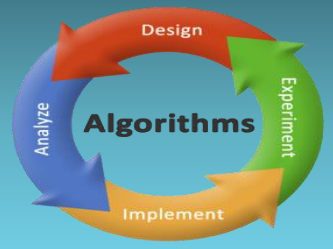
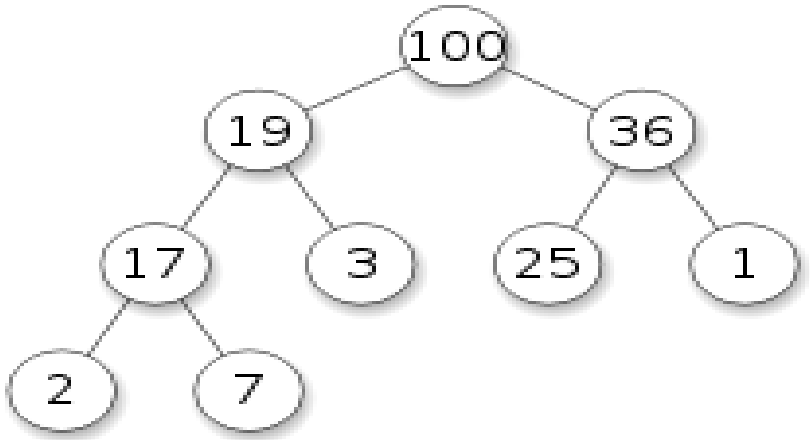
مرتب‌سازی هرمی

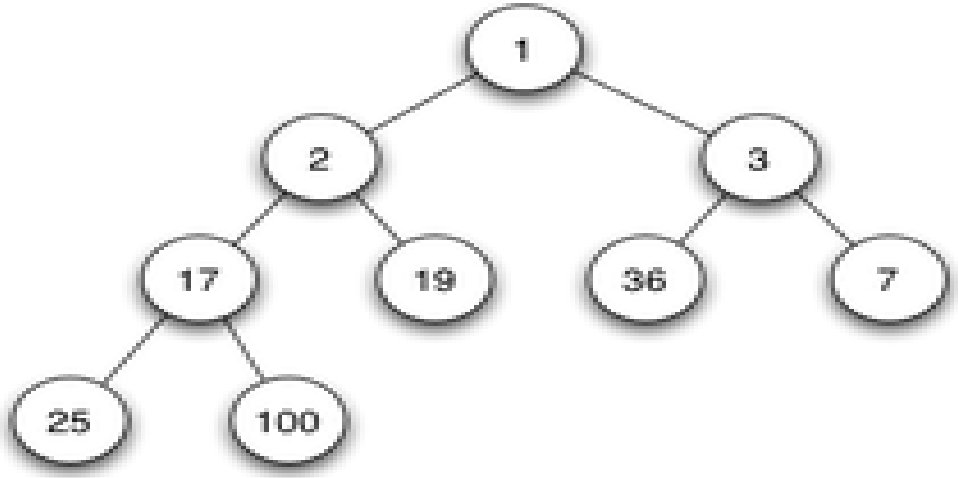


هرم (Heap)

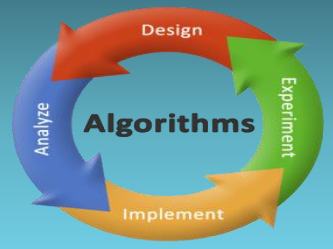
- یک هرم دودویی، یک درخت دودویی است با ۲ ویژگی (شرط) اضافه:
  - ✓ **ویژگی شکل**: یک درخت دودوی کامل است.
  - ✓ **ویژگی هرم**: درخت یا هرم حداکثر است، یعنی کلید هر گره بزرگتر یا مساوی هر دو فرزندش است، یا هرم حداقل است، یعنی کلید هر گره کوچکتر یا مساوی هر دو فرزندش است.



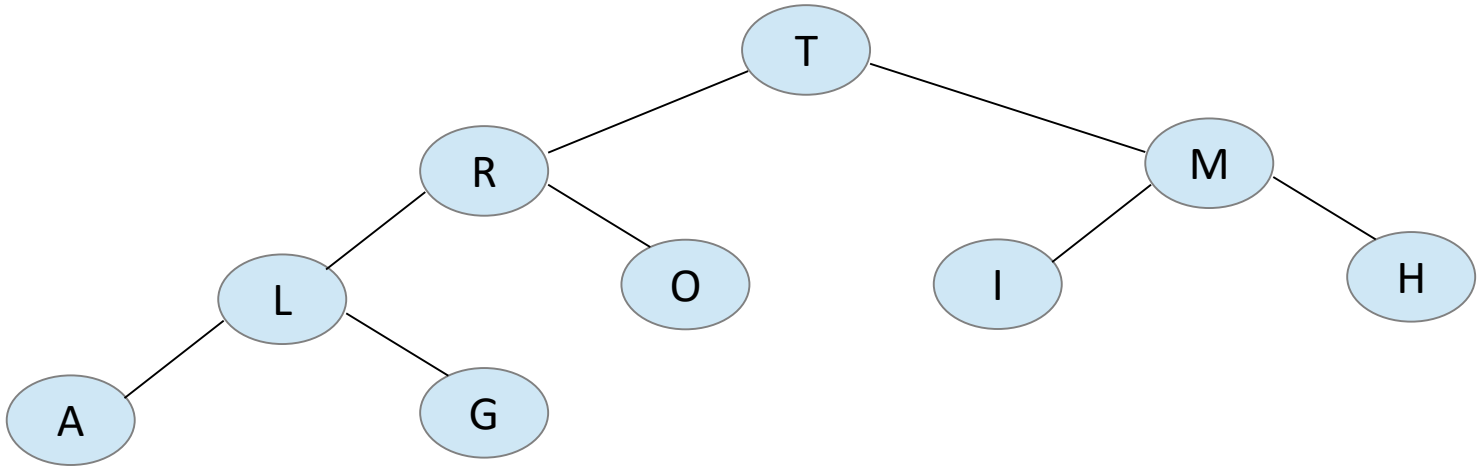
Max-Heap

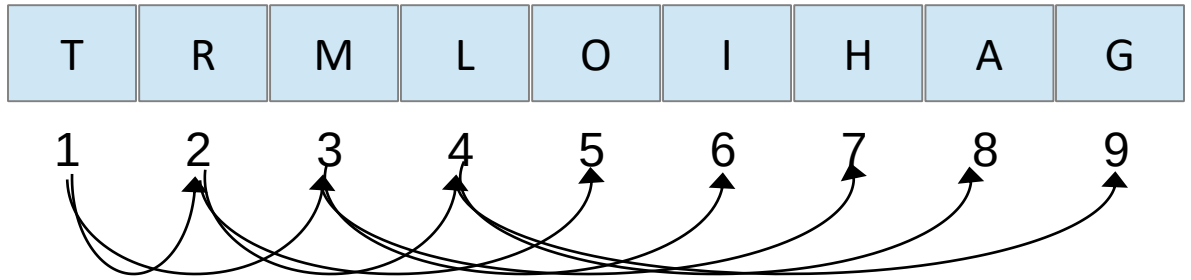


Min-Heap

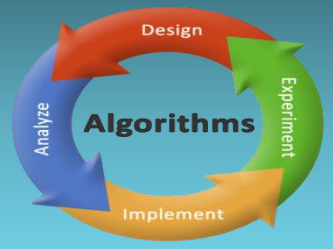


موقعیت فرزندان در هرم



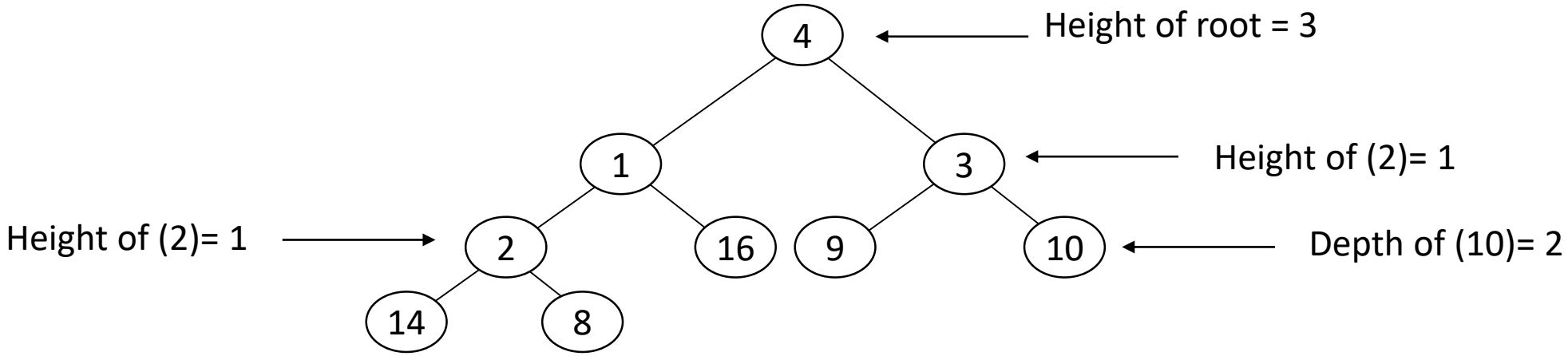


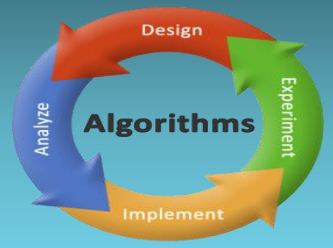
leftChild(i) = 2*I
rightChild(i) = 2*i+1
Parent (i) = [i/2]



ارتفاع و عمق در هرم

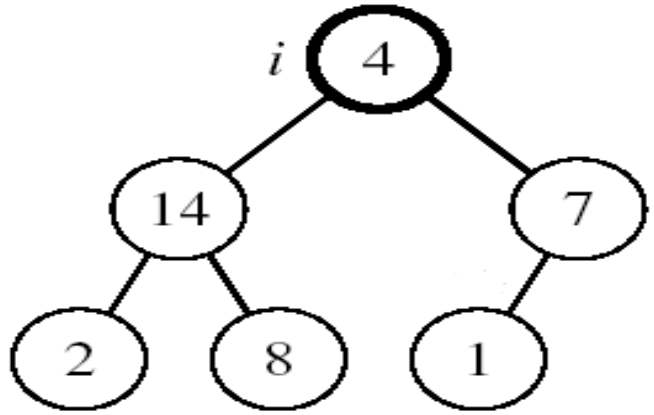
- **height** of a node = the number of edges on the longest simple path from the node down to a leaf
- **Depth** of a node = the length of a path from the root to the node
- **Height** of tree = height of root node = **Depth** of the tree

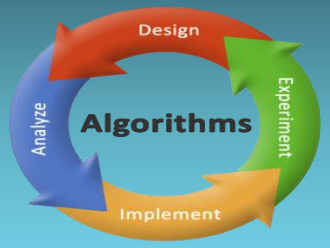




Maintaining the Heap Property

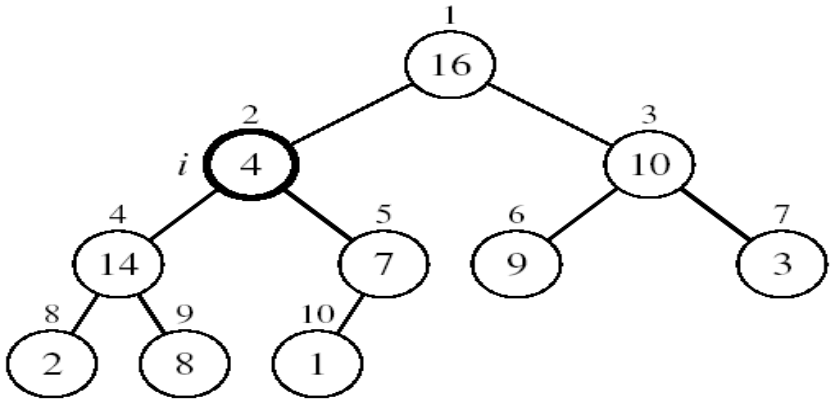
- Suppose a node is smaller than a child
  - ✓ Left and Right subtrees of i are max-heaps
- To eliminate the violation:
  - ✓ Exchange with larger child
  - ✓ Move down the tree
  - ✓ Continue until node is not smaller than children





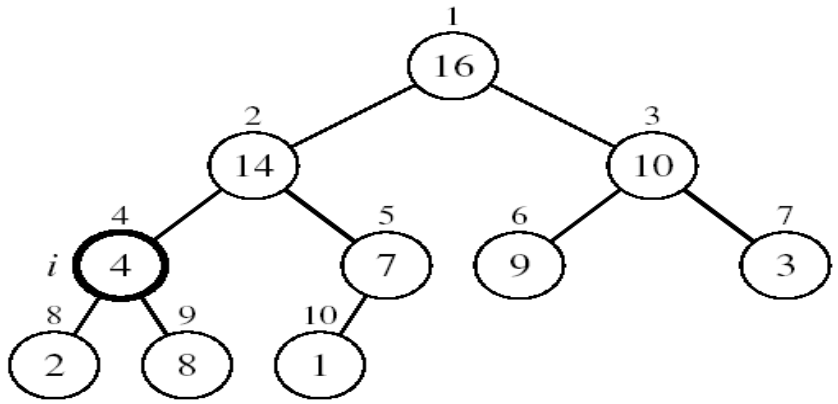
Example

MAX-HEAPIFY(A, 2, 10)



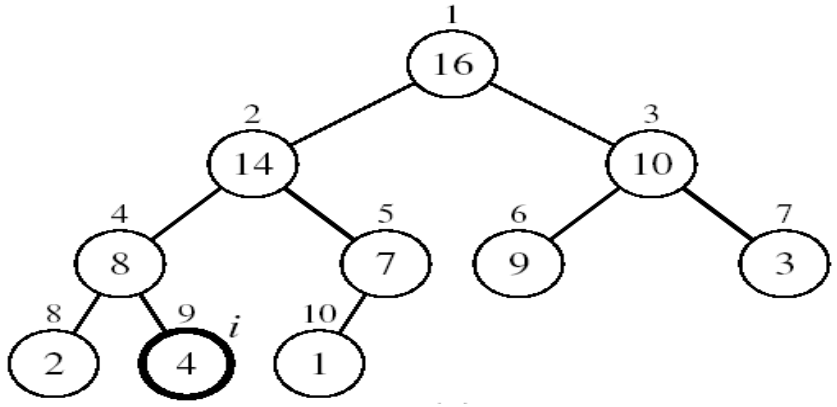
A[2] violates the heap property

A[2] ↔ A[4]

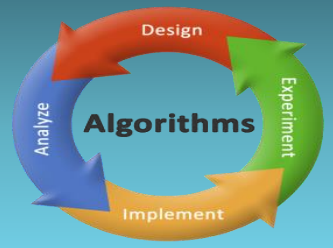


A[4] violates the heap property

A[4] ↔ A[9]

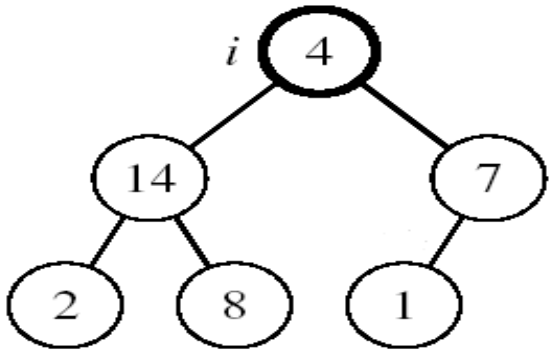


Heap property restored



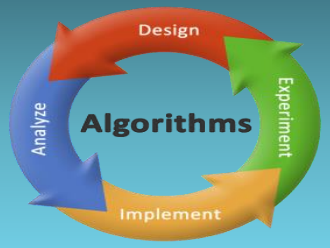
Maintaining the Heap Property

- Assumptions:
  - ✓ Left and Right subtrees of i are max-heaps
  - ✓ A[i] may be smaller than its children



Alg: MAX-HEAPIFY(A, i, n)

1. l ← LEFT(i)
2. r ← RIGHT(i)
3. **if** l ≤ n and A[l] > A[i]
4. **then** largest ←l
5. **else** largest ←i
6. **if** r ≤ n and A[r] > A[largest]
7. **then** largest ←r
8. **if** largest ≠ i
9. **then** exchange A[i] ↔ A[largest]
10. MAX-HEAPIFY(A, largest, n)

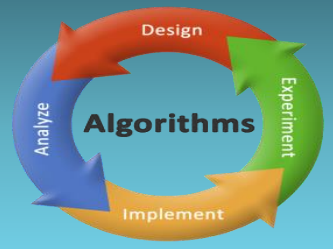


MAX-HEAPIFY Running Time

- Intuitively:

  - It traces a path from the root to a leaf (longest path length: h l)
  - At each level, it makes exactly 2 comparisons
  - Total number of comparisons is 2h
  - Running time is O(h) or $O(lgn)$

- Running time of MAX-HEAPIFY is $O(lgn)$
- Can be written in terms of the height of the heap, as being $O(h)$
  - ✓ Since the height of the heap is $\lfloor lgn \rfloor$

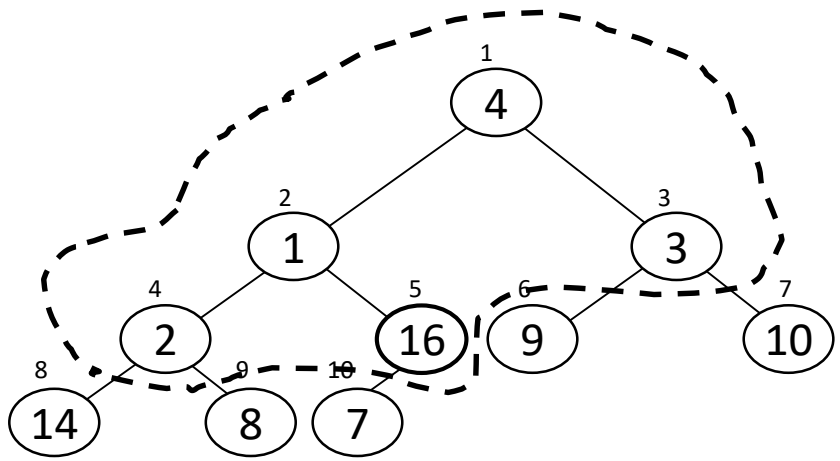


Building a Heap

- Convert an array $A[1 \ldots n]$ into a max-heap ($n = length[A]$)
- The elements in the subarray $A[(\lfloor n/2 \rfloor + 1) .. n]$ are leaves
- Apply MAX-HEAPIFY on elements between 1 and $\lfloor n/2 \rfloor$

Alg: BUILD-MAX-HEAP(A)

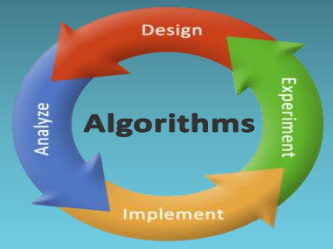
1. n = length[A]
2. **for** $i \leftarrow \lfloor n/2 \rfloor$ **downto** 1
3. **do** MAX-HEAPIFY(A, i, n)



A:

4	1	3	2	16	9	10	14	8	7

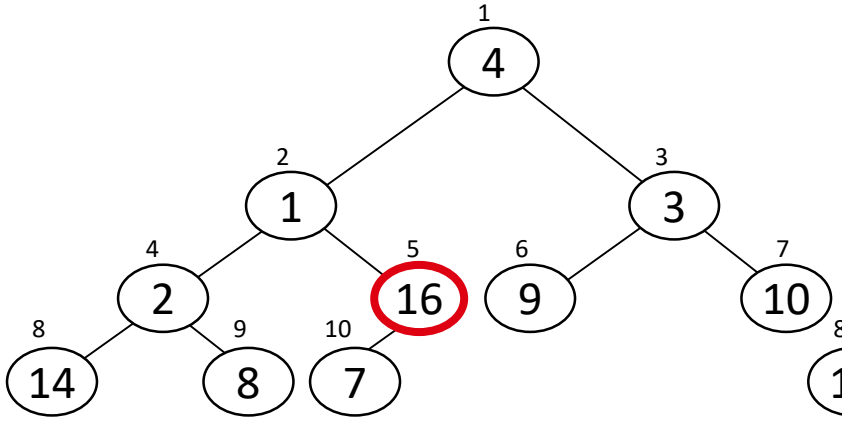
9



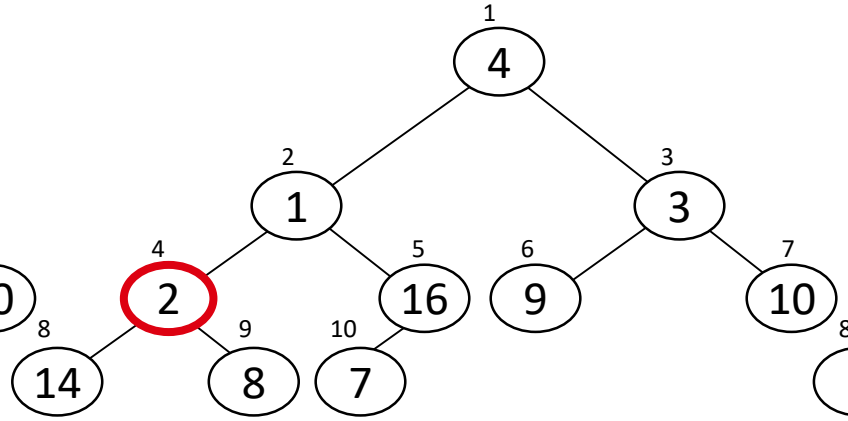
Example:

4	1	3	2	16	9	10	14	8	7

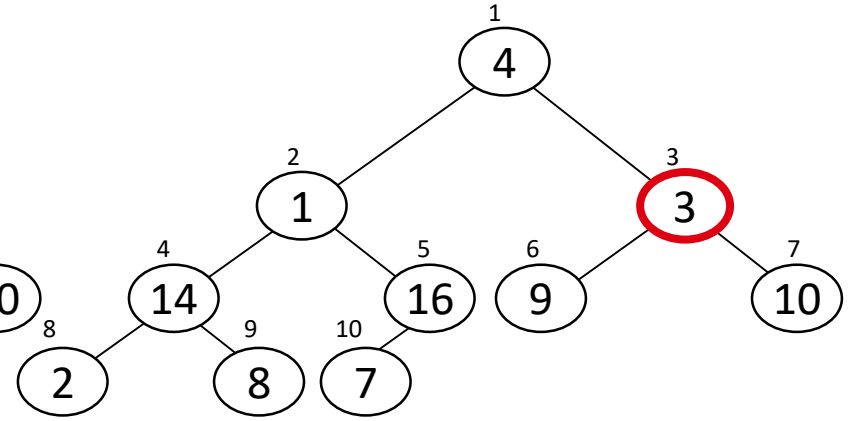
$i = 5$



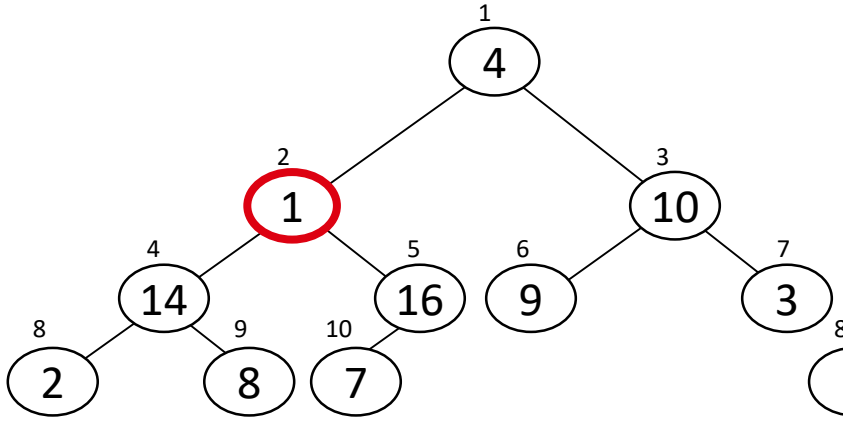
$i = 4$



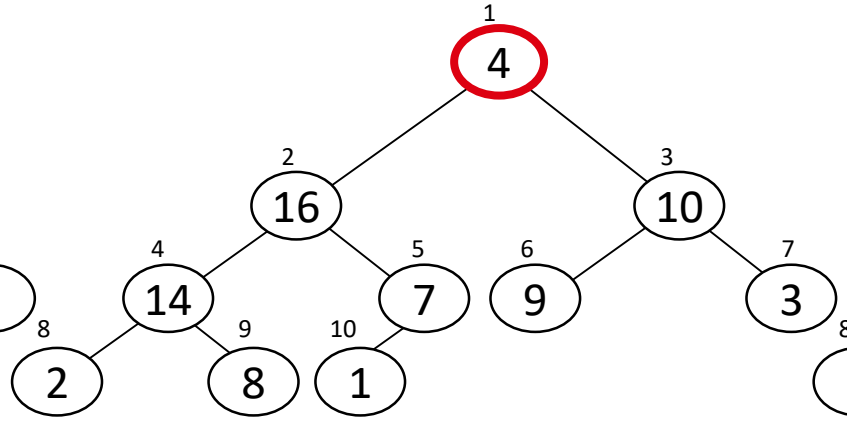
$i = 3$

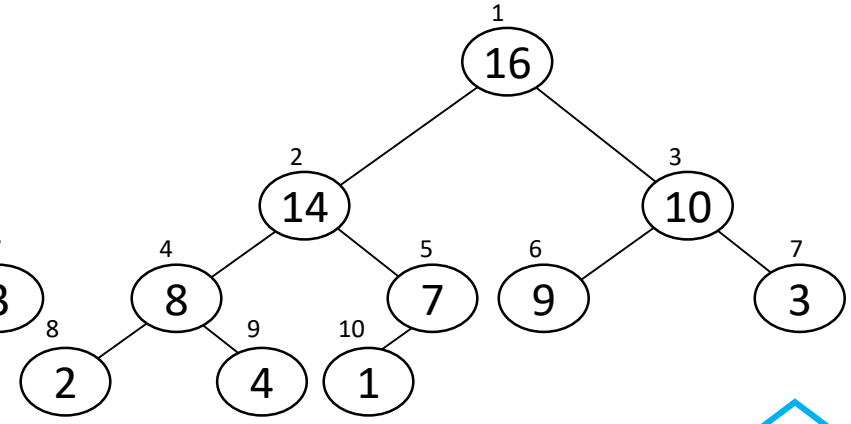


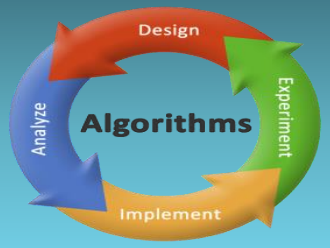
$i = 2$



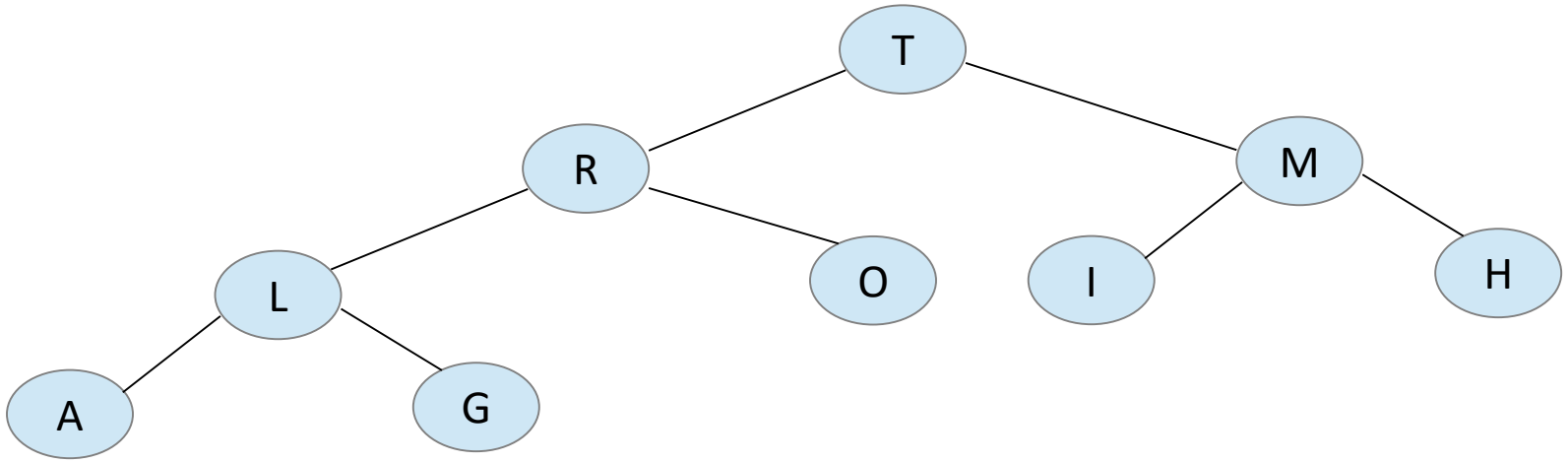
$i = 1$



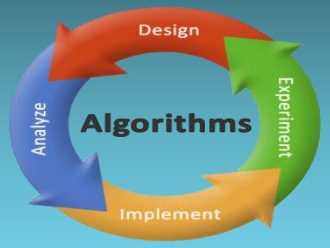




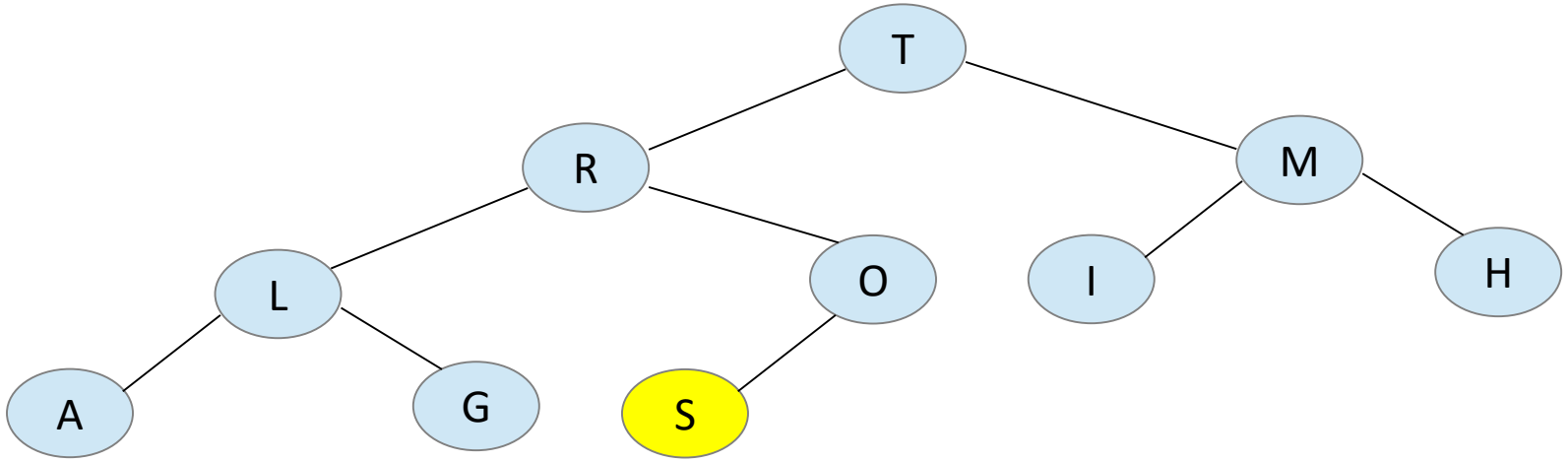
Another Example



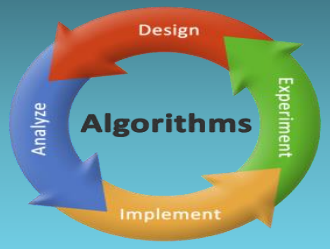
T	R	M	L	O	I	H	A	G
1	2	3	4	5	6	7	8	9



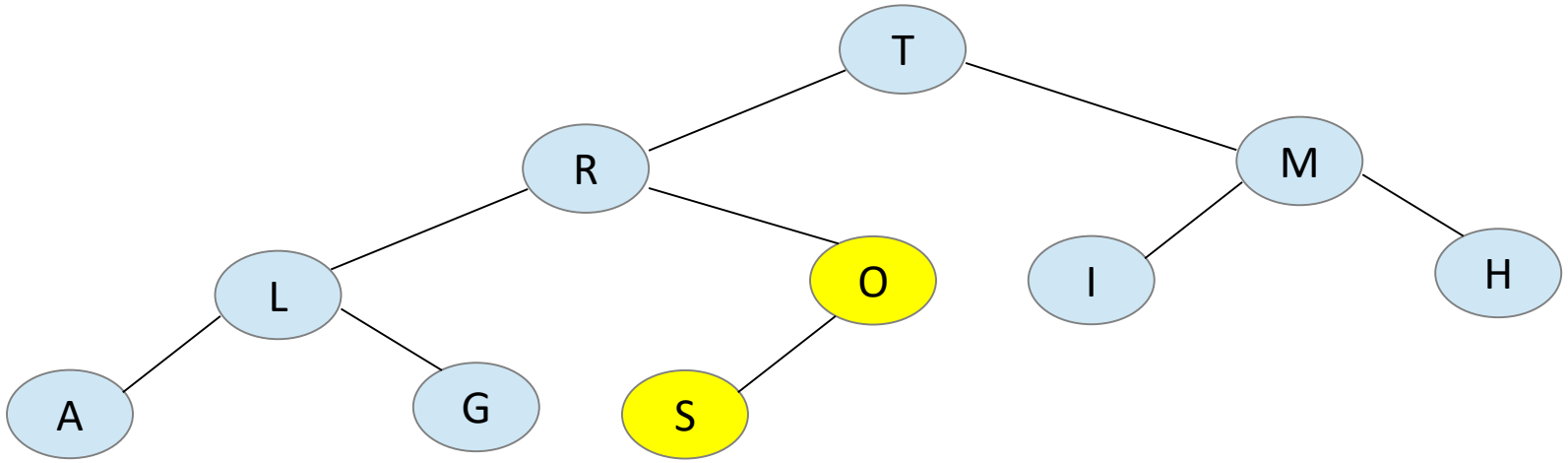
Insert: Heap - broken



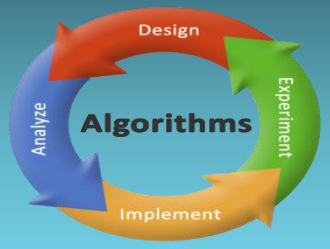
T	R	M	L	O	I	H	A	G	S
1	2	3	4	5	6	7	8	9	10



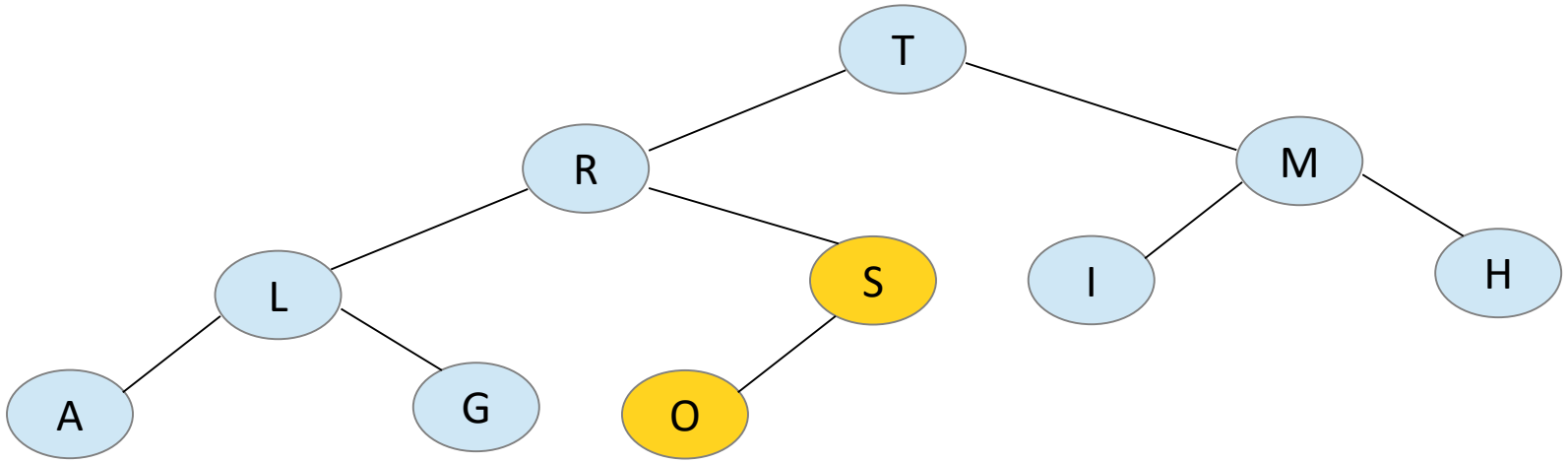
Heap - broken



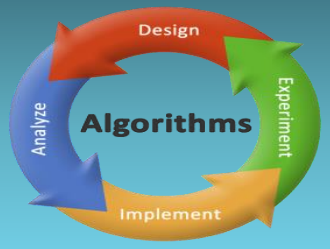
T	R	M	L	O	I	H	A	G	S
1	2	3	4	5	6	7	8	9	10



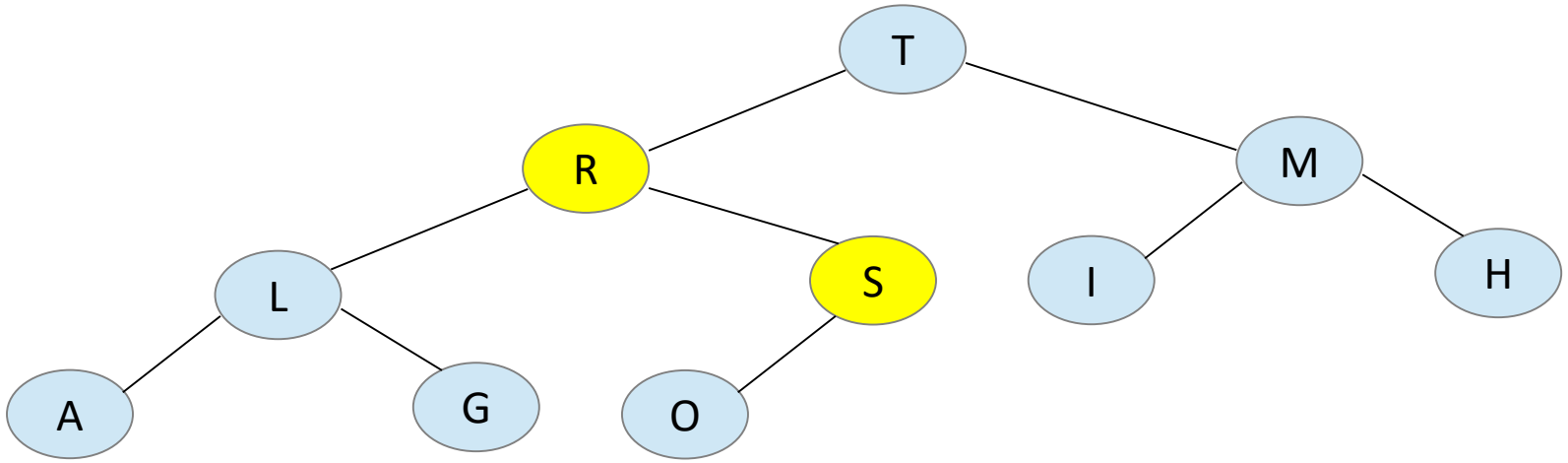
Heap - broken



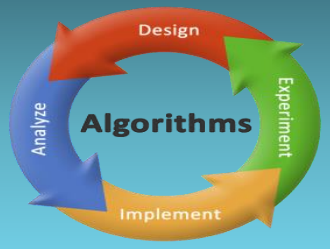
T	R	M	L	S	I	H	A	G	O
1	2	3	4	5	6	7	8	9	10



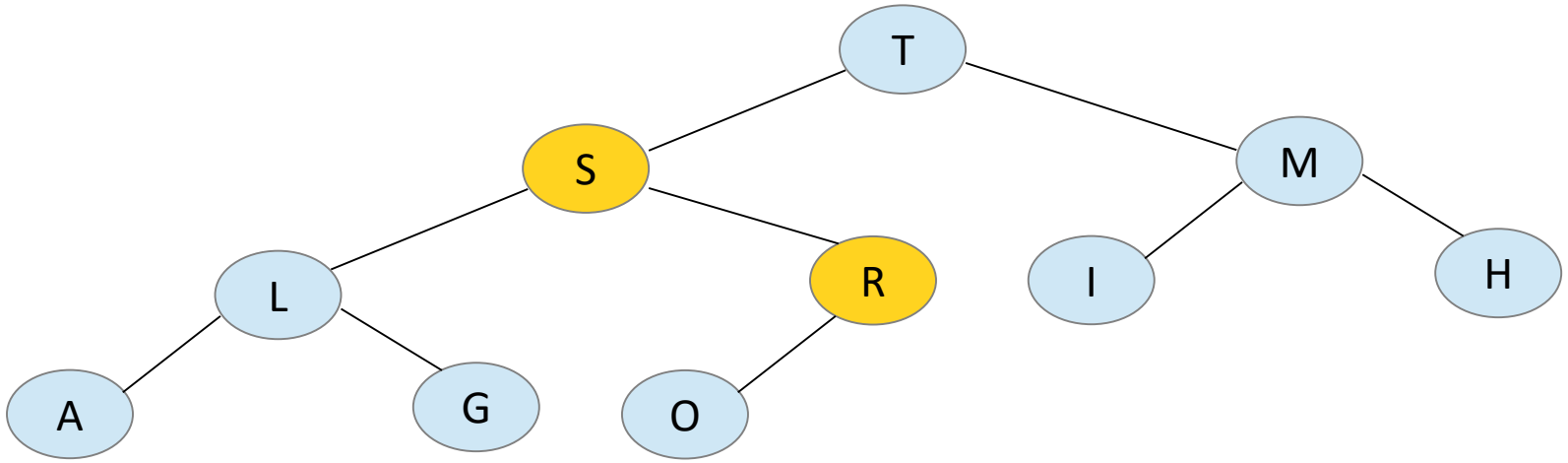
Heap - broken



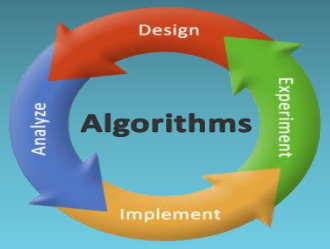
T	R	M	L	S	I	H	A	G	O
1	2	3	4	5	6	7	8	9	10



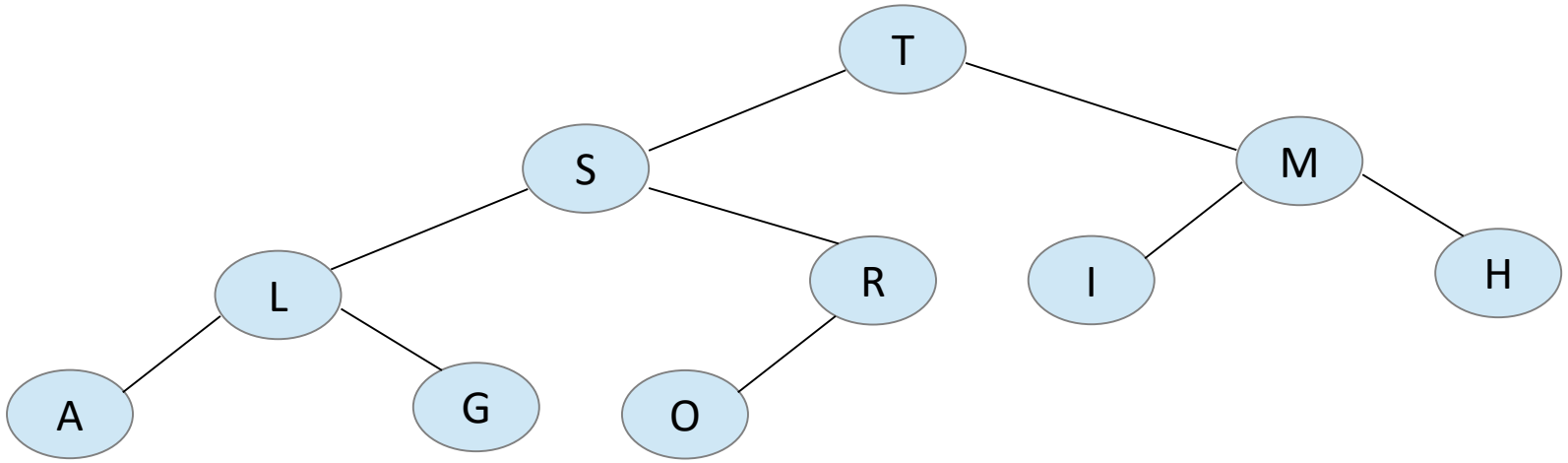
Heap - broken



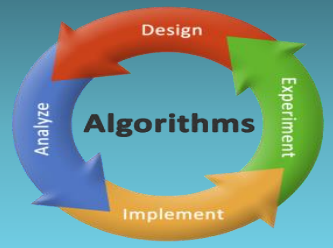
T	S	M	L	R	I	H	A	G	O
1	2	3	4	5	6	7	8	9	10



Heap - fixed



T	S	M	L	R	I	H	A	G	O
1	2	3	4	5	6	7	8	9	10



Running Time of BUILD MAX HEAP

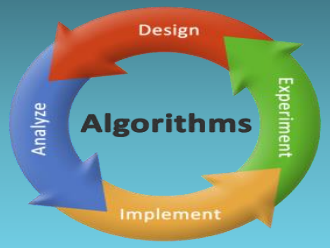
Alg: BUILD-MAX-HEAP(*A*)

1. n = length[A]
2. **for** i ← ⌊n/2⌋ **downto** 1
3. **do** MAX-HEAPIFY(*A*, i, n)

O(h): *O*(lgn)

O(n) ⇒ Running time: *O*(nlgn)

- این محکمترین حد نیست.
  - ✓ارتفاع تمام گره‌ها lgn نیست.



Running Time of BUILD MAX HEAP

✓ حد محکم از رابطه زیر قابل محاسبه است:

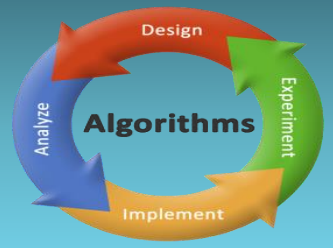
$$t(n) = \sum_{h=0}^{\lfloor lgn \rfloor} o(h) n_h$$

✓ می‌دانیم:

$$n_h \leq \left\lceil \frac{n}{2^{h+1}} \right\rceil$$

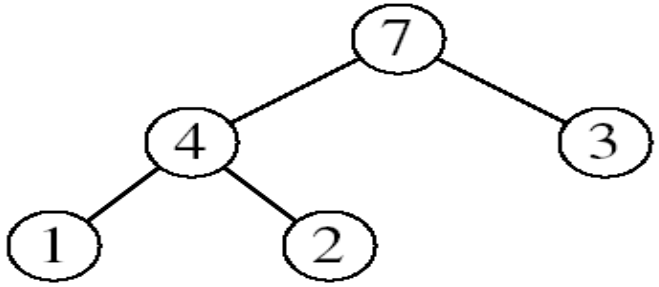
✓ پس:

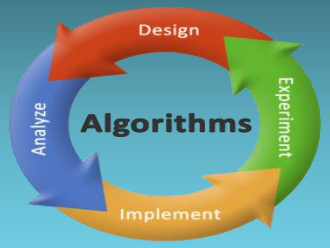
$$t(n) \leq \sum_{h=0}^{\lfloor lgn \rfloor} O(h) \left\lceil \frac{n}{2^{h+1}} \right\rceil = O\left(n \sum_{h=0}^{\lfloor lgn \rfloor} \frac{h}{2^h}\right) = O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right) \xrightarrow{\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}} O(n)$$



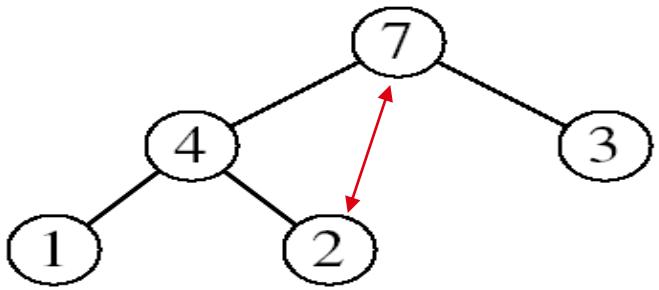
Heapsort

- Goal:
  - ✓ Sort an array using heap representations
- Idea:
  - ✓ Build a **max-heap** from the array
  - ✓ Swap the root (the maximum element) with the last element in the array
  - ✓ "Discard" this last node by decreasing the heap size
  - ✓ Call MAX-HEAPIFY on the new root
  - ✓ Repeat this process until only one node remains

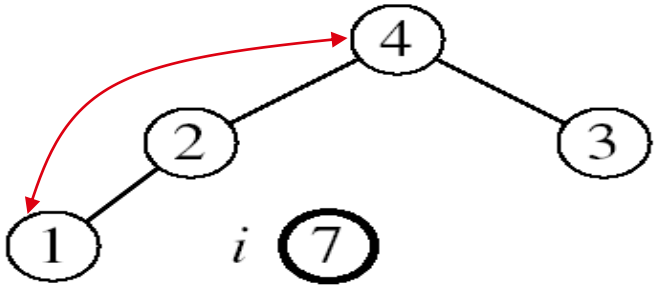




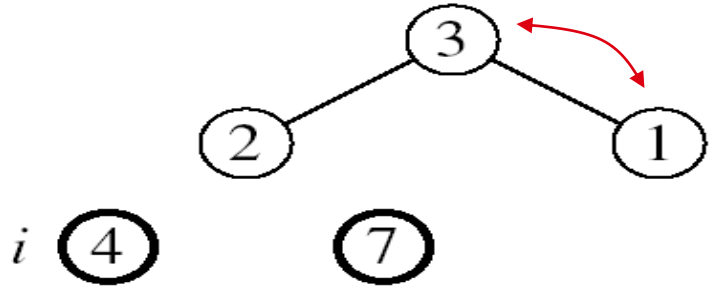
Example: A=[7, 4, 3, 1, 2]



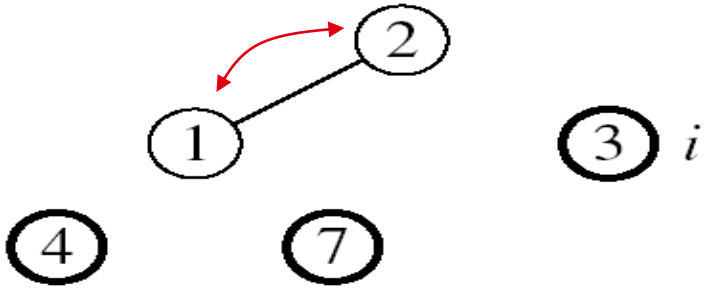
MAX-HEAPIFY(A, 1, 4)

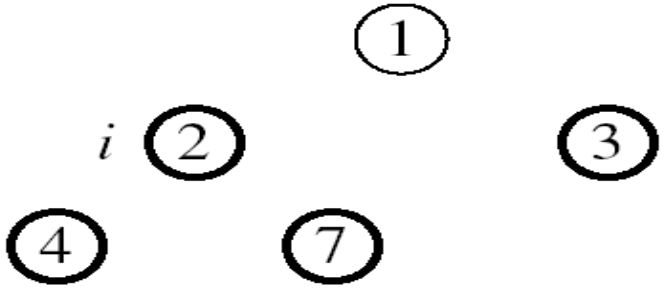


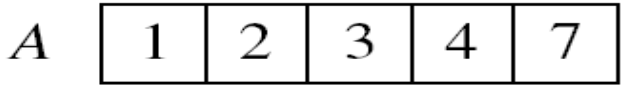
MAX-HEAPIFY(A, 1, 3)



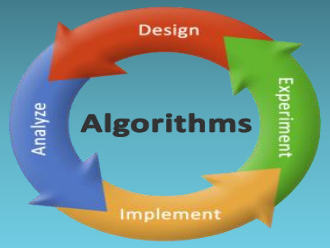
MAX-HEAPIFY(A, 1, 2)







MAX-HEAPIFY(A, 1, 1)

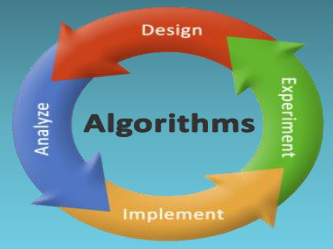


Alg: HEAPSORT*(A)*

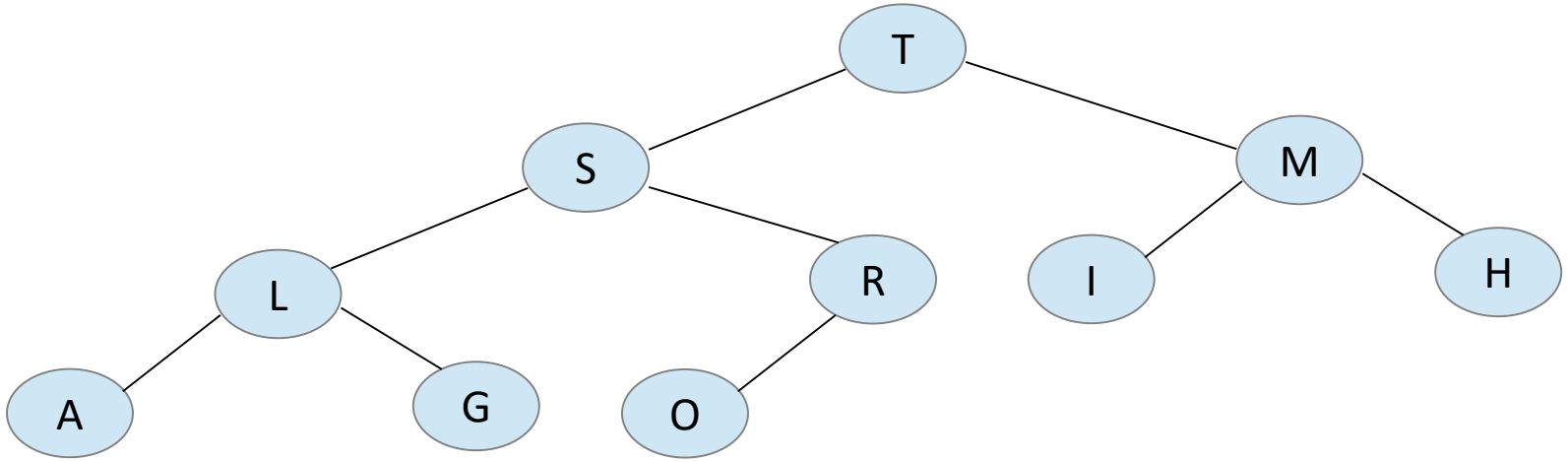
1. BUILD-MAX-HEAP(*A*) $O(n)$
2. **for** i ← length[*A*] **downto** 2
3. **do** exchange *A*[1] ↔ *A*[i]
4. MAX-HEAPIFY(*A*, 1, i - 1) $O(\lg n)$

} n-1 times

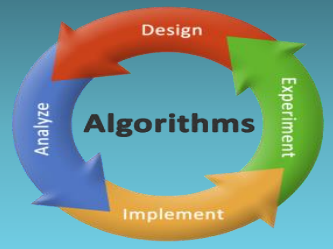
- Running time: $O(n \lg n)$ --- Can be shown to be $\Theta(n \lg n)$



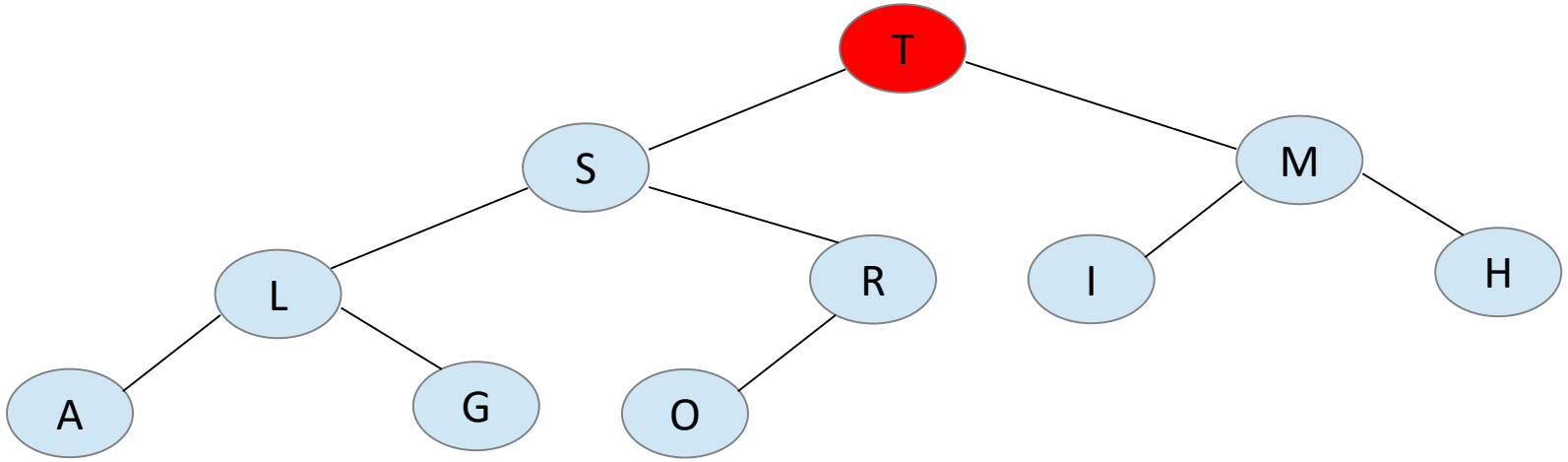
HeapSort – remove & replace root



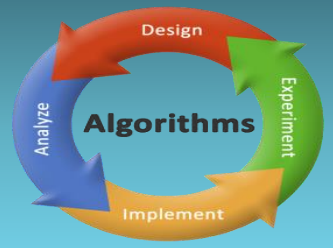
T	S	M	L	R	I	H	A	G	O
1	2	3	4	5	6	7	8	9	10



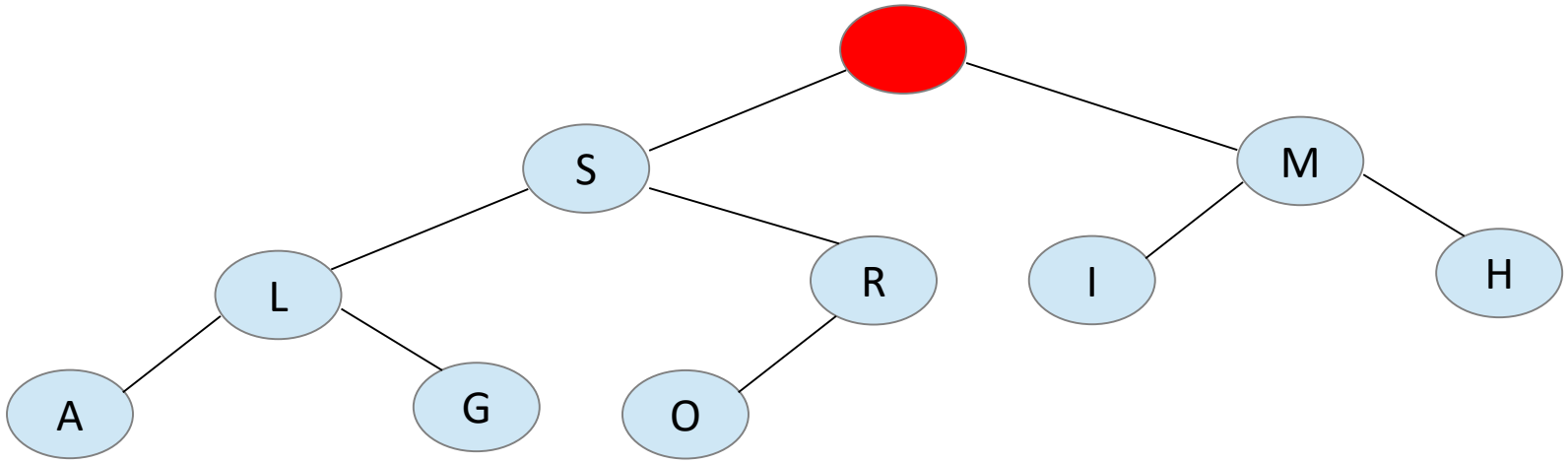
Heap – remove & replace root



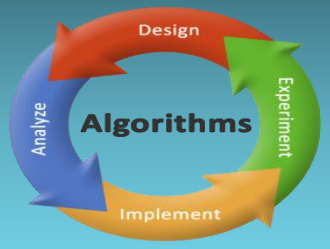
T	S	M	L	R	I	H	A	G	O
1	2	3	4	5	6	7	8	9	10



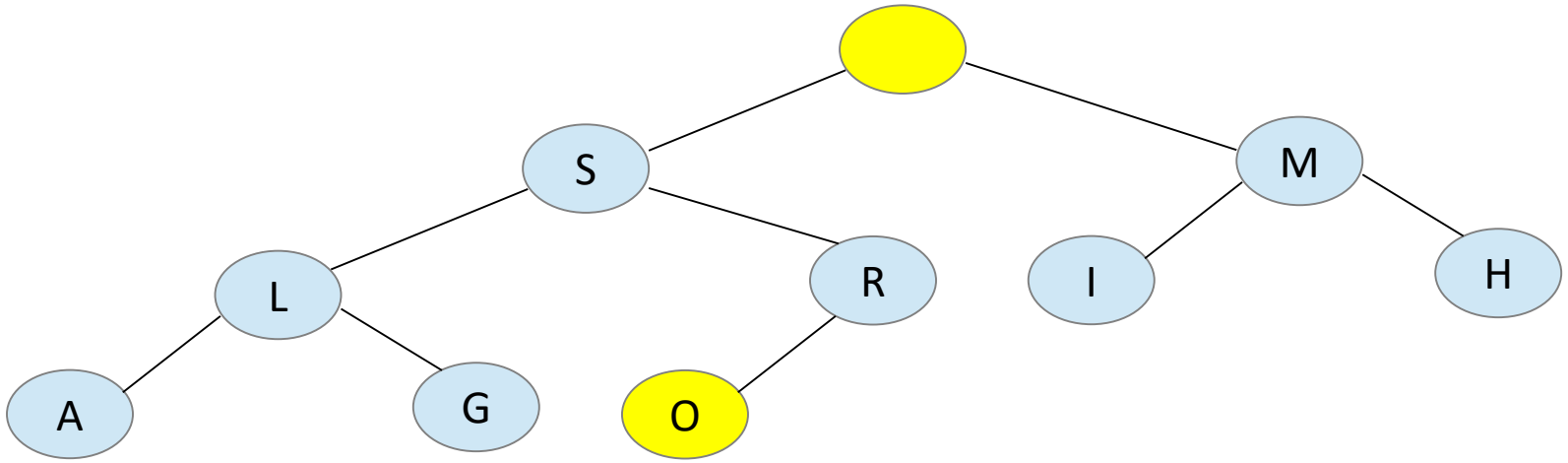
Heap – remove & replace root



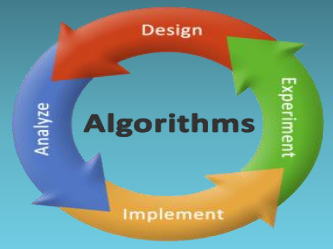
	S	M	L	R	I	H	A	G	O
1	2	3	4	5	6	7	8	9	10



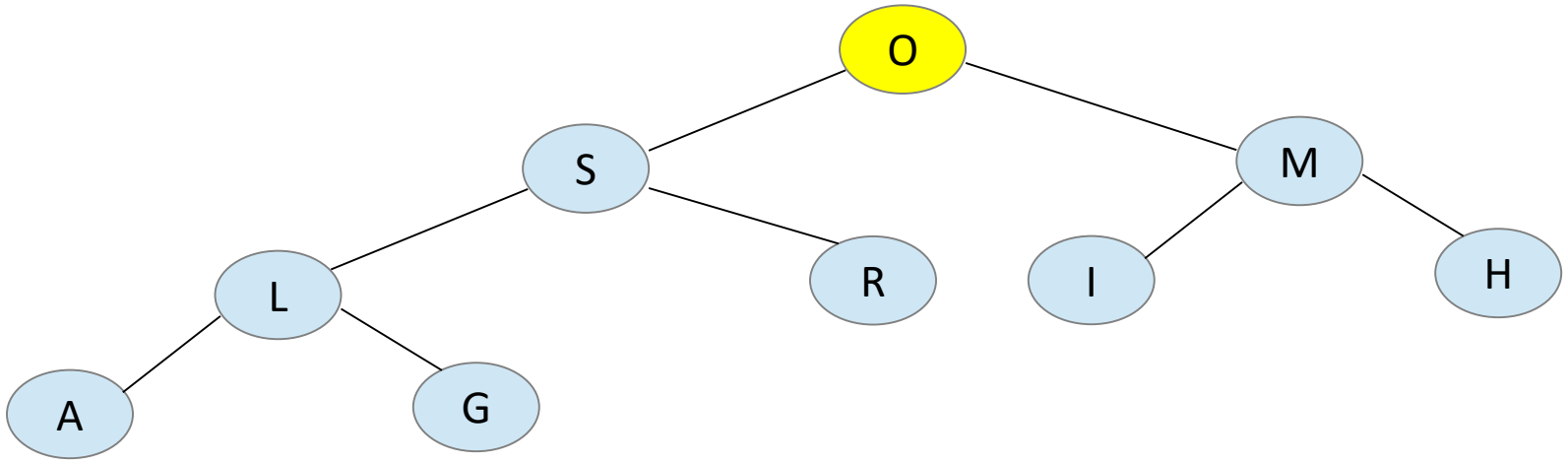
Heap – remove & replace root



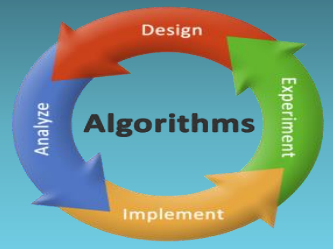
	S	M	L	R	I	H	A	G	O
1	2	3	4	5	6	7	8	9	10



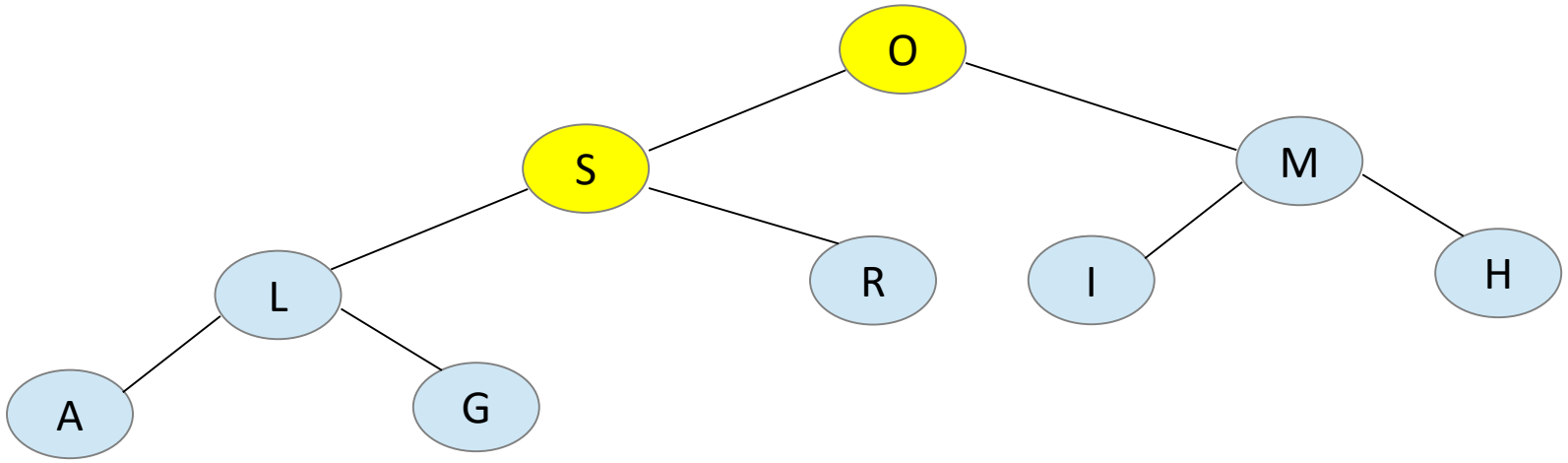
Heap – broken



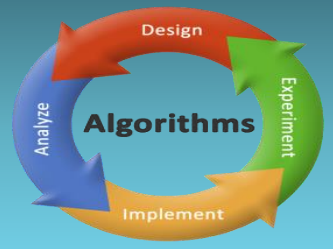
O	S	M	L	R	I	H	A	G
1	2	3	4	5	6	7	8	9



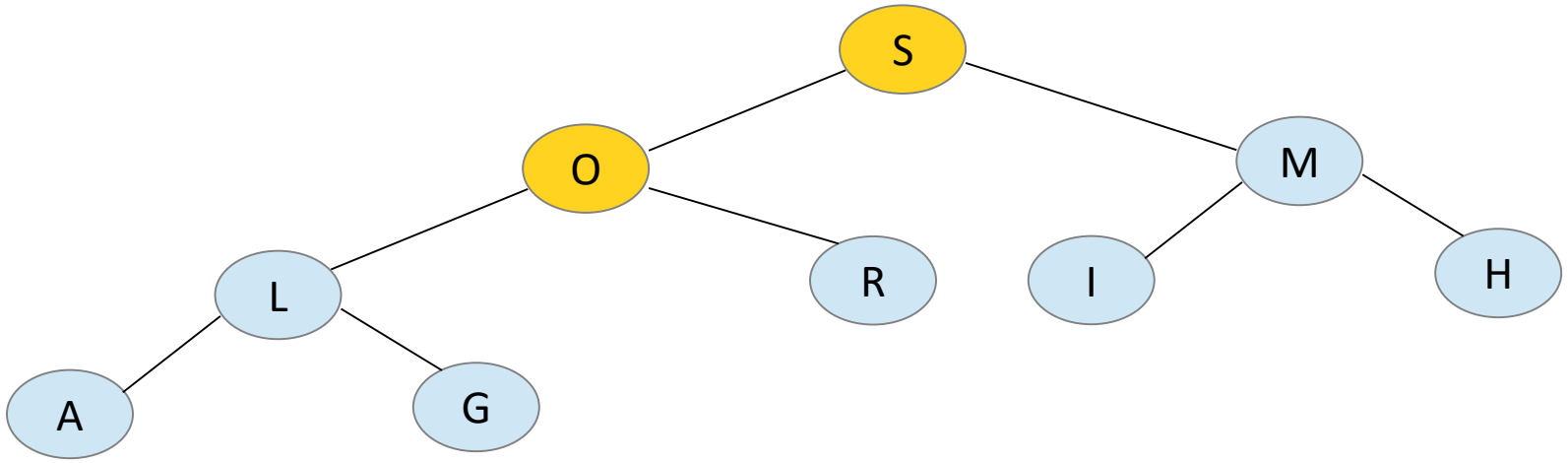
Heap – broken



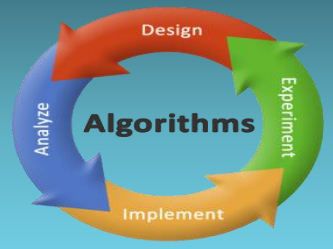
O	S	M	L	R	I	H	A	G
1	2	3	4	5	6	7	8	9



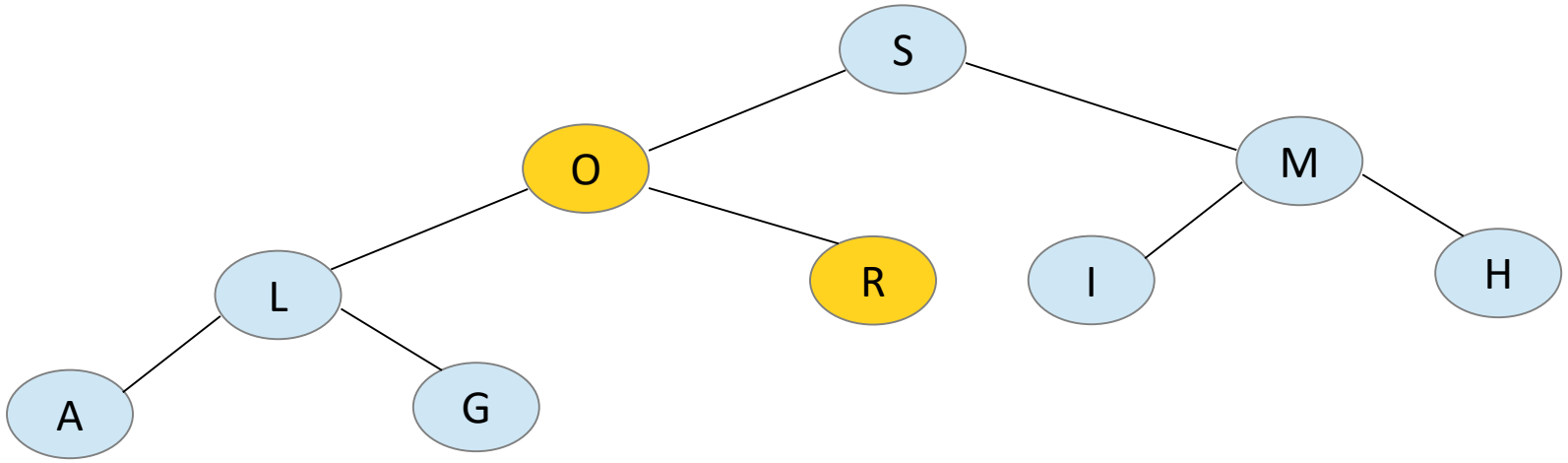
Heap – broken



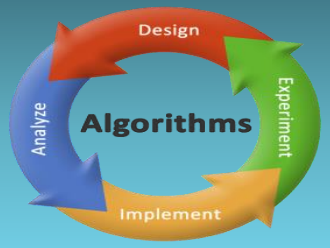
S	O	M	L	R	I	H	A	G
1	2	3	4	5	6	7	8	9



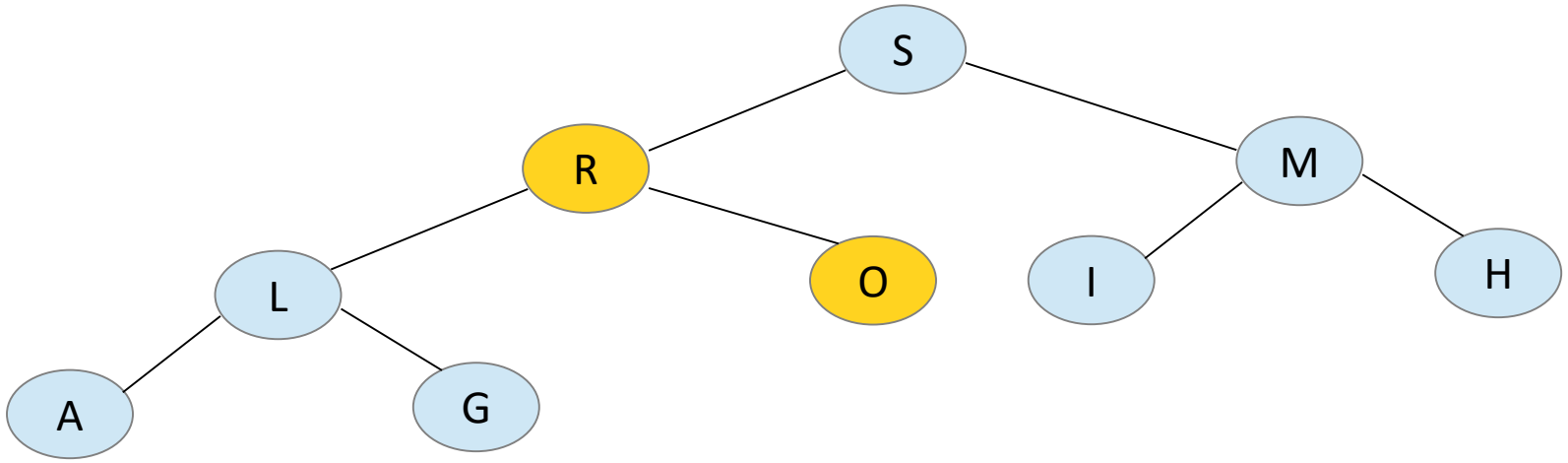
Heap – broken



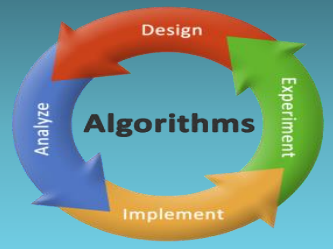
S	O	M	L	R	I	H	A	G
1	2	3	4	5	6	7	8	9



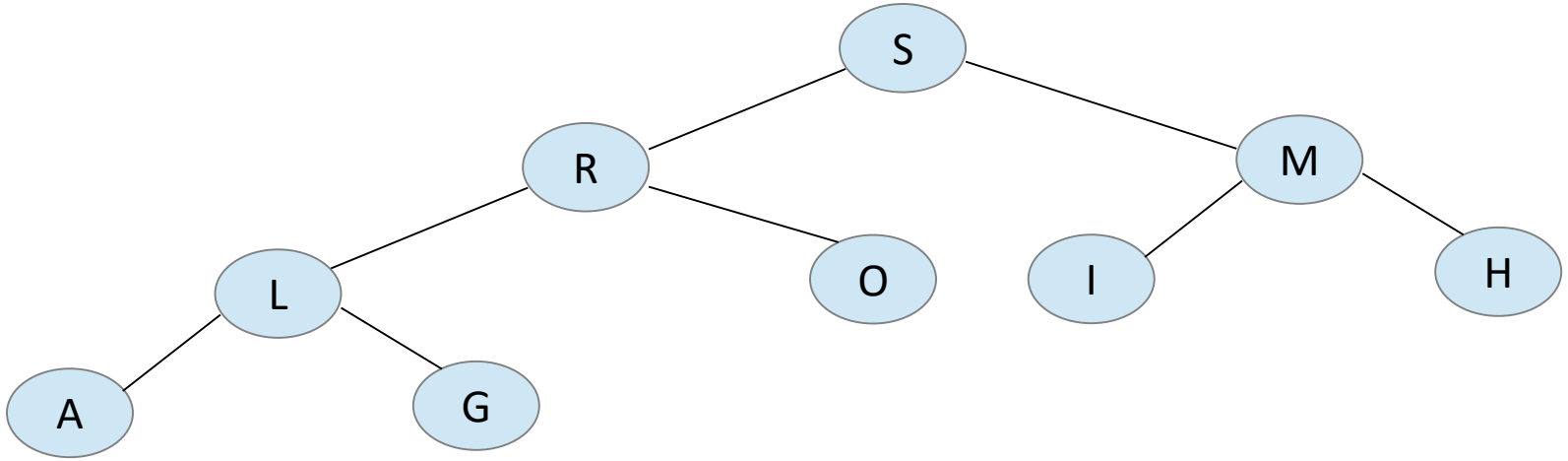
Heap – broken



S	R	M	L	O	I	H	A	G
1	2	3	4	5	6	7	8	9



Heap – fixed



S	R	M	L	O	I	H	A	G
1	2	3	4	5	6	7	8	9