Pop Quiz

2. Show that an n-element heap has height (lg n)

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

- Since the height of an n-element heap must satisfy that $2^h \le n \le 2^{h+1}-1 < 2^{h+1}$.
- We have $h \le \lg n < h+1$.
- h is an integer so $h = \lg n$.

Pop Quiz

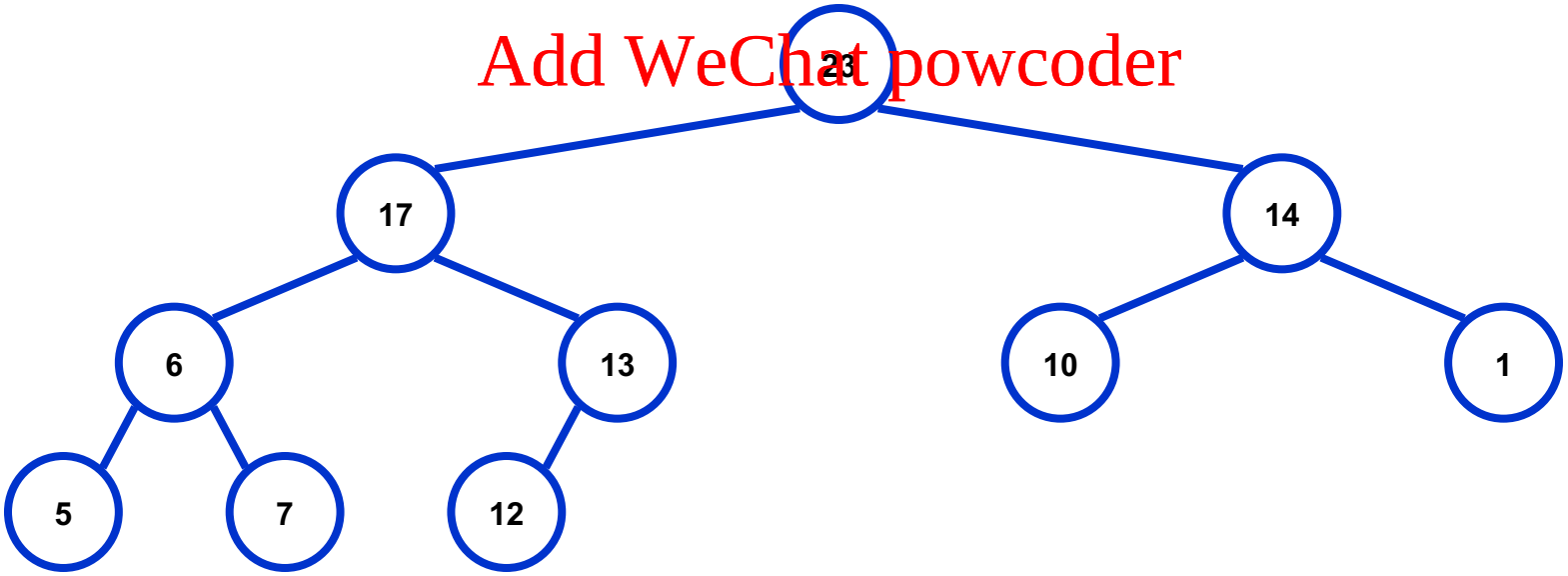
3. Is the array with values
   [23,17,14,6,13,10,1,5,7,12] a Max-Heap?

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

- No



Basic procedures on heap

- **`Max-Heapify`**
- **`Build-Max-Heap`**
- **`Heapsort`**
- **`Max-Heap-Insert`**
- **`Heap-Extract-Max`**
- **`Heap-Increase-Key`**
- **`Heap-Maximum`**

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

6.2 Maintaining the heap property

`Max-Heapify(A, i)`

Assignment Project Exam Help

- To maintain the max-heap property.

https://powcoder.com

- Assume that the binary trees rooted at `Left(i)` and `Right(i)` are heaps, but that `A[i]` may be smaller than its children, thus violating the heap property.

Add WeChat powcoder

```
Max-Heapify(A,i)
 1 l = Left(i)
 2 r = Right(i)
 3 if l ≤ A.heap-size and A[l] > A[i]
 4       largest = l
 5 else largest = i
 6 if r ≤ A.heap-size and A[r] > A[largest]
 7       largest = r
 8 if largest ≠ i
 9       exchange A[i] ↔ A[largest]
10       Max-Heapify(A, largest)
```

Assumption:
Left(i) and Right(i) are max-heaps.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

$$T(n) \le T(\frac{2n}{3}) + \Theta(1) \Rightarrow T(n) = O(\lg n)$$

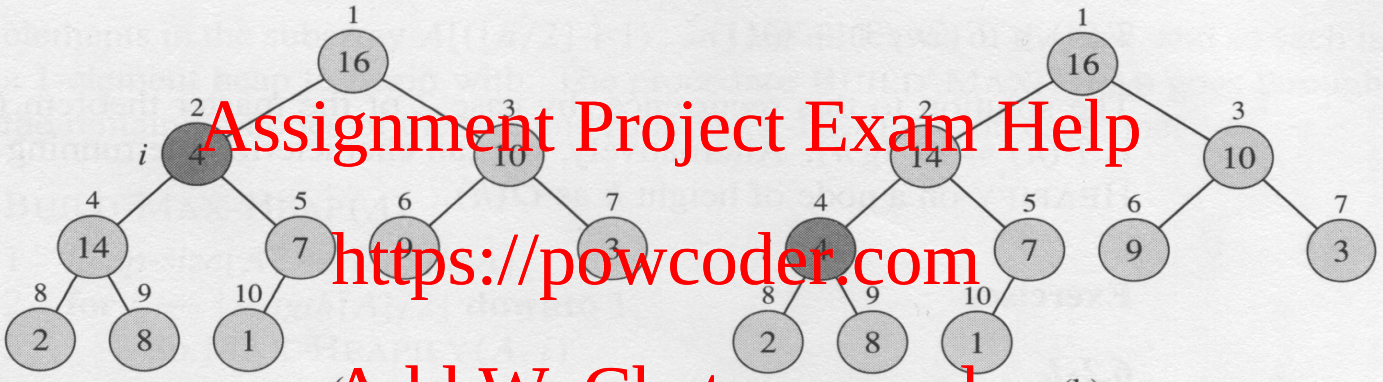
Alternatively O(h) (h: height)

Max-Heapify(A,2)
heap-size[A] = 10

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

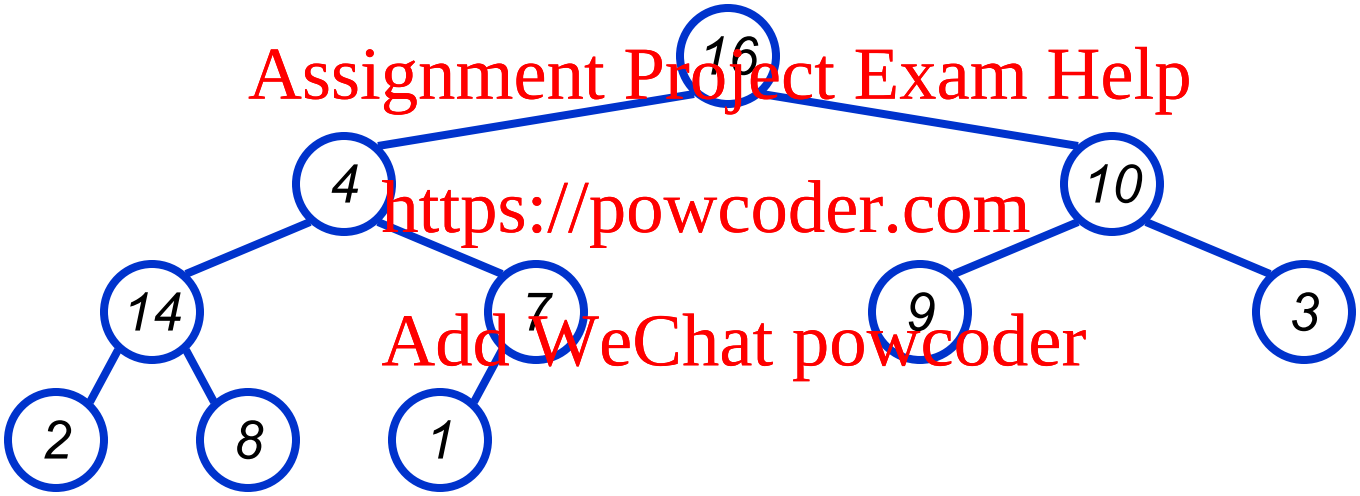


Max-Heapify(A,2)

-> Max-Heapify(A,4)

-> Max-Heapify(A,9)

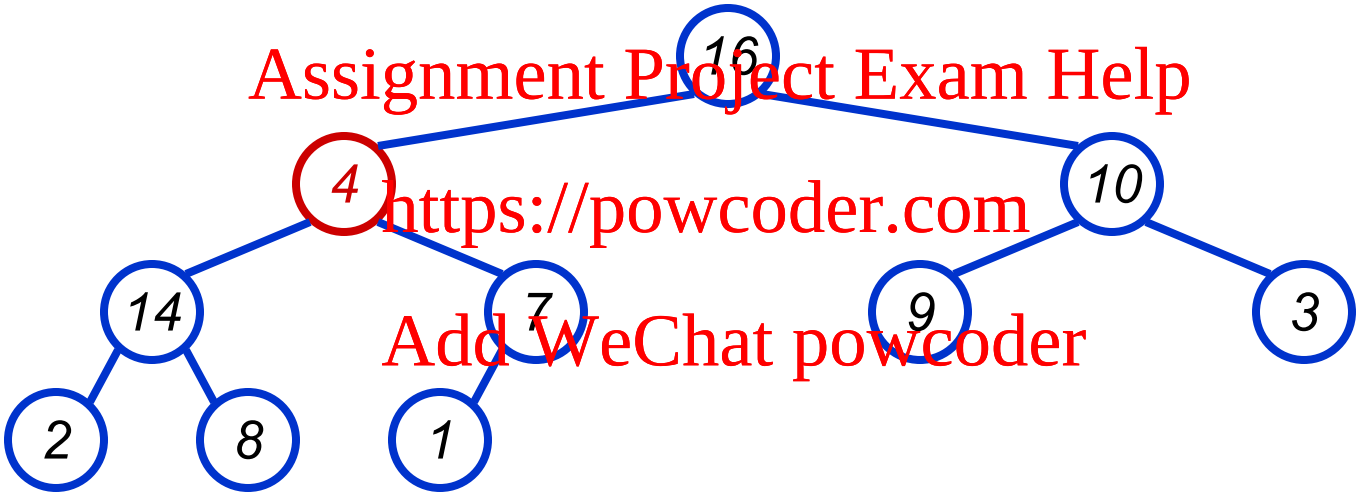
Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

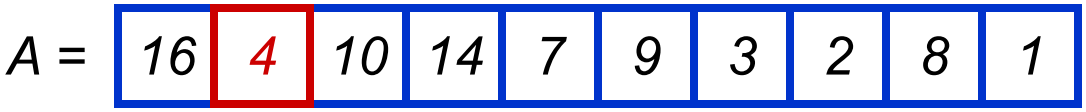


$A =$

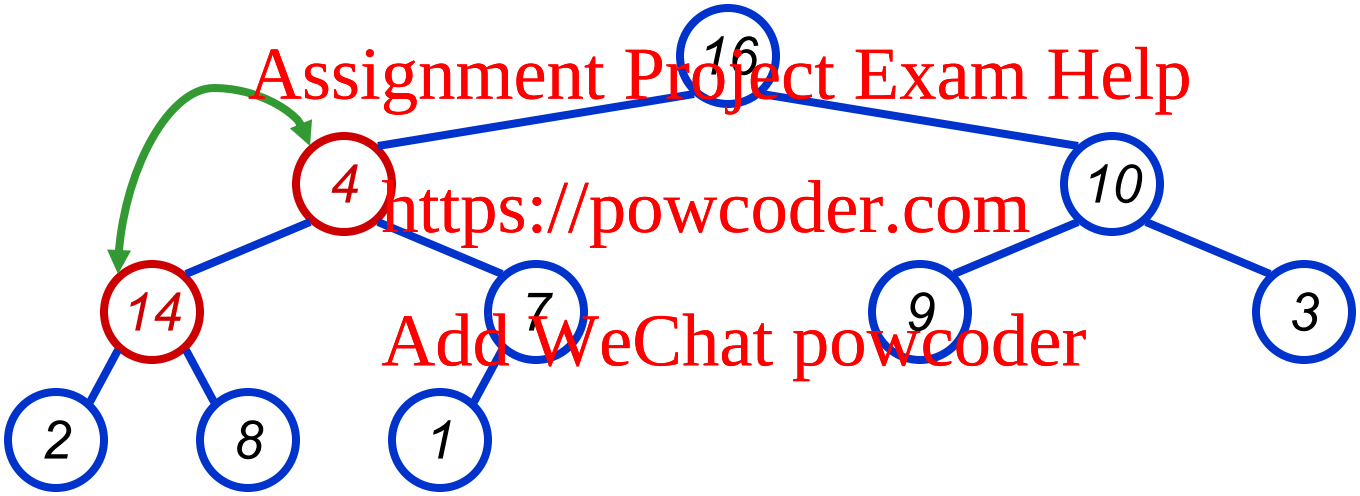
16	4	10	14	7	9	3	2	8	1

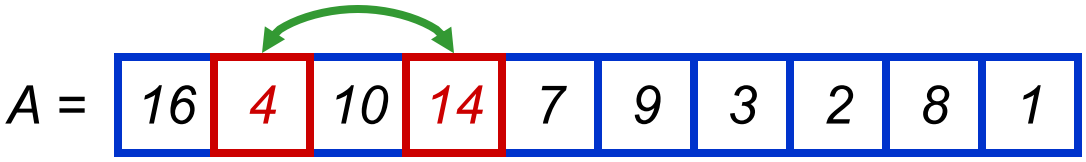
Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



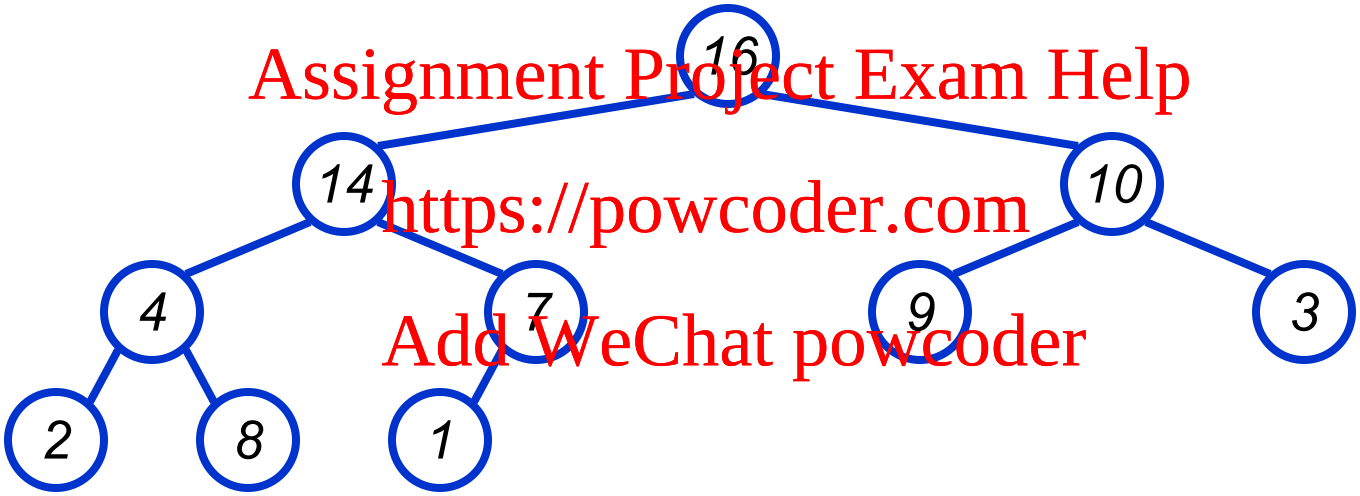


Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder





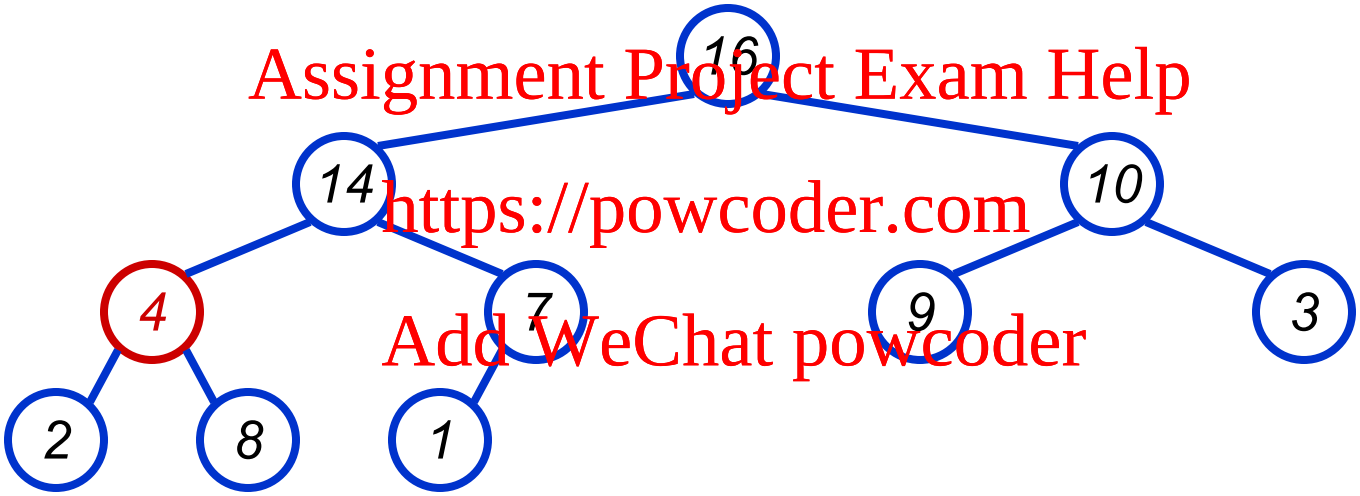
Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

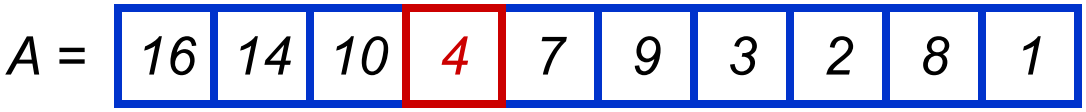


$A =$

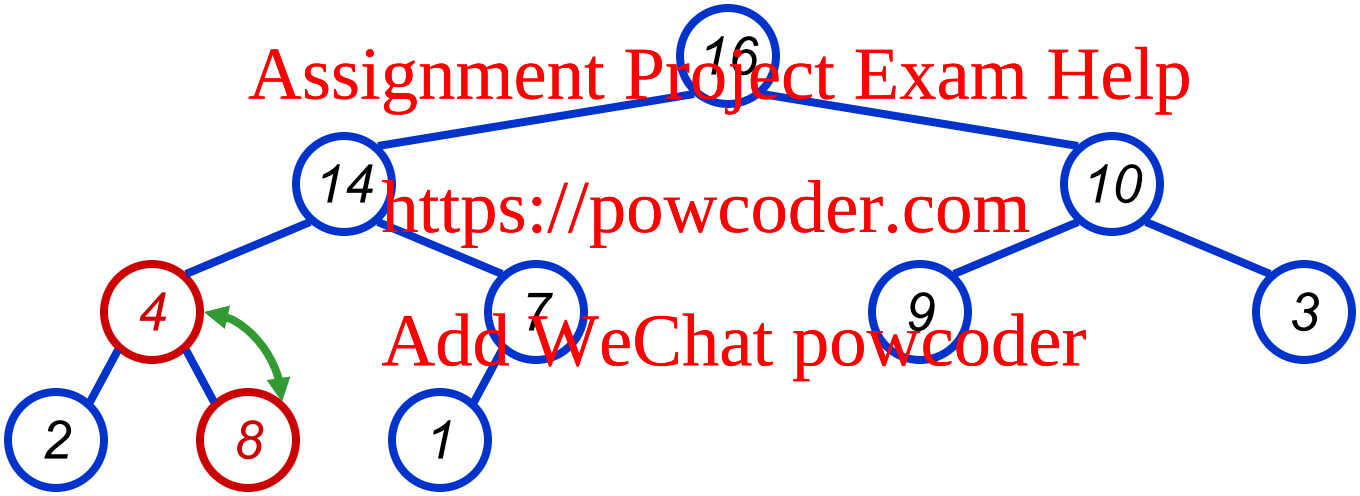
16	14	10	4	7	9	3	2	8	1

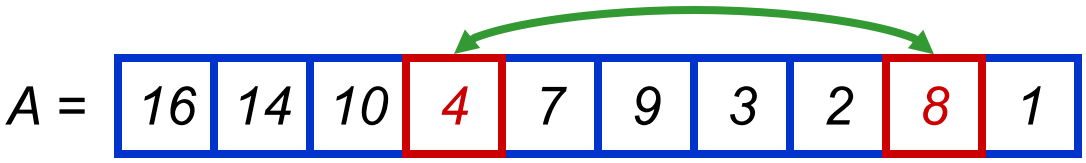
Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



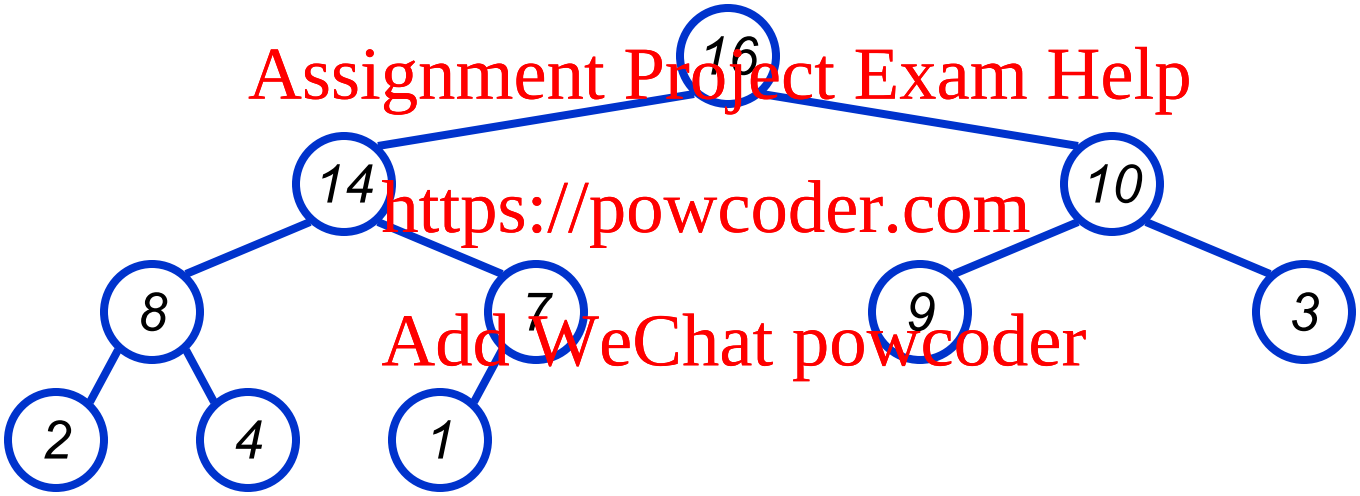


Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder





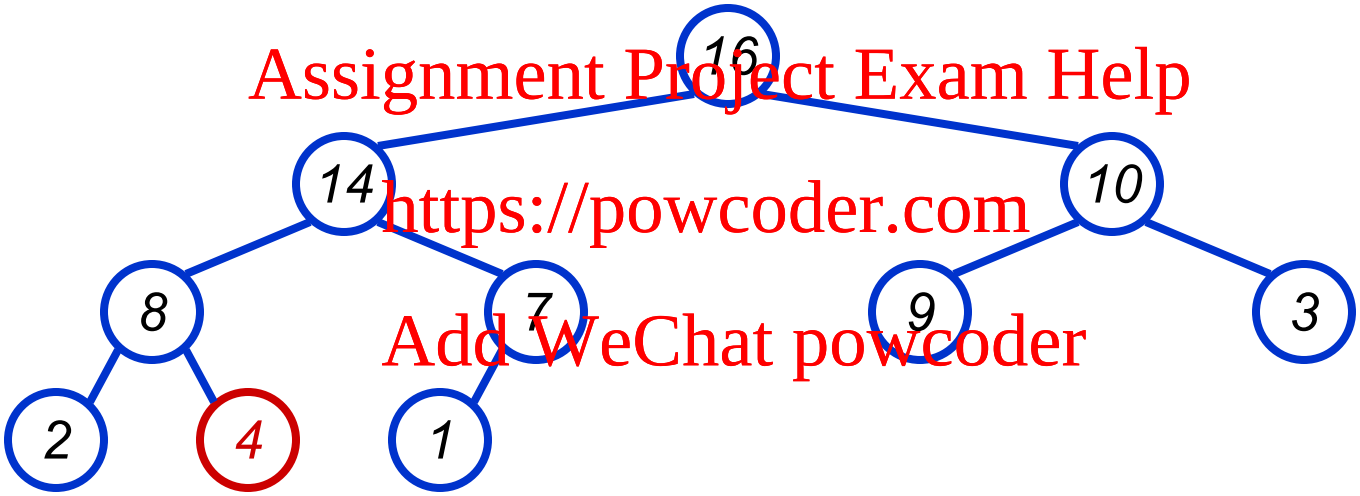
Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

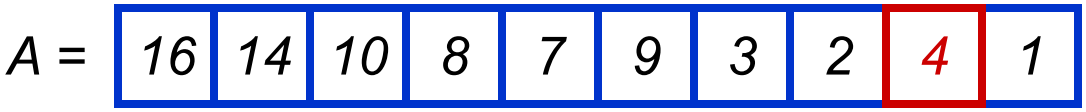


$A =$

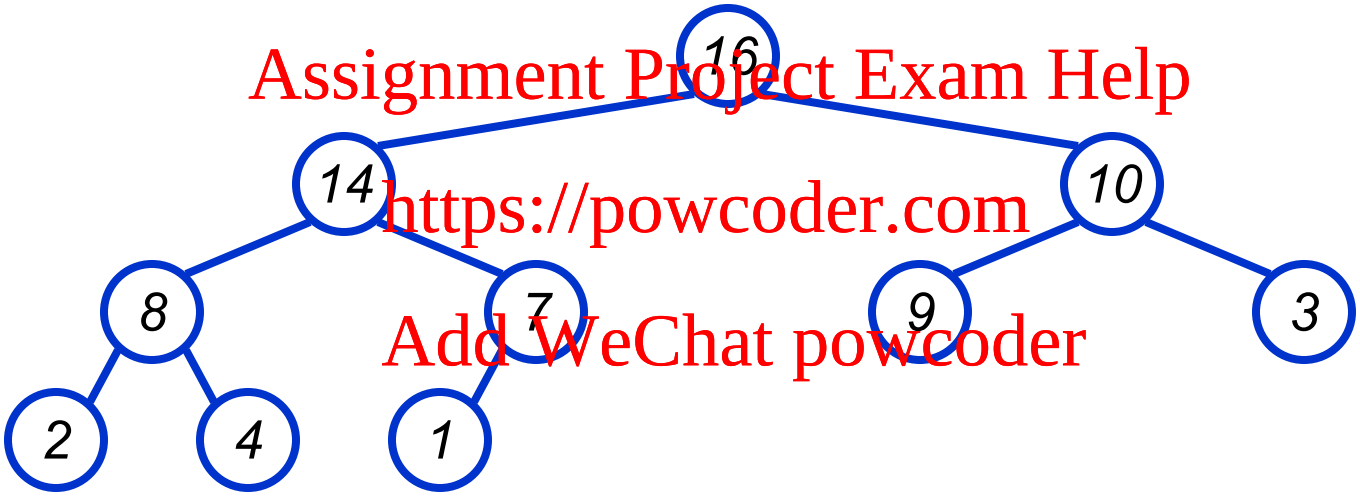
16	14	10	8	7	9	3	2	4	1

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder





Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



$A =$

16	14	10	8	7	9	3	2	4	1

Running Time for Max-Heapify

```
Max-Heapify(A,i)
 1 l = Left(i)
 2 r = Right(i)
 3 if l ≤ A.heap-size and A[l] > A[i]
 4         largest  = l
 5 else  largest = i
 6 if r ≤ A.heap-size and A[r] > A[largest]
 7         largest = r
 8 if largest ≠ i
 9       exchange A[i] ↔ A[largest]
10       Max-Heapify(A, largest)
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Time to fix node i and its children = ***Θ(1)***

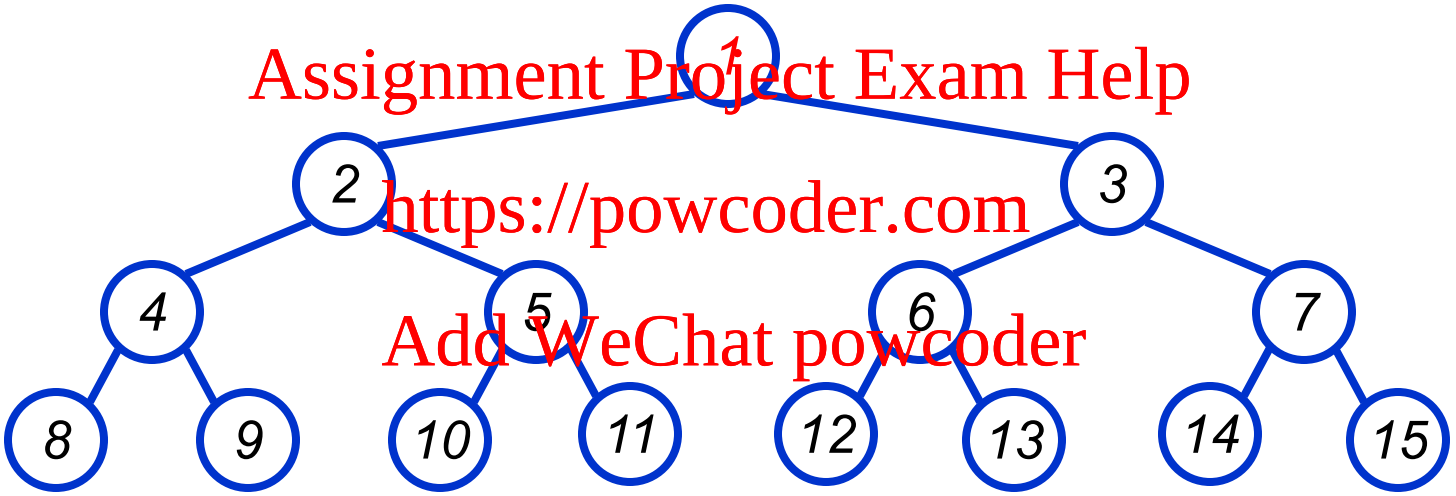
PLUS

Time to fix the subtree rooted at one of i's children = T(size of subree at largest)

Max-Heapify
-To make node *i* max heap

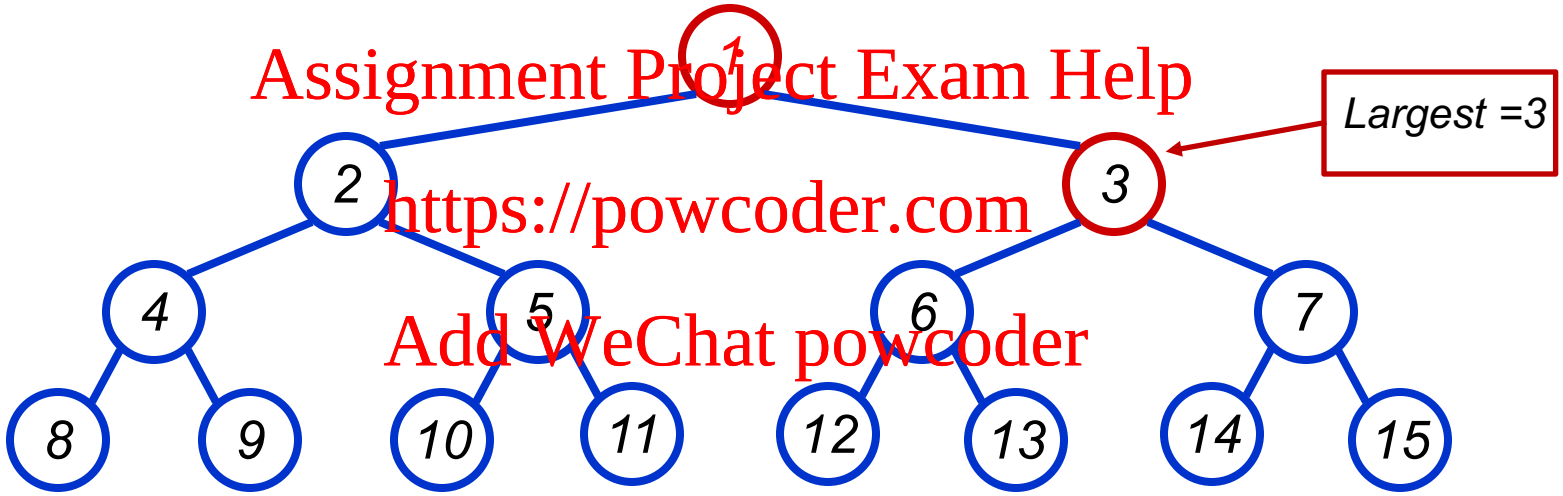
Max-Heapify(A,1)

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



A =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Max-Heapify(A,1)

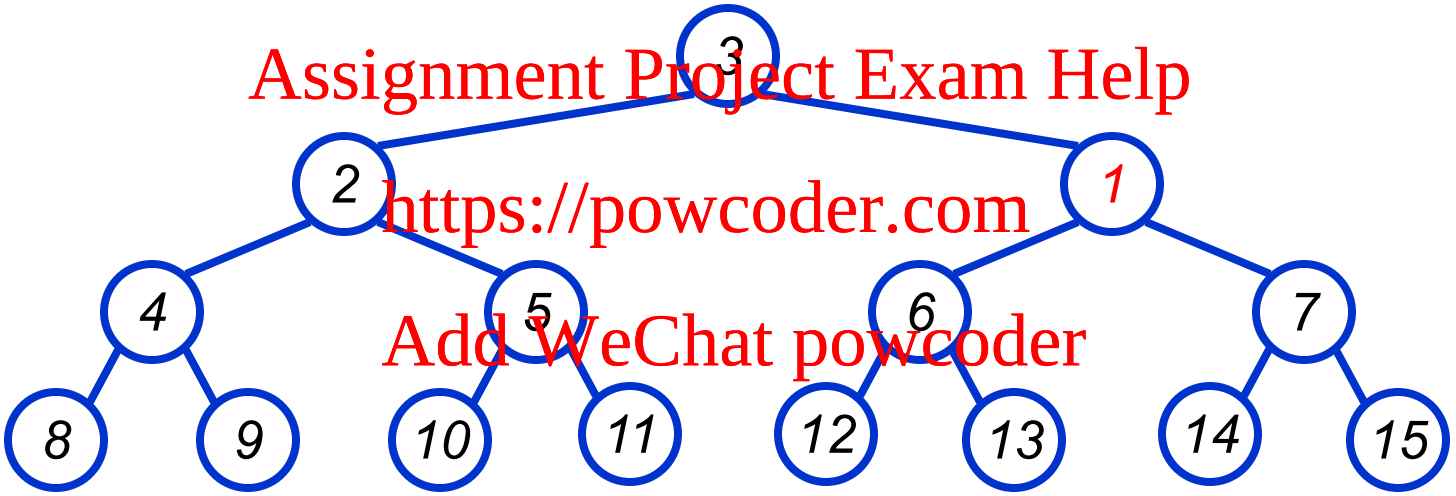


Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

A =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Max-Heapify(A,largest) = Max-Heapify(A,3)

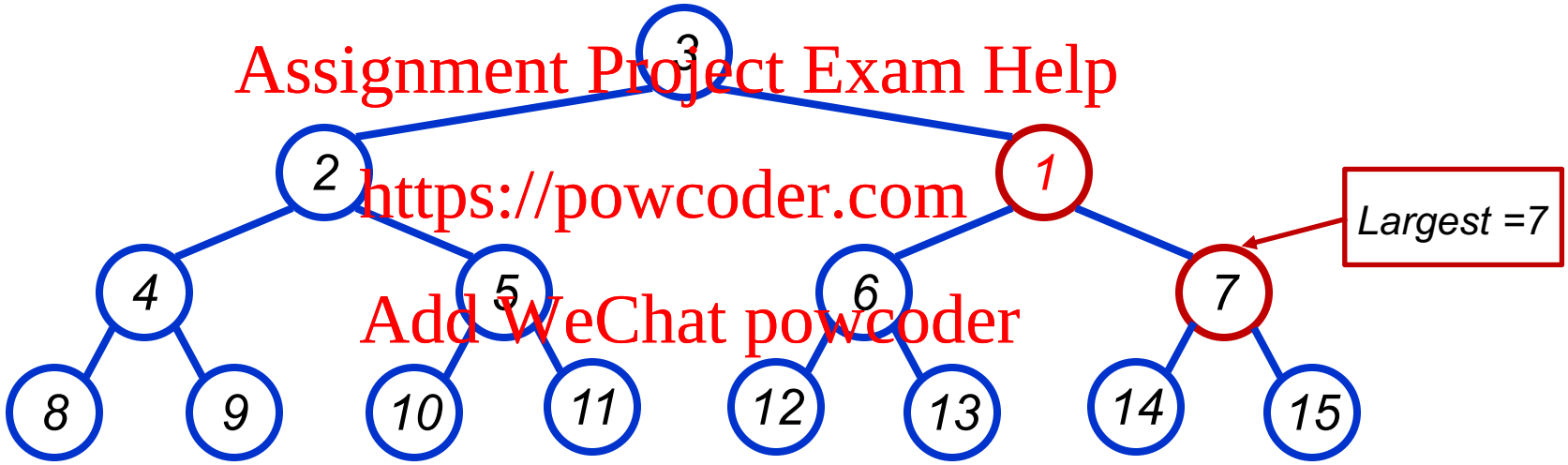
Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



A =	3	2	1	4	5	6	7	8	9	10	11	12	13	14	15

Max-Heapify(A,3)

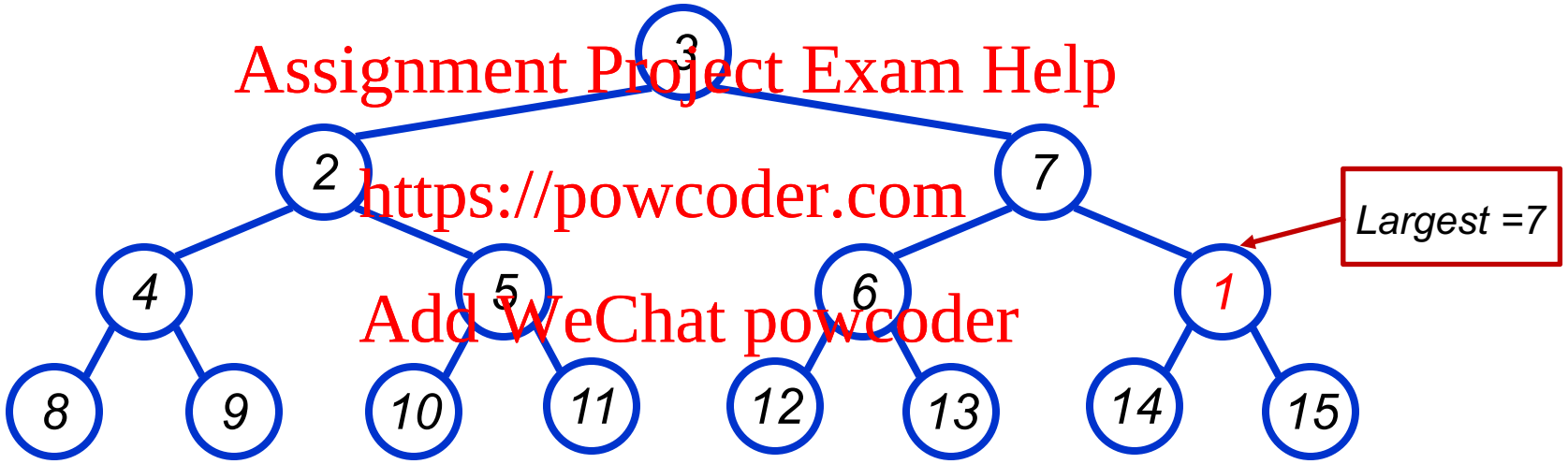
Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



A =	3	2	1	4	5	6	7	8	9	10	11	12	13	14	15

Max-Heapify(A,7)

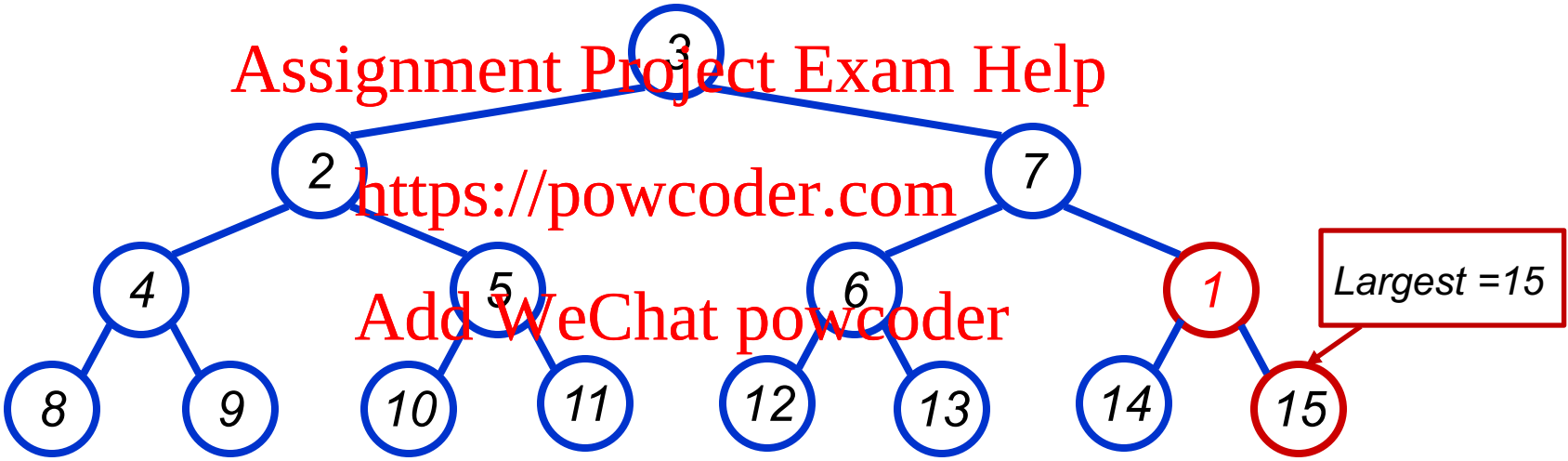
Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



A =	3	2	7	4	5	6	1	8	9	10	11	12	13	14	15

Max-Heapify(A,7)

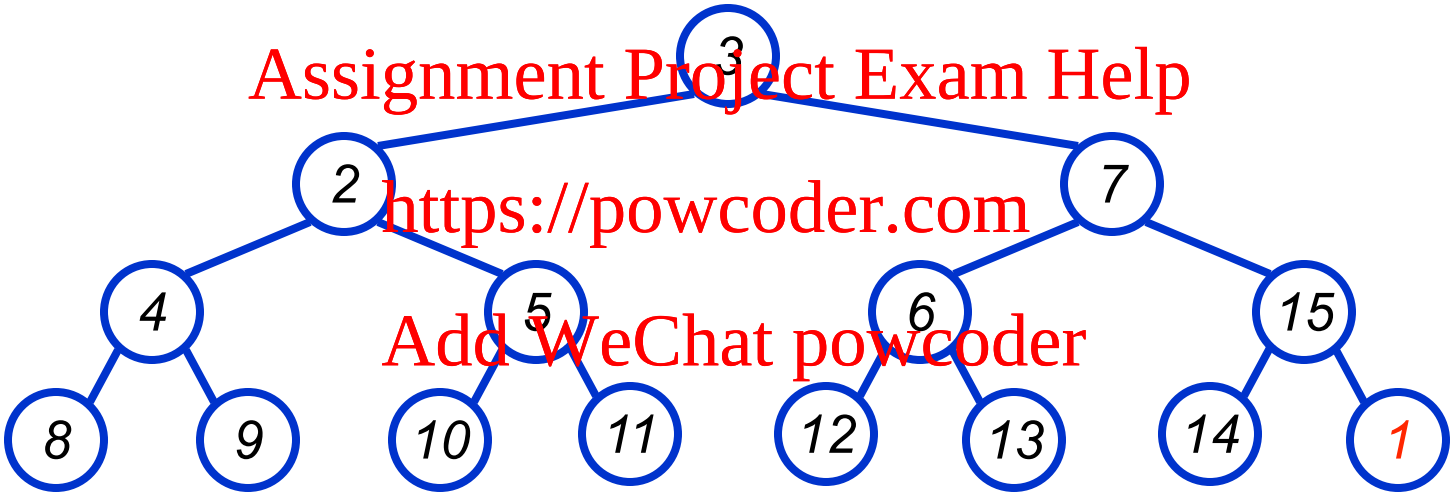
Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



A =	3	2	7	4	5	6	1	8	9	10	11	12	13	14	15

Max-Heapify(A,15)

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



A =	3	2	7	4	5	6	15	8	9	10	11	12	13	14	1

Running Time for Max-Heapify

```
Max-Heapify(A,i)
 1 l = Left(i)
 2 r = Right(i)
 3 if l ≤ A.heap-size and A[l] > A[i]
 4         largest  = l
 5 else  largest = i
 6 if r ≤ A.heap-size and A[r] > A[largest]
 7         largest = r
 8 if largest ≠ i
 9       exchange A[i] ↔ A[largest]
10       Max-Heapify(A, largest)
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Time to fix node i and its children = ***Θ(1)***

PLUS

Time to fix the subtree rooted at one of i's children = T(size of subree at largest)

Running Time for Max-Heapify(A, n)

- Fixing up relationships between i, l, and r takes $\Theta(1)$ time
- $T(n) = T(largest) + \Theta(1)$
- *If the heap at i has n elements, how many elements can the subtrees at l or r have?*
  - Draw it
- Answer: $2n/3$ (worst case: bottom row 1/2 full)
- $largest \leq 2n/3$ (worst case occurs when the last row of tree is exactly half full)
- So time taken by **`Max-Heapify()`** is given by

  $T(n) \leq T(2n/3) + \Theta(1)$

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Why T(2n/3) ?

- The worst case: bottom row 1/2 full
- Number of nodes at -
- level 0 i.e. root is 2^0
- level 1 is 2^1
- level 2 is 2^2
- ...
- level h is 2^h
- Summation of all nodes from level 0 up to level h
- $S = 2^0 + 2^1 + 2^2 + ... + 2^h$
- From geometric series summation rule we know that
- $x^0 + x^1 + x^2 + ... + x^n = (x^{n+1} - 1)/(x-1)$
- Substituting x = 2, we get
- $S = 2^{h+1} - 1$. i.e. $2^{h+1} = S + 1$
- As 2^{h+1} is the total nodes at level h+1, which means that the total number of leaf nodes (2^{h+1}) is one node more than the total number of non-leaf nodes (S).

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Why T(2n/3) ?

Now let's calculate the number of nodes in left subtree, right subtree and total ..

- Assume that number of non-leaf nodes in the left subtree of root = k.
- By the above reasoning, number of leaf nodes in the left subtree of root = k + 1. Number of non-leaf nodes in the right subtree of root = k as the tree is said to be exactly half full.
- Total number of nodes in the left subtree of root = k + k + 1 = 2k +1
- Total number of nodes in the tree, n = (2k + 1) + k + 1(root) = 3k + 2.
- Ratio of nodes in the left subtree and total nodes = (2k + 1) / (3k + 2) which is bounded above by 2/3.
- That's the reason of saying that the children's subtrees each have size at most 2n/3.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Analyzing Heapify(): Formal

- So we have

$$T(n) \le T(2n/3) + \Theta(1)$$

Assignment Project Exam Help

- By case 2 of the Master Theorem,

https://powcoder.com

$$T(n) = \mathrm{O}(\lg n)$$

Add WeChat powcoder

- Thus, **`Max-Heapify()`** takes logarithmic time

6.3 Building a heap

- Use Max-Heapify to convert an array A into a max-heap.
- How?
- Call Max-Heapify on each element in a bottom-up manner.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

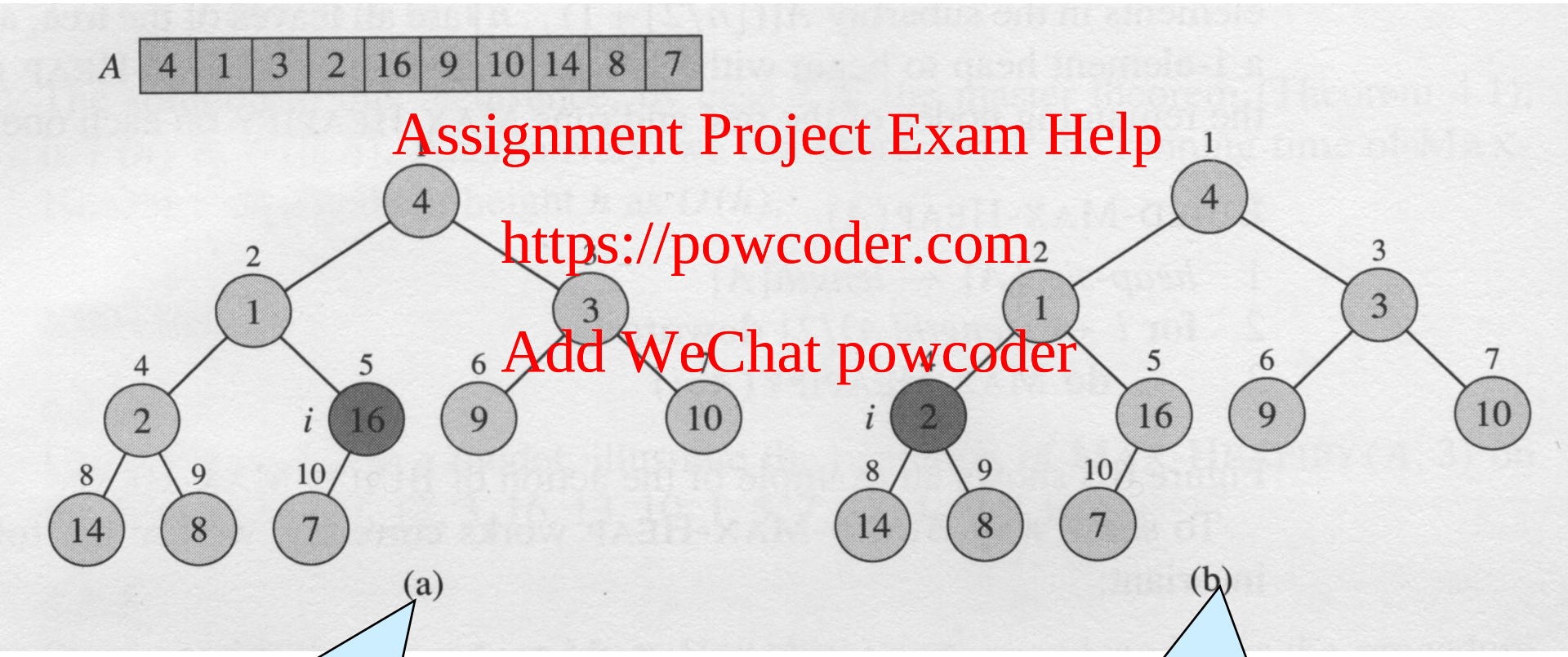
```
Build-Max-Heap(A)
1. A.heap-size = A.length
2.  for i = ⌊A.length / 2⌋ downto 1
3.        do Max-Heapify(A, i)
```

Why $\lfloor A.length / 2 \rfloor$?

-- the total number of nodes at leaf level is roughly the same as the total number of nodes without leaf level.

Build-Max-Heap(A)

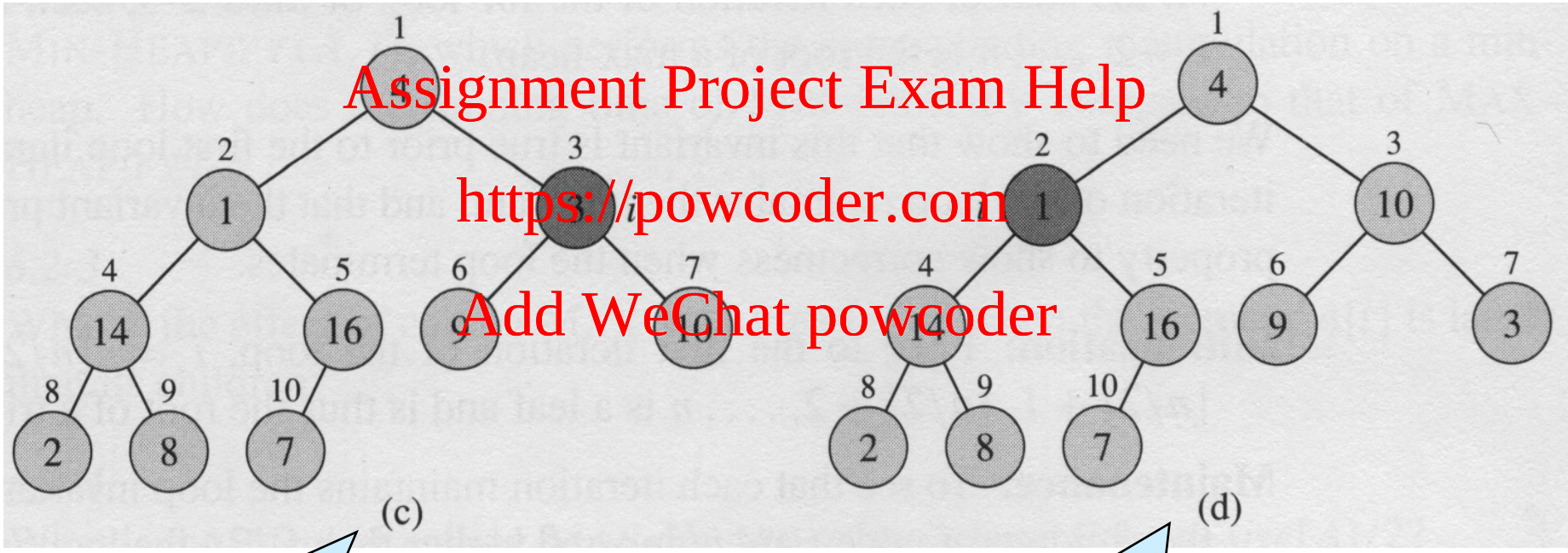
heap-size[A] = 10



i=5
Max-Heapify(A, i)

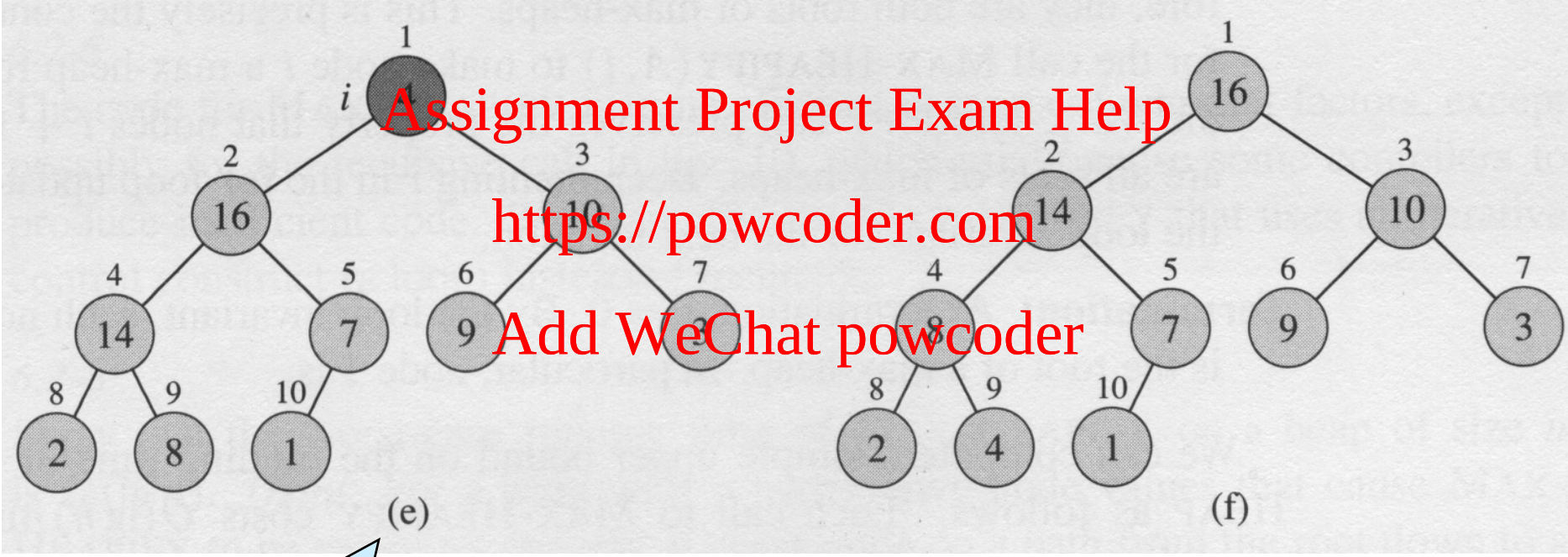
i=4
Max-Heapify(A, i)

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



i=3
Max-Heapify(A, i)

i=2
Max-Heapify(A, i)



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

i=1
Max-Heapify(A, i)

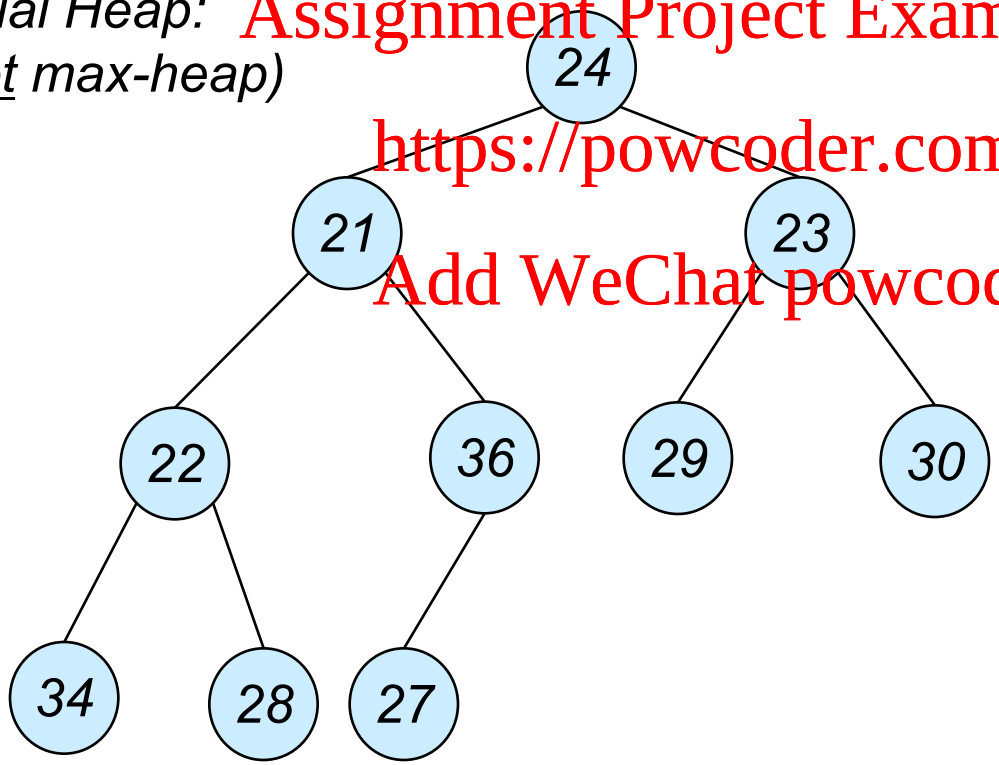
Build-Max-Heap – Example

Input Array:

24	21	23	22	36	29	30	34	28	27

Initial Heap:
(not max-heap)

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



Build-Max-Heap – Example

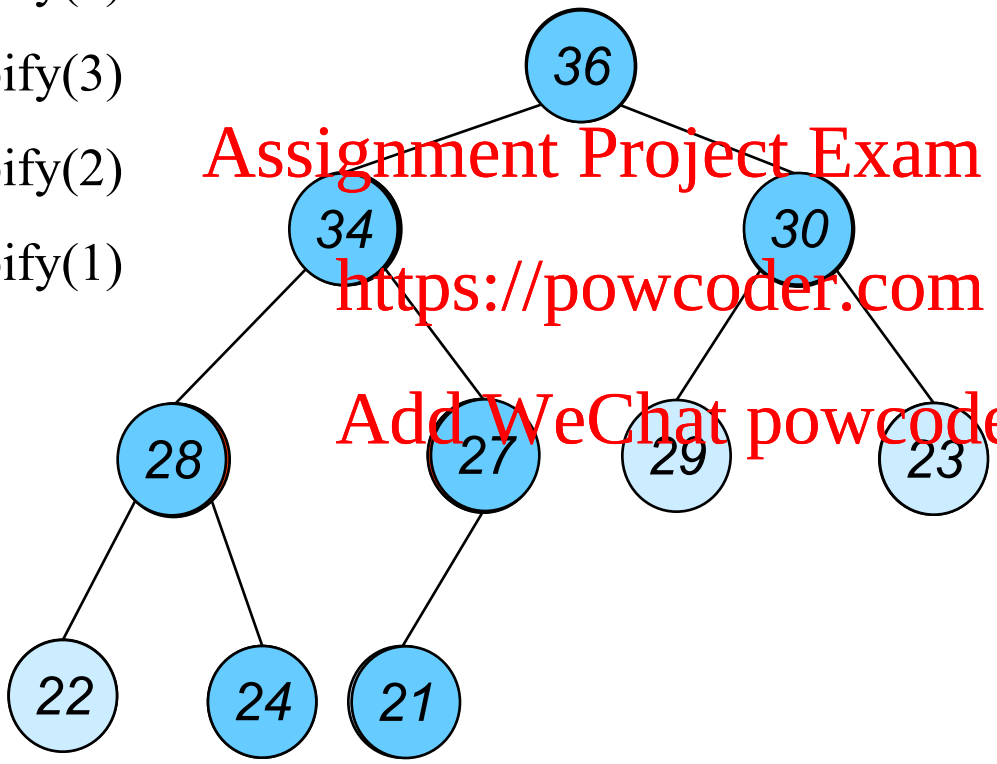
MaxHeapify($\lfloor 10/2 \rfloor = 5$)

MaxHeapify(4)

MaxHeapify(3)

MaxHeapify(2)

MaxHeapify(1)



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Running Time of *Build-Max-Heap*

- Loose upper bound:
  - Cost of a *Max-Heapify* call × No. of calls to *Max-Heapify*
  - $O(\lg n) \times O(n) = O(n \lg n)$
- Tighter bound:
  - Cost of a call to *Max-Heapify* at a node depends on the height, h, of the node – $O(h)$.
  - Height of most nodes smaller than n.
  - Height of nodes h ranges from 0 to $\lfloor \lg n \rfloor$.
  - A heap of size, n, has at most no. of nodes with height h ? (6.3-3)
  - is $\lceil n/2^{h+1} \rceil$ Why?
  - Answer will be posted on Canvas.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

6.3-3

A heap of size n has at most $\lceil n/2^{h+1} \rceil$ nodes with height h. **Key Observation**: For any $n > 0$, the number of leaves of nearly complete binary tree is $\lceil n/2 \rceil$. *Proof by induction* **Base case**: Show that it's true for $h = 0$. This is the direct result from above observation. **Inductive step**: Suppose it's true for $h - 1$. Let N_h be the number of nodes at height h in the n-node tree T. Consider the tree T' formed by removing the leaves of T . It has $n' = n - \lceil n/2 \rceil = \lfloor n/2 \rfloor$ nodes. Note that the nodes at height h in T would be at height $h - 1$ in tree T'. Let N'_{h-1} denote the number of nodes at height $h - 1$ in T', we have $N_h = N'_{h-1}$. By induction, we have $N_h = N'_{h-1} = \lceil n'/2^h \rceil = \lceil \lfloor n/2 \rfloor / 2^h \rceil \leq \lceil (n/2)/2^h \rceil = \lceil n/2^{h+1} \rceil$.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Running Time of *Build-Max-Heap*

Tighter Bound for T(Build-Max-Heap)

T(Build-Max-Heap)

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h)$$

$$= O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right)$$

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}$$

$$\leq \sum_{h=0}^{\infty} \frac{h}{2^h} \qquad (\because \sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2})$$

$$= \frac{1/2}{(1-1/2)^2} \qquad , x = 1/2 \text{ in (A.8)}$$

$$= 2$$

$$O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right) = O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h} \right)$$

$$= O(n)$$

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Can build a heap from an unordered array in linear time

6.4 Heapsort algorithm

- Sort by maintaining the as yet unsorted elements as a max-heap.
- Start by building a max-heap on all elements in A.
  - Maximum element is in the root, $A[1]$.
- Move the maximum element to its correct final position.
  - Exchange $A[1]$ with $A[n]$.
- Discard $A[n]$ – it is now sorted.
  - Decrement heap-size[A].
- Restore the max-heap property on $A[1..n-1]$.
  - Call *Max-Heapify*(A, 1).
- Repeat until heap-size[A] is reduced to 2.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Heapsort algorithm

`Heapsort(A)`

- To sort an array in place.

```
Heapsort(A)
1  Build-Max-Heap(A)
2  for i = A.length down to 2
3      exchange A[1]↔A[i]
4       A.heap-size = A.heap-size -1
5       Max-Heapify(A,1)
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Heapsort Visualization 1

Heap Sort Dancing 2

Heapsort Visualization 3

Heapsort Visualization 4

Heapsort

```
Heapsort(A)
{
      Build-Max-Heap(A);
      for (i = length(A) downto 2)
      {
            Swap(A[1], A[i]);
            heap_size(A) -= 1;
            Max-Heapify(A, 1);
      }
}
```

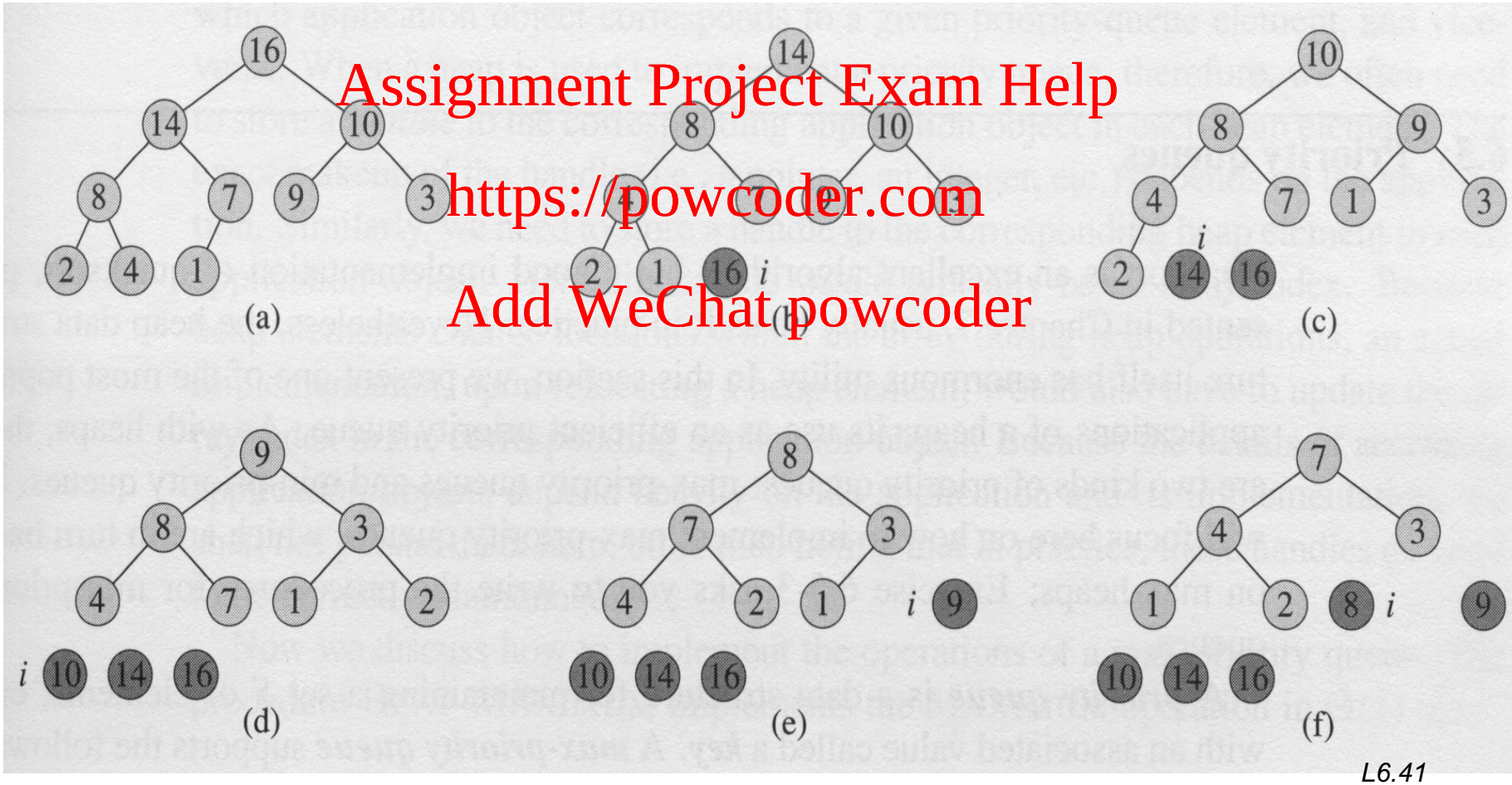
Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Operation of Heapsort

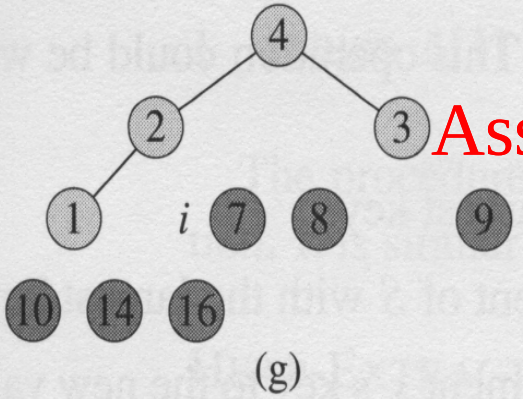
Assignment Project Exam Help

https://powcoder.com

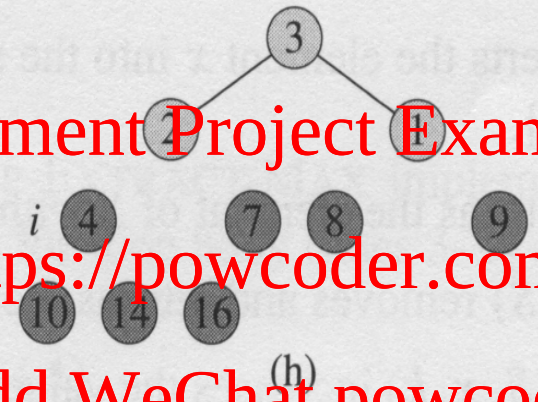
Add WeChat powcoder



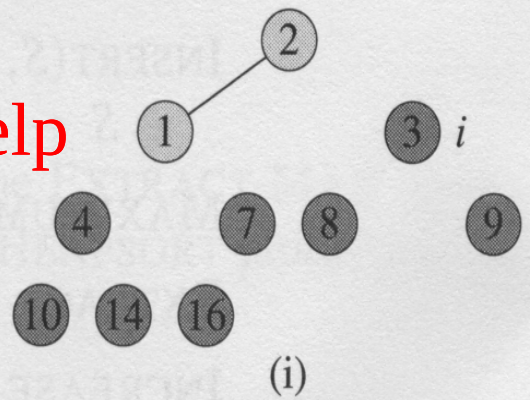
Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



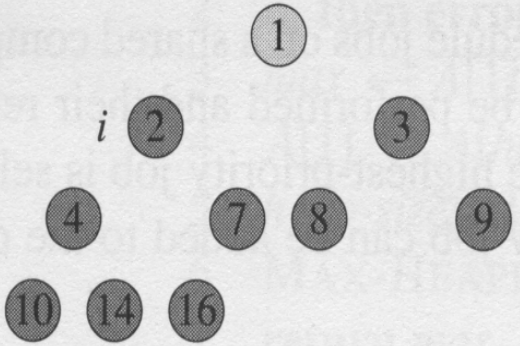
(g)



(h)



(i)

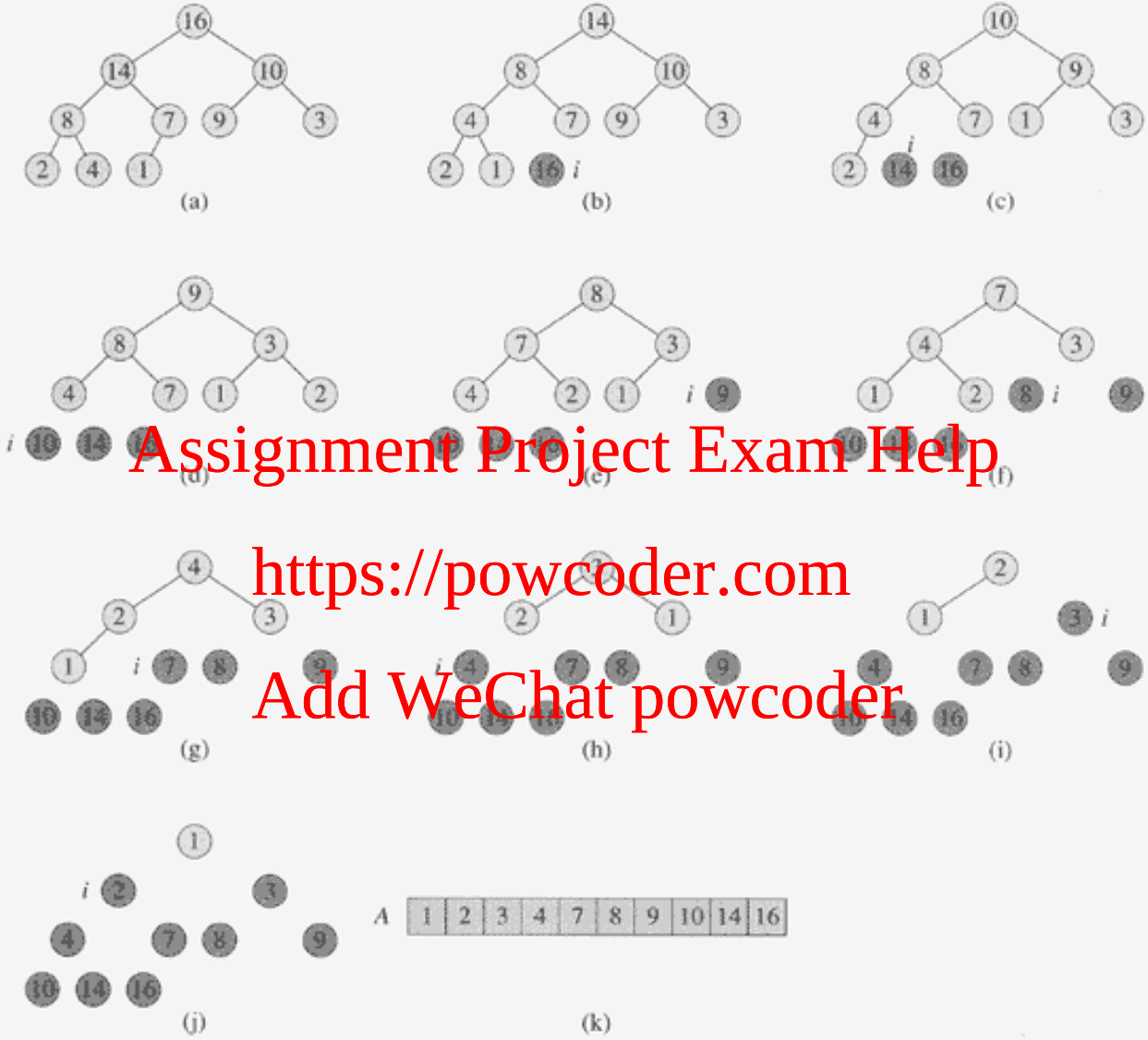


(j)

(k)

Analysis: O(n lg n)

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Figure 6.4 The operation of HEAPSORT. **(a)** The max-heap data structure just after it has been built by BUILD-MAX-HEAP. **(b)–(j)** The max-heap just after each call of MAX-HEAPIFY in line 5. The value of i at that time is shown. Only lightly shaded nodes remain in the heap. **(k)** The resulting sorted array A.

Algorithm Analysis

```
Heapsort(A)
1  Build-Max-Heap(A)
2  for i = A.length down to 2
3      exchange A[1]↔A[i]
4       A.heap-size = A.heap-size -1
5       Max-Heapify(A,1)
```

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

- In-place: no need additional space for sorting.
- Not Stable: do not maintain the relative location/order of records with equal keys.
- Build-Max-Heap takes $O(n)$ and each of the $n-1$ calls to Max-Heapify takes time $O(\lg n)$.
- Therefore, $T(n) = O(n \lg n)$

Heap Procedures for Sorting

- Max-Heapify $O(\lg n)$
- Build-Max-Heap $O(n)$
- HeapSort $O(n \lg n)$

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Analyzing Heapsort

- The call to **`Build-Max-Heap()`** takes $O(n)$ time
- Each of the n - 1 calls to **`Max-heapify()`** takes $O(\lg n)$ time
- Thus the total time taken by **`HeapSort()`**

$= O(n) + (n - 1)\,O(\lg n)$

$= O(n) + O(n \lg n)$

$= O(n \lg n)$

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

6.5 Priority Queues

- Popular & important application of heaps.
- Max and min priority queues.
- Maintains a *dynamic* set S of elements.
- Each set element has a *key* – an associated value.
- Goal is to support insertion and extraction efficiently.
- Applications:
  - Ready list of processes in operating systems by their priorities – the list is highly dynamic
  - In event-driven simulators to maintain the list of events to be simulated in order of their time of occurrence.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Priority Queues

- Heapsort is a nice algorithm, but in practice Quicksort (coming up) usually wins
- But the heap data structure is incredibly useful for implementing *priority queues*
  - A data structure for maintaining a set S of elements, each with an associated value or *key*
  - Supports the operations **`Insert()`**, **`Maximum()`**, and **`ExtractMax()`**
  - ***What might a priority queue be useful for?***

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Priority Queues

- A max-priority queue support the following operations:

`Insert(S, x)` $O(\lg n)$

inserts the element x into the set S.

Assignment Project Exam Help

`Maximum(S)` $O(1)$

returns the element of S with the largest key.

https://powcoder.com

Add WeChat powcoder

`Extract-Max(S)` $O(\lg n)$

removes and returns the element of S with the largest key.

`Increase-Key(S,x,k)` $O(\lg n)$

increases the value of element x's key to the new value k, where $k \geq x$'s current key value

Heap-Maximum

```
Heap-Maximum(A)
1 return A[1]
```

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Heap_Extract-Max

```
Heap_Extract-Max(A)
1  if A.heap-size < 1
2      error "heap underflow"
3  max = A[1]
4  A[1]= A[A.heap-size]
5  A.heap-size = A.heap-size - 1
6  Max-Heapify(A, 1)
7  return max
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

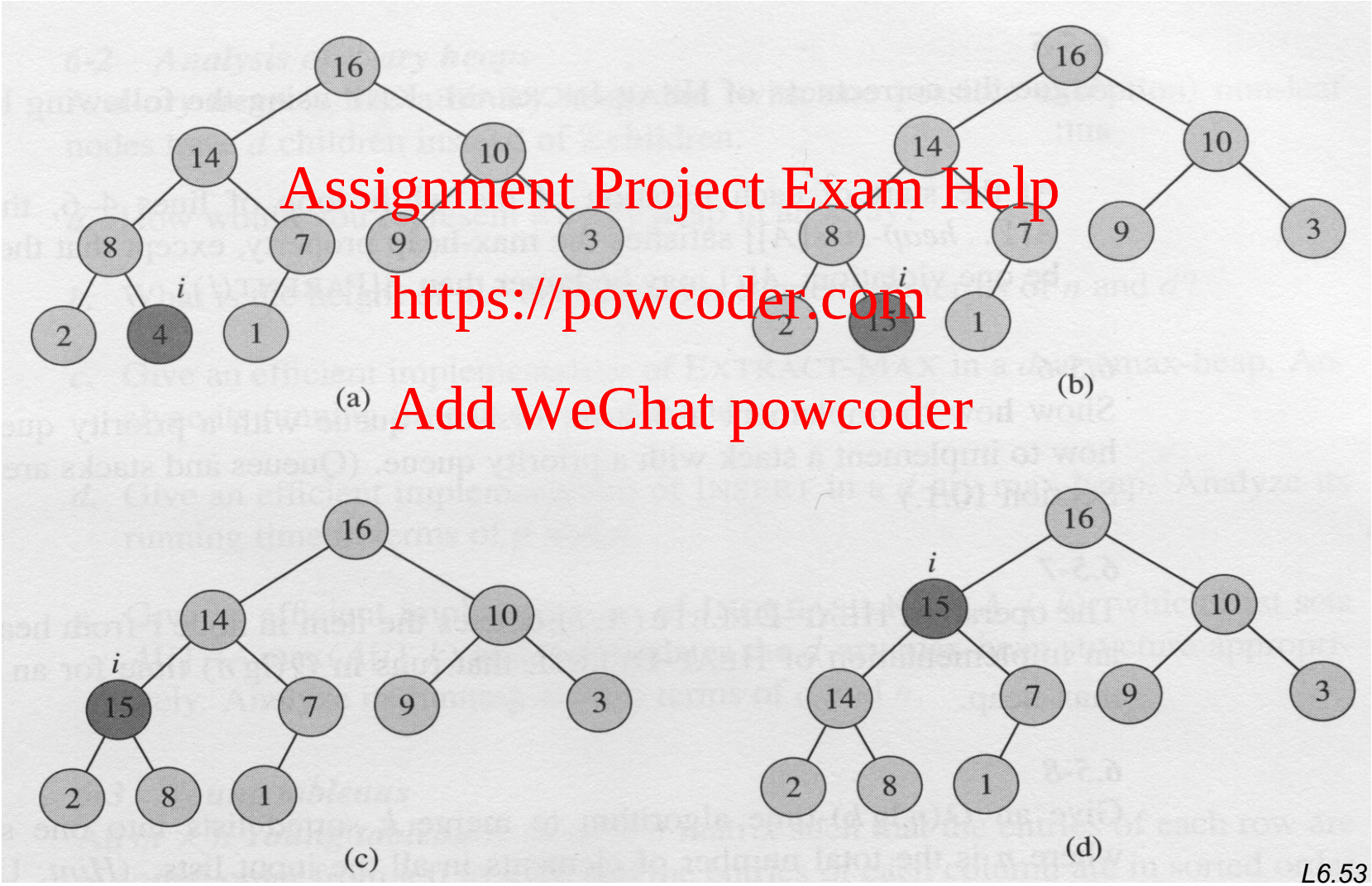
Running time : Dominated by the running time of Max-Heapify
= O(lg n)

Heap-Increase-Key

```
Heap-Increase-Key (A, i, key)
1  if key < A[i]
2     error "new key is smaller than
                    current key"
3  A[i] = key
4  while i > 1 and A[Parent(i)] < A[i]
5        exchange A[i] ↔ A[Parent(i)]
6        i = Parent(i)
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Heap-Increase-Key(*A*,*i*,15)



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

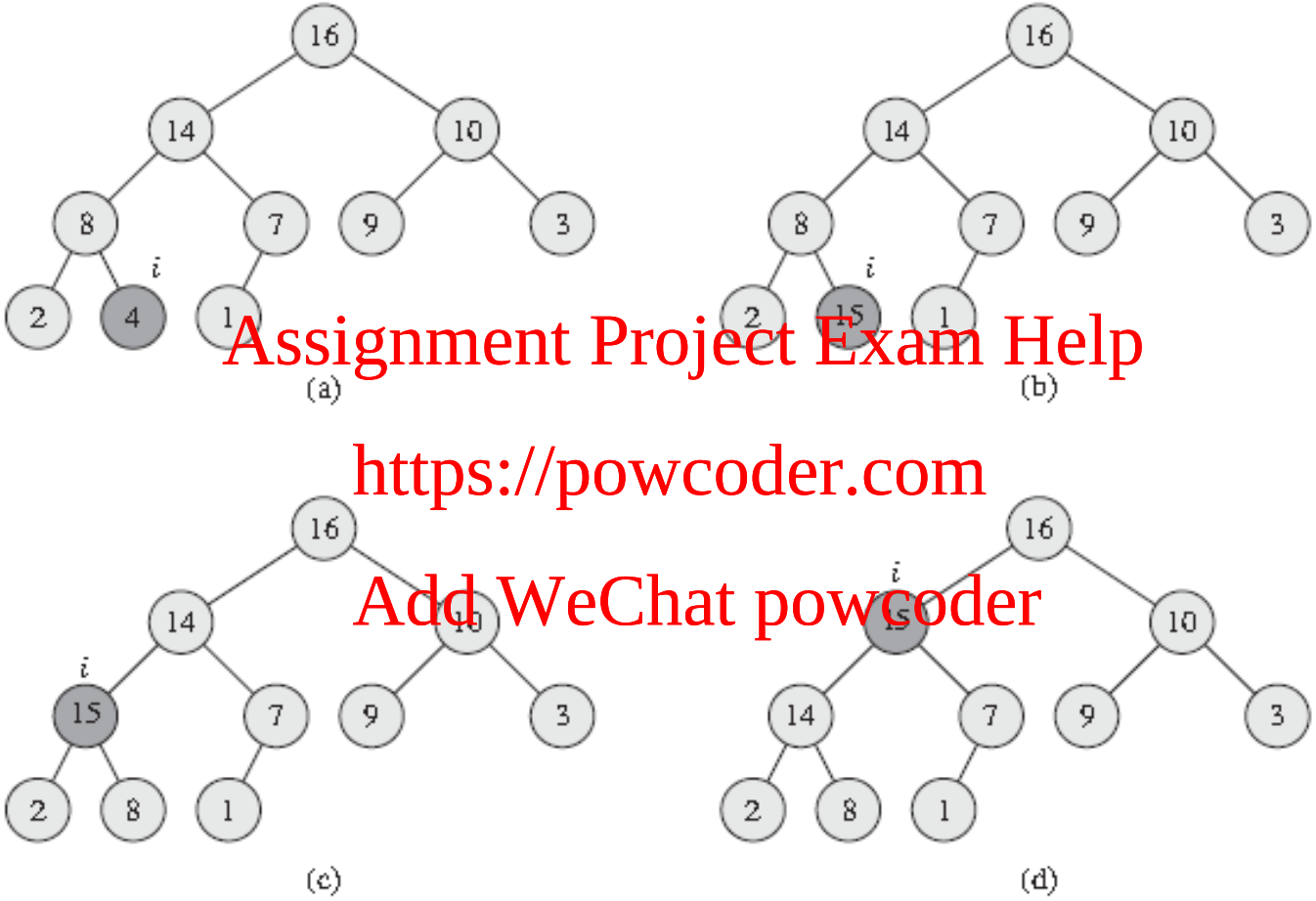
Heap-Increase-Key(*A*, *i*, *key*)

```
Heap-Increase-Key(A, i, key)
1    If key < A[i]
2          error "new key is smaller than the current key"
3    A[i] = key
4    while i > 1 and A[Parent[i]] < A[i]
5          exchange A[i] ↔ A[Parent[i]]
6                i = Parent[i]
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
Heap-Insert(A, key)
1    A.heap-size = heap-size[A] + 1
2    A[A.heap-size]  = –∞
3    Heap-Increase-Key(A, A.heap-size, key)
```

Examples



Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Figure 6.5 The operation of HEAP-INCREASE-KEY. **(a)** The max-heap of Figure 6.4(a) with a node whose index is i heavily shaded. **(b)** This node has its key increased to 15. **(c)** After one iteration of the **while** loop of lines 4–6, the node and its parent have exchanged keys, and the index i moves up to the parent. **(d)** The max-heap after one more iteration of the **while** loop. At this point, $A[\text{PARENT}(i)] \geq A[i]$. The max-heap property now holds and the procedure terminates.