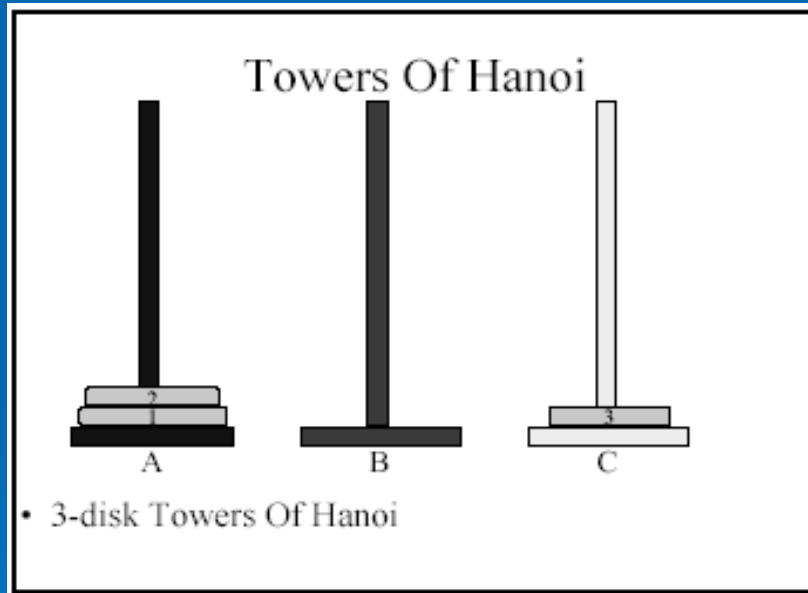
Towers Of Hanoi



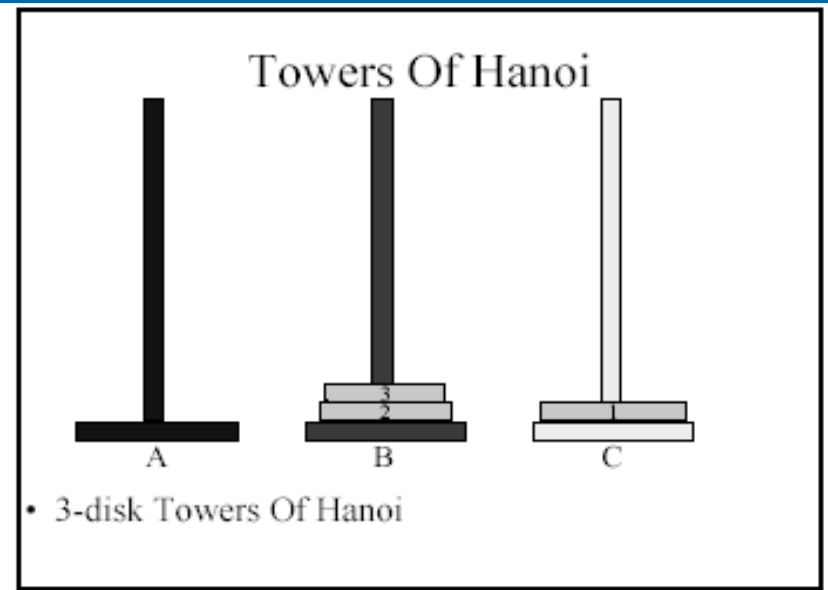
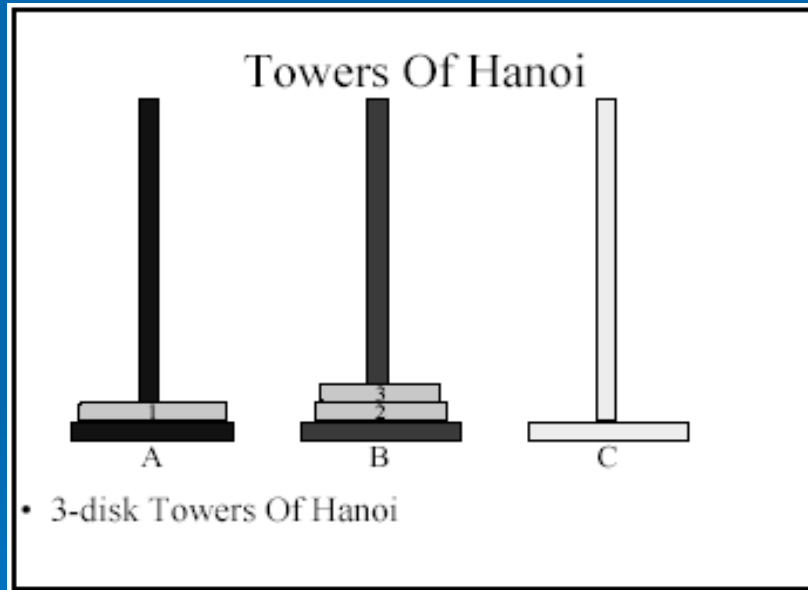
The diagram shows three vertical towers labeled A, B, and C. Tower A is on the left and has three disks stacked on it. Tower B is in the middle and is empty. Tower C is on the right and is empty. The disks on tower A are of increasing size from bottom to top, with the top disk being the smallest.

- 3-disk Towers Of Hanoi

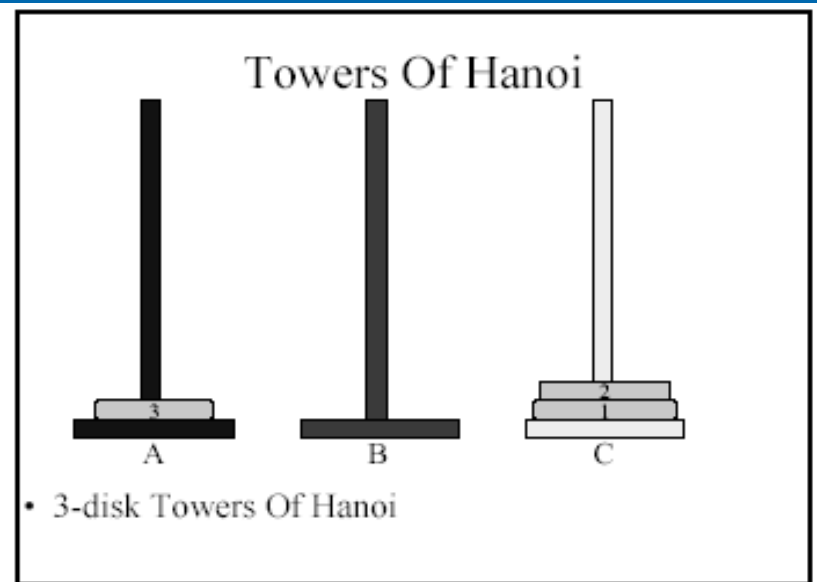
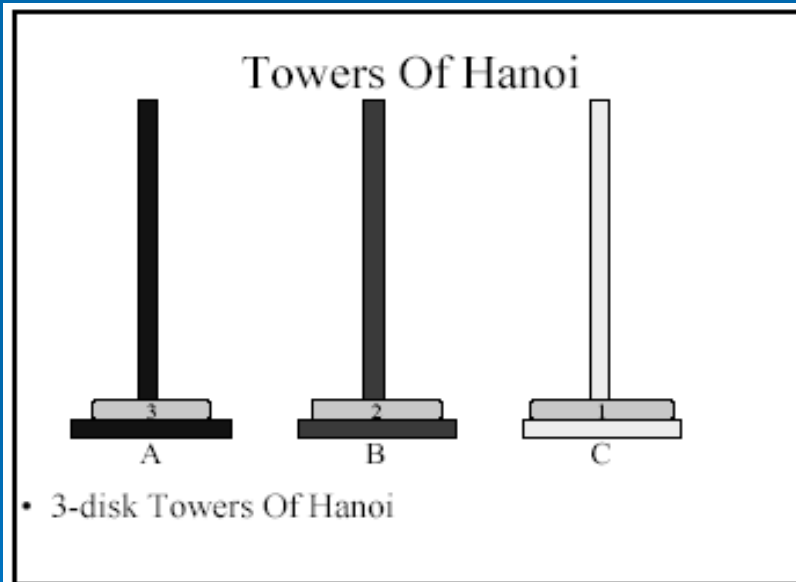
Towers of Hanoi



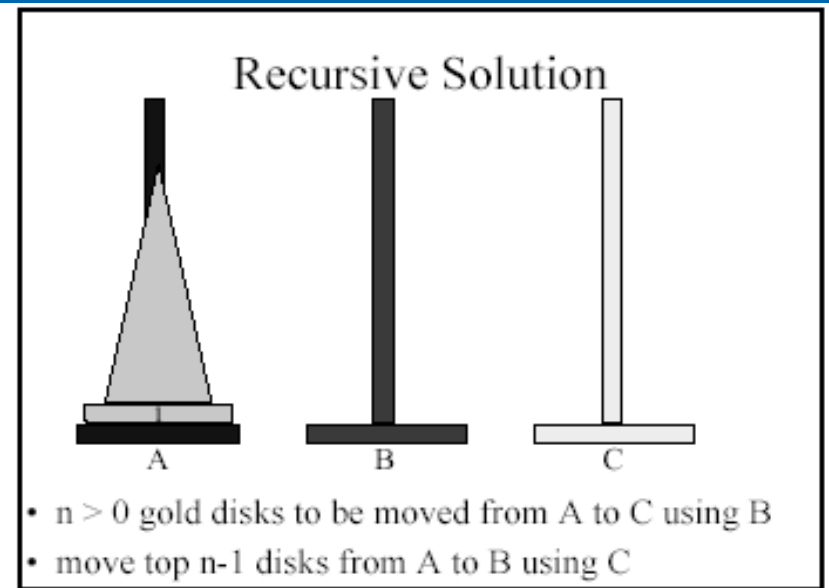
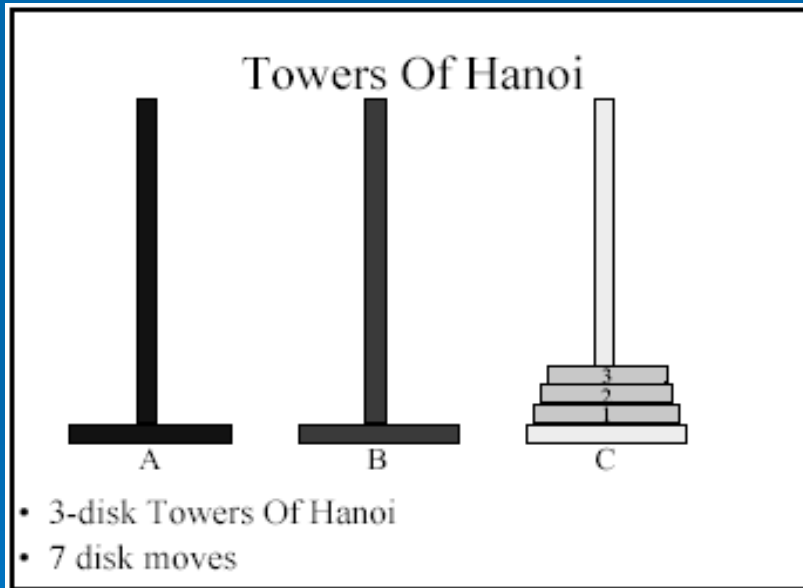
Towers of Hanoi



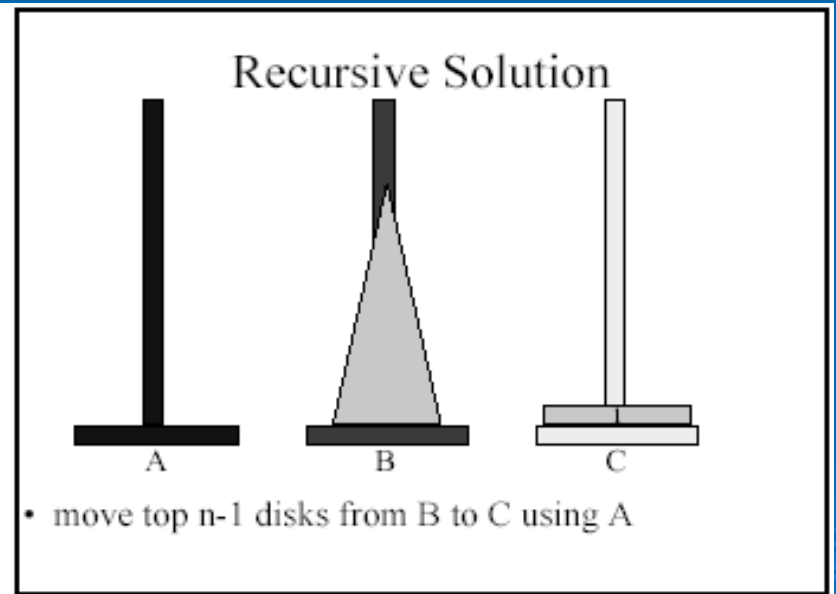
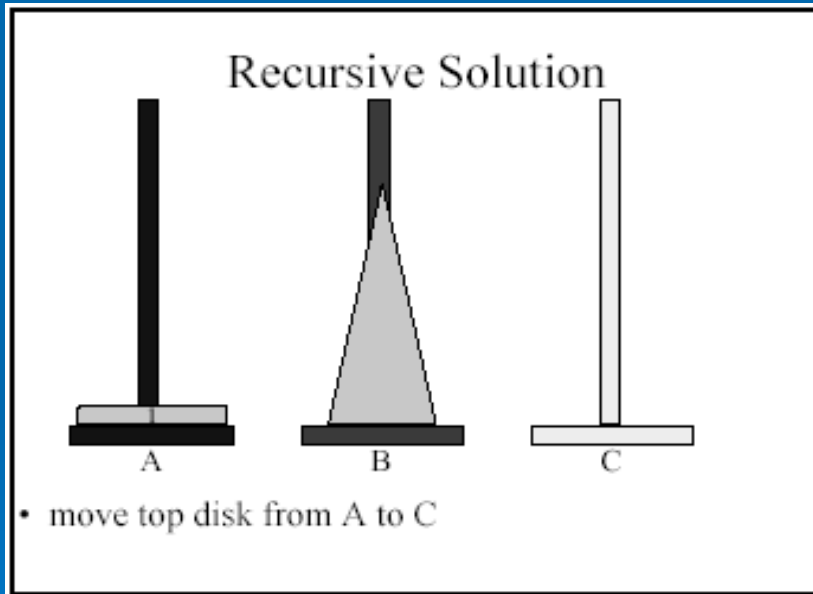
Towers of Hanoi



Towers of Hanoi



Towers of Hanoi



TOWER (N, BEG, AUX, END)

If N=1 then BEG "→" END

If N>1 then

 Tower (N-1, BEG, END, AUX)

 write BEG "→" END

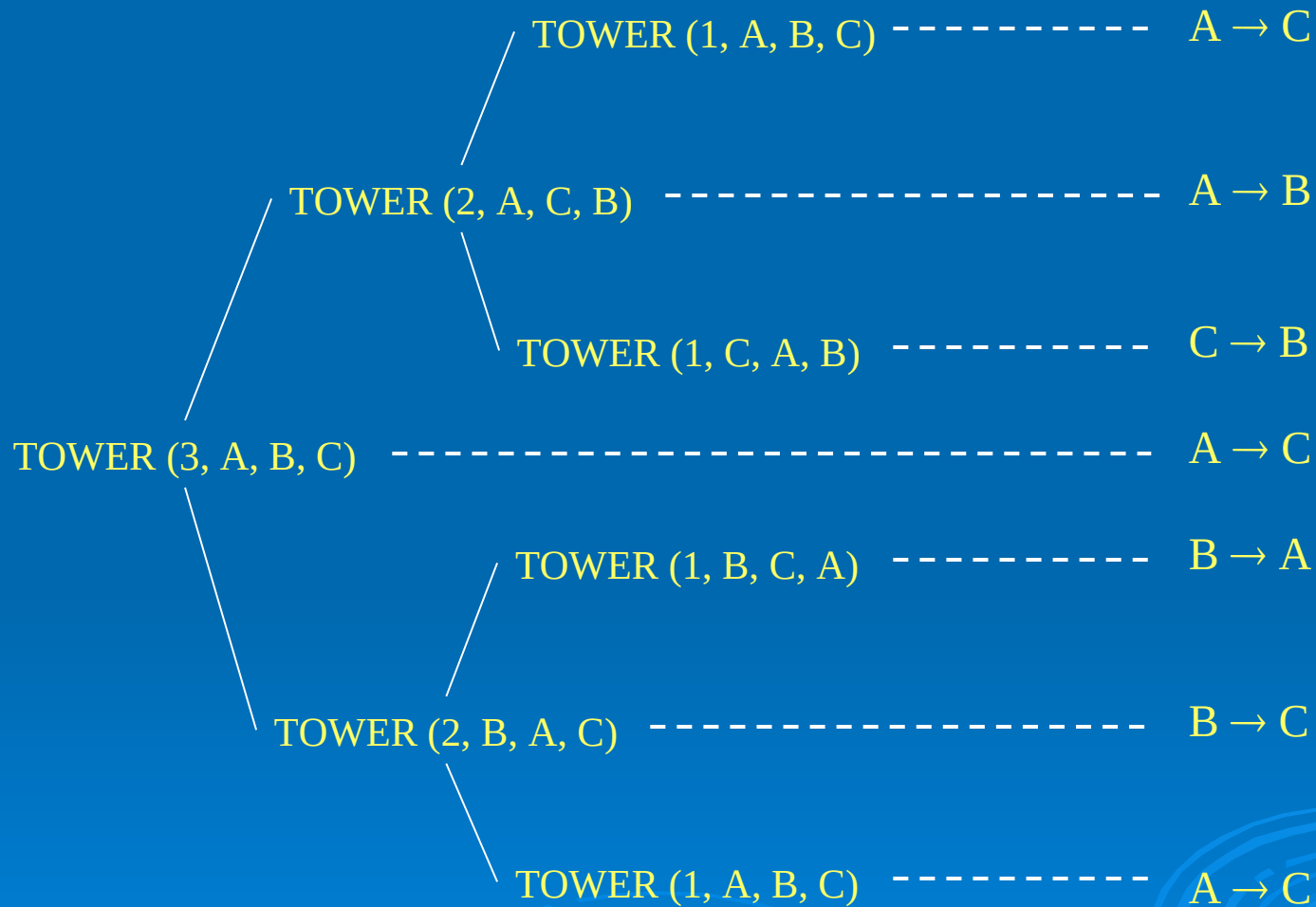
 Tower (N-1, AUX, BEG, END)

- *Let $T(\mathcal{N})$ denote the number of times it is executed on a call of \mathcal{N} .*

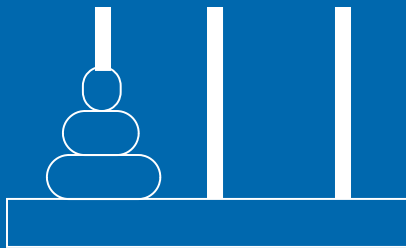
$$T(N) = \begin{cases} 0 & N = 0 \\ 2T(N - 1) + \Theta(1) & N > 0 \end{cases}$$

- *Number of moves are: $2^{\mathcal{N}} - 1$*
- *Time to solve the problem with \mathcal{N} rings = $\theta(2^{\mathcal{N}})$*

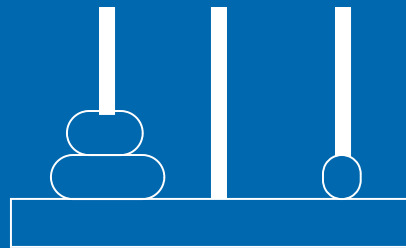
$$n = 3$$



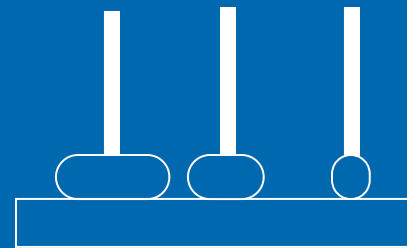
$n = 3$



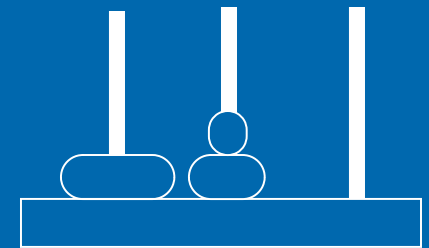
(a) Initial



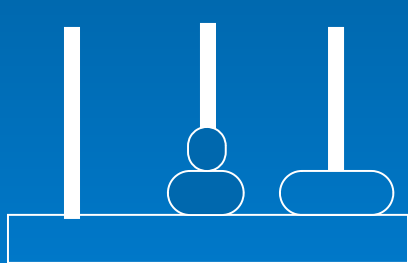
(1) $A \rightarrow C$



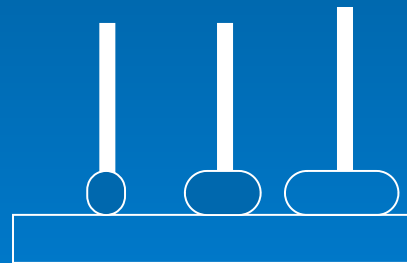
(2) $A \rightarrow B$



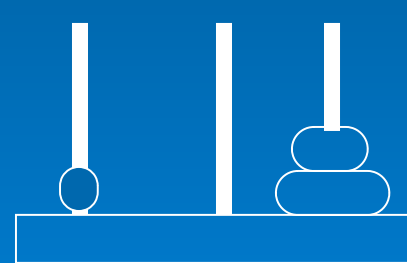
(3) $C \rightarrow B$



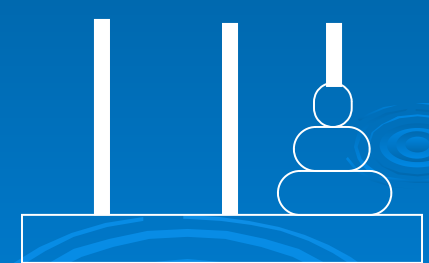
(4) $A \rightarrow C$



(5) $B \rightarrow A$



(6) $B \rightarrow C$



(3) $A \rightarrow C$

Non Credit Assignment

- Implement the problem of *Towers of Hanoi* in any language of your choice.
- Also view that increasing **N** how much your system becomes slow down.

Selection Sort

- The basic idea of Selection Sort is to make a number of passes through the list or part of the list and, on each pass, select one element to be correctly positioned.
- For example, on each pass through a sub list, the smallest element in the sub list might be found and then moved to its proper location.

- As an illustration, suppose that following list has to be sorted into ascending order :

67 33 21 84 49 50 75

- We scan the list to locate the smallest element and find it in **position 3**:
- Then we interchange this element with the first element and thus properly position the smallest element at the beginning of the list.

21 33 **67** 84 49 50 75

- We now scan the sub list consisting of the elements from **position 2** on to find the next smallest element

21 33 67 84 49 50 75

and exchange it with the second element (itself in this case) and then properly position the next-to-smallest element in **position 2**

- We continue in this fashion, locating the smallest element in the sub list of elements from **position 3** and interchanging it with the third element, then properly positioning the smallest element in the sub list of elements from **position 4** on, and so on until we eventually do this for the sub list consisting of the last two elements as follows.

21 33 49 84 67 50 75

21 33 49 50 67 84 75

21 33 49 50 67 84 75

21 33 49 50 67 75 84

- Positioning the smallest element in this last sub list obviously also positions the last element correctly and thus completes the sort.

Analysis of Selection Sort

- Looking to the above algorithm of selection sort we observe that on the first pass through the list, the first item is compared with each of $n-1$ elements that follow it.
- On the second pass, second element is compared with the $n-2$ elements following it and so on. So a total of
$$(n-1) + (n-2) + (n-3) + \dots + 2 + 1 = n(n-1)/2$$
comparisons are required for any list.

and

$$\begin{aligned}(n-1) + (n-2) + (n-3) + \dots + 2 + 1 &= n(n-1)/2 \\ &= (n^2 - n)/2 \\ &= n^2/2 - n/2 \\ &= 0.5n^2 - 0.5n\end{aligned}$$

So ignoring smaller terms and coefficients we conclude that the worst case computing time for selection sort is $O(n^2)$.

Exchange Sort (Bubble Sort)

- Exchange sort systematically interchange pairs of elements that are out of order until eventually no such pairs remain and the list is therefore sorted.
- To illustrate bubble sort, consider the following list

67 33 21 84 49 50 75

- On the first pass, we compare the first two elements, 67 and 33 and interchange them because they are out of order.

67 33 21 84 49 50 75

33 67 21 84 49 50 75

- Now we compare the second and third elements, 67 and 21 and interchange them

33 21 67 84 49 50 75

- Next we compare 67 and 84 but do not interchange them because they are already in the correct order.

33 21 67 84 49 50 75

- Next 84 and 49 are compared and interchanged.

33 21 67 49 84 50 75

- Then 84 and 50 are compared and interchanged.

33 21 67 49 50 84 75

- Finally 84 and 75 are compared and interchanged.

33 21 67 49 50 75 84

So the first pass through the list is now complete.

- We are guaranteed that on this pass, the **largest** element in the list will go down to the end of the list, since it will obviously be moved and pass all smaller elements.
- But notice that some of smaller items have “Bubbled Up” toward their proper positions nearer the front of the list.
- So we scan the list again but this time we ignore the last item because it is already in its proper location.

33 21 67 49 50 75 84

So in each pass the last element will be placed in its proper location in the sub list.

- The *worst case* of bubble sort occurs when the list elements are in reverse order because in this case only one item (the largest) item is placed correctly on each pass through the list.
- On the *first* pass through the list $n-1$ comparisons and interchanges are made and only the largest element is correctly positioned.
- On the *next* pass, sub list consisting of the first $n-1$ elements are scanned; there are $n-2$ comparisons and interchanges; and the next largest element sink to position $n-1$.

- Finally, process continues until the sub list consisting of the first two elements is scanned and on this pass there is one comparison & interchange.
- Thus a total of
$$(n-1) + (n-2) + (n-3) + \dots + 1 = n(n-1)/2$$
$$= n^2/2 - n/2$$

Comparisons and interchanges are required. It follows that worst case computing time for Bubble Sort is $O(n^2)$