S5 Non-Linear Data Structures: Heap and Priority Queue

BITS Pilani
Hyderabad Campus

Dr. Rajib Ranjan Maiti
CSIS Dept, Hyderabad Campus

innovate achieve lead

BITS Pilani
Hyderabad Campus

Data Structures and Algorithms Design (Merged-SEZG519/SSZG519)

S5 Non-Linear Data Structures: Heap and Priority Queue

Content of L-3

3.3. Heaps

- 3.3.1. Definition and Properties
- 3.3.2. Representations (Vector Based and Linked)
- 3.3.3. Insertion and deletion of elements
- 3.3.4. Heap sort

3.4 Priority Queue

- 3.4.1. Concept
- 3.4.2. Implementation

So far we studied

- Non-Linear data structure: Tree and Binary Tree

Heap tree

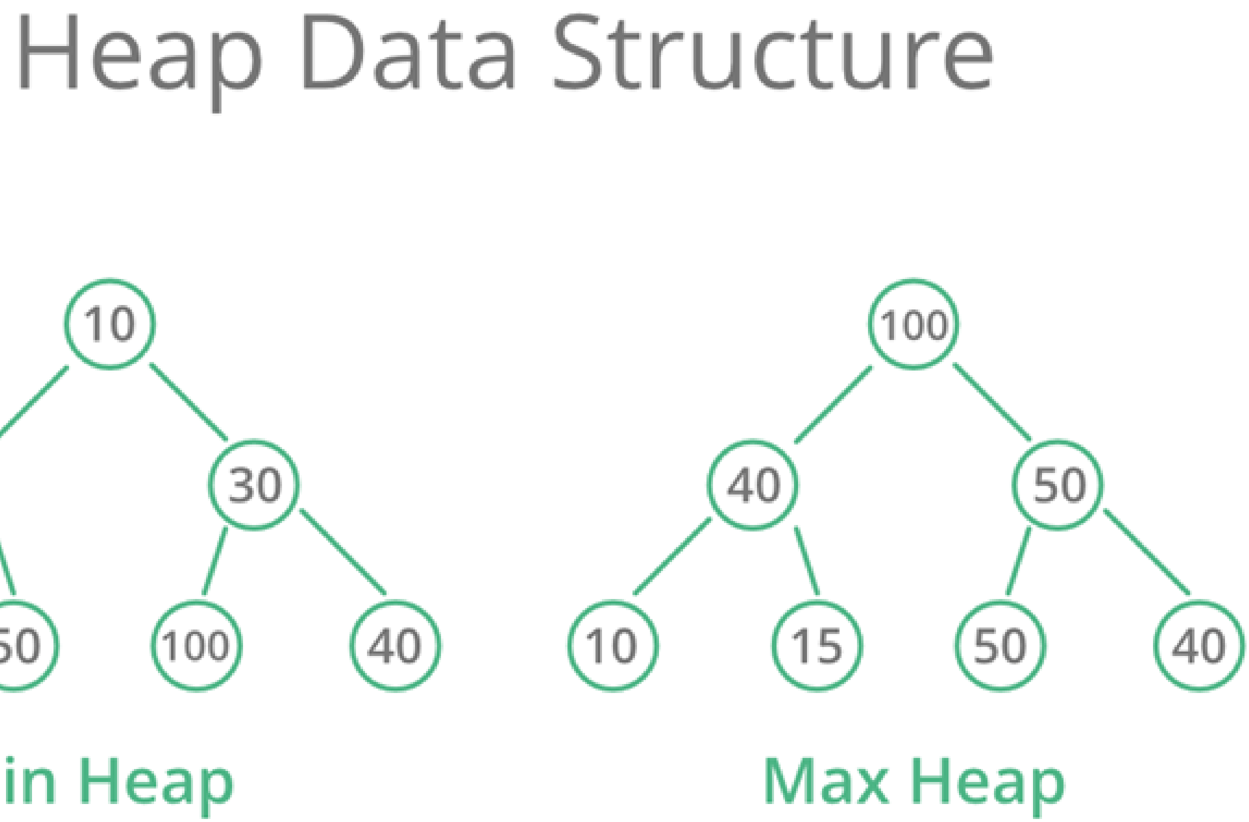
Heap: a complete binary tree that satisfies the following properties:

- Max heap: For each node N in H, the value at N is greater than or equal to value of each children of N.
- Min heap: For each node N in H, the value at N is smaller than or equal to value of each children of N.

Applications:

- Heap sort
- Implementation of priority queue

Heap tree


Heap Data Structure
10
15
30
40
50
100
40
Min Heap
100
40
50
10
15
50
40
Max Heap

Heap tree (Insert)

Algorithm MAX_HEAP_INSERT(item)

//Tree is stored in array A with N elements.

```
if N >= size
        print "Tree is full"
        exit
N ← N+1
A[N] ← item
i ← N
```

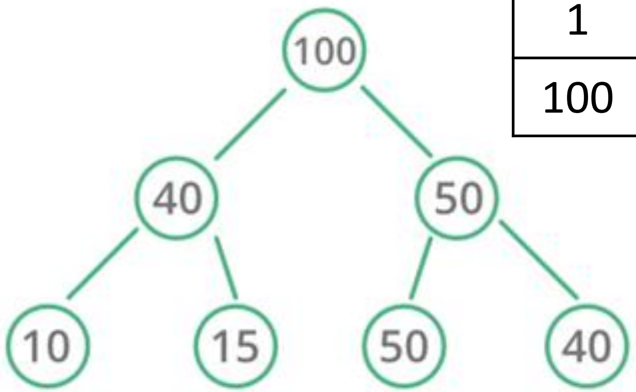
$j \leftarrow \left\lfloor \frac{i}{2} \right\rfloor$

while j > 0 and A[j] < A[i] **do**

```
        Exchange(A[i], A[j])
        i ← j
```

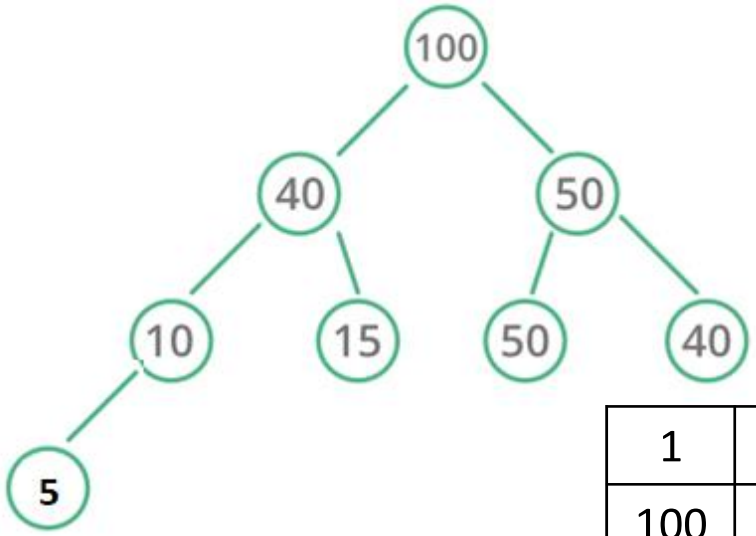
$\quad\quad j \leftarrow \left\lfloor \frac{j}{2} \right\rfloor$

Heap tree (Insert) Example



1	2	3	4	5	6	7	8	9	10
100	40	50	10	15	50	40			

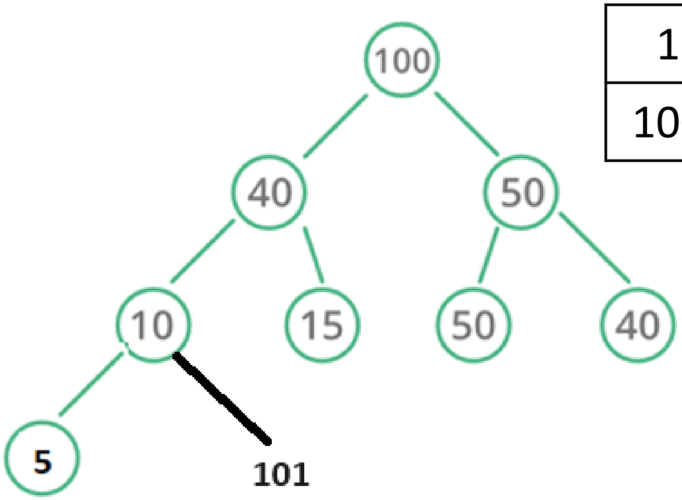
Insert(5)



i ← 8
j ← 8/2 = 4
A[j] > A[i] ➔ Return

1	2	3	4	5	6	7	**8**	9	10
100	40	50	10	15	50	40	**5**		

Heap tree (Insert) Example



1	2	3	4	5	6	7	8	9	10
100	40	50	10	15	50	40	5		

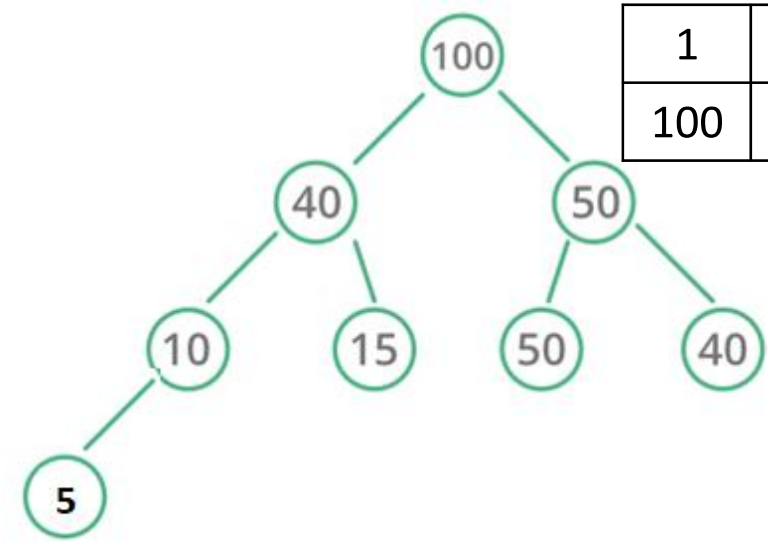
Insert(101)

1	2	3	**4**	5	6	7	8	**9**	10
100	40	50	**10**	15	50	40	5	**101**	

i ← 9
j ← 9/2 = 4
A[j] < A[i] ➔ Exchange

1	2	3	**4**	5	6	7	8	**9**	10
100	40	50	**101**	15	50	40	5	**10**	

Heap tree (Insert) Example



1	2	3	4	5	6	7	8	9	10
100	40	50	10	15	50	40	5		

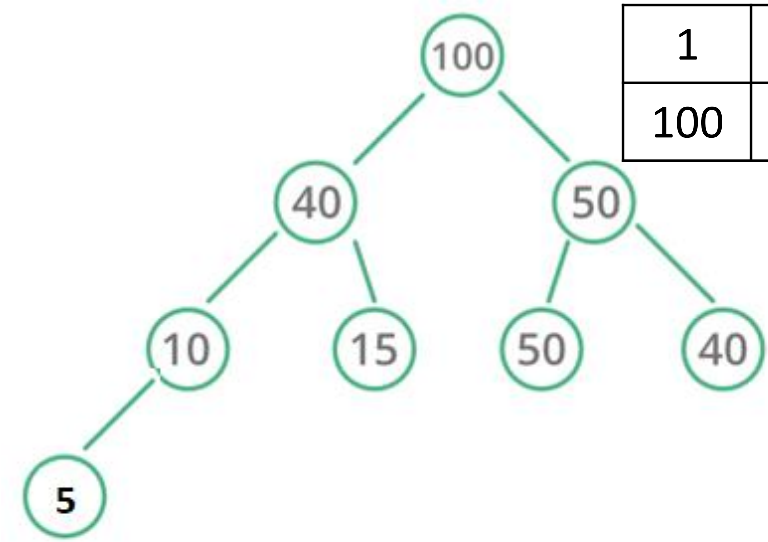
Insert(101)

1	**2**	3	**4**	5	6	7	8	9	10
100	**40**	50	**101**	15	50	40	5	10	

1	**2**	3	**4**	5	6	7	8	9	10
100	**101**	50	**40**	15	50	40	5	10	

i ← 4
j ← 4/2 = 2
A[j] < A[i] ➔ Exchange

Heap tree (Insert) Example



1	2	3	4	5	6	7	8	9	10
100	40	50	10	15	50	40	5		

Insert(101)

1	2	3	**4**	5	6	7	8	**9**	10
100	40	50	**10**	15	50	40	5	**101**	

1	**2**	3	**4**	5	6	7	8	9	10
100	**101**	50	**40**	15	50	40	5	10	

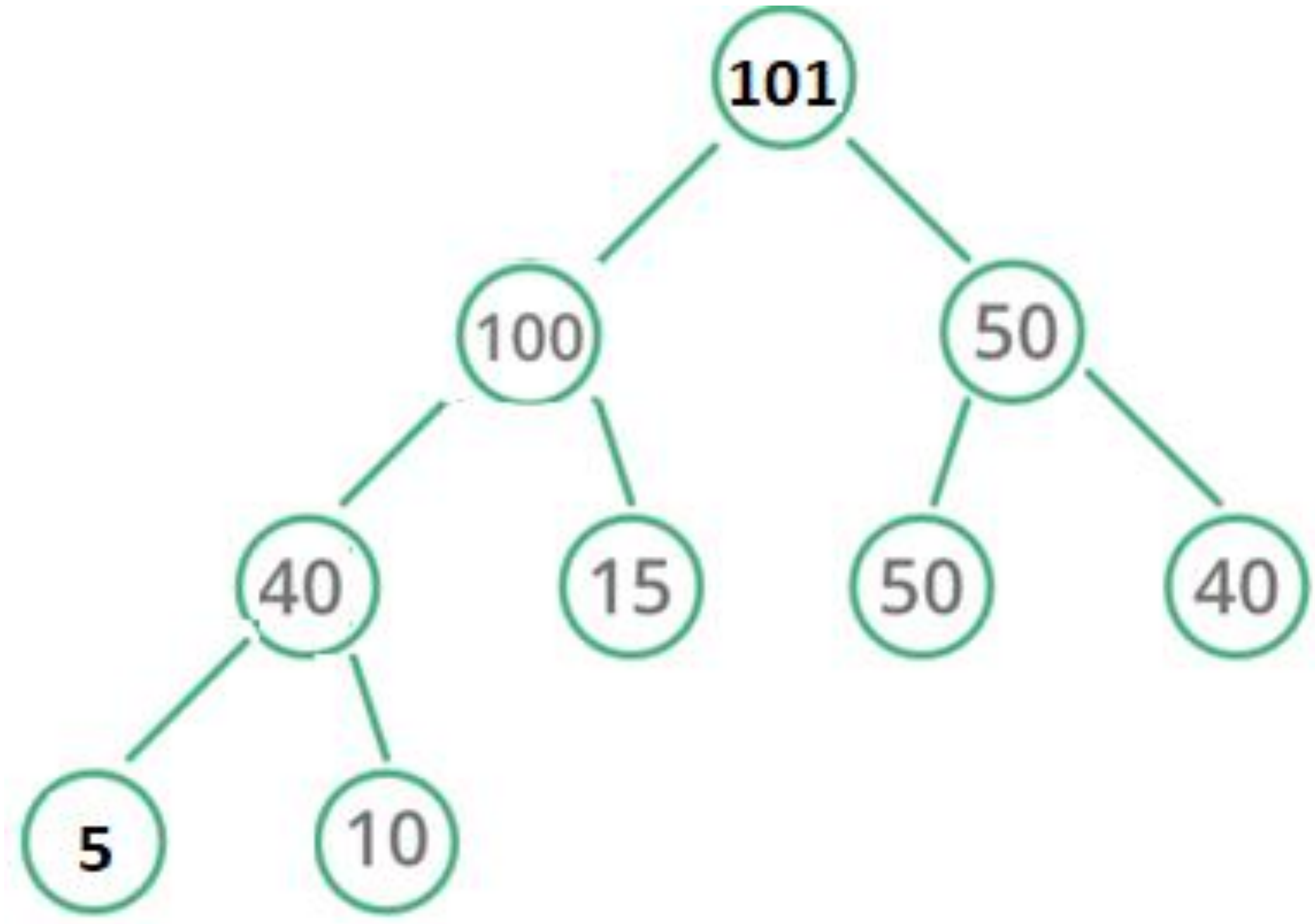
1	**2**	3	4	5	6	7	8	9	10
101	**100**	50	40	15	50	40	5	10	

$i \leftarrow 2$

$j \leftarrow 2/2 = 1$

$A[j] < A[i] \Rightarrow$ Exchange

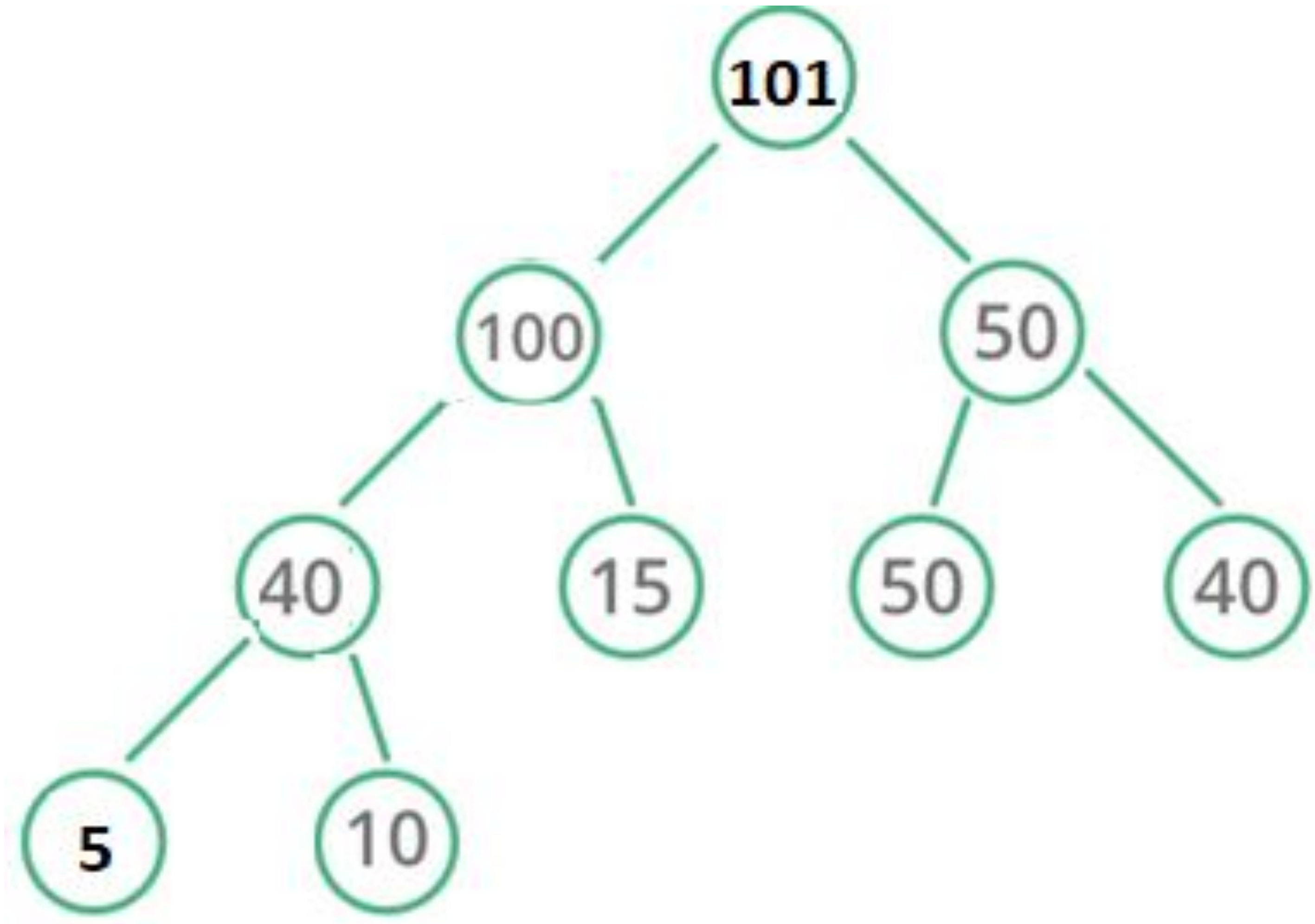
Heap tree (Insert) Example



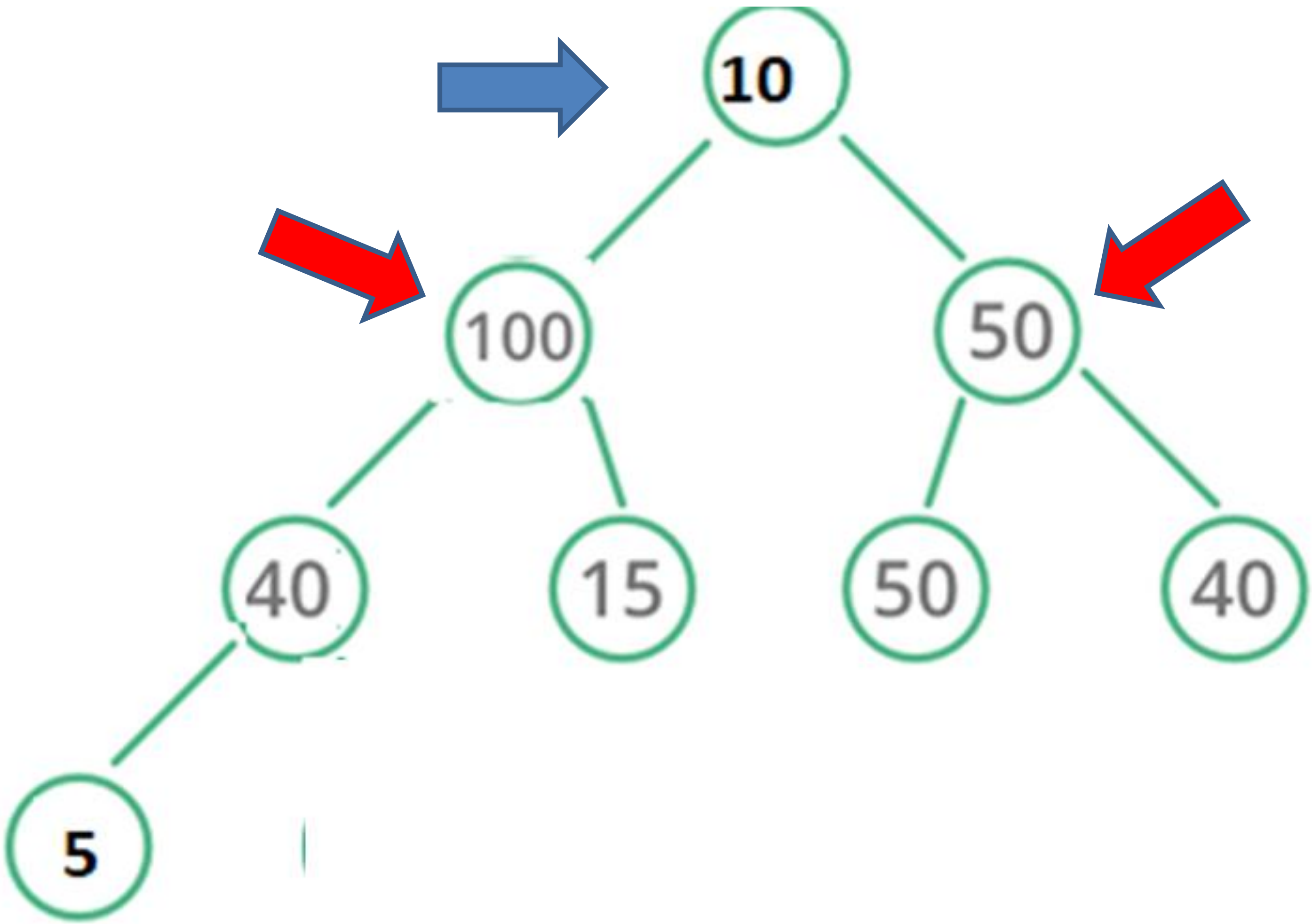
Heap tree (Delete)

```
Algorithm MAX_HEAP_DELETE( )
if N = 0 then
        print "Empty Tree"
        exit
item ← A[1]
A[1] ← A[N]
N ← N – 1
flag ← FALSE
i ← 1
while flag = FALSE and i < N do
        left ← 2*i, right ← 2*i +1
        if A[i] > A[left] and A[i] > A[right] then
                flag ← TRUE
        else if A[left] > A[right] and A[i] < A[left] then
                Swap(A[i], A[left])
                i ← left
        else if A[right] > A[left] and A[i] < A[right] then
                Swap(A[i], A[right])
                i ← right
```

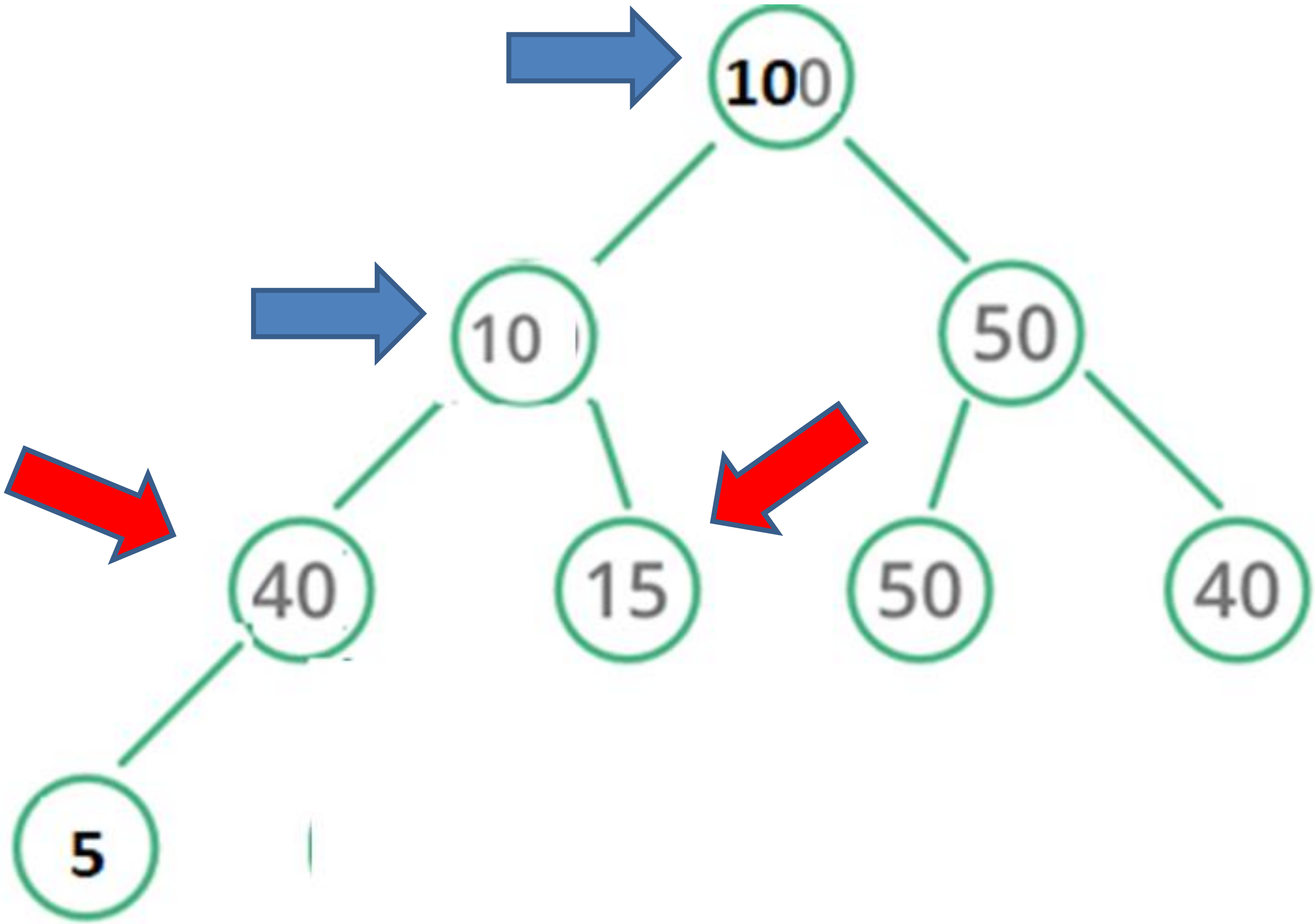
Heap tree (Delete) Example



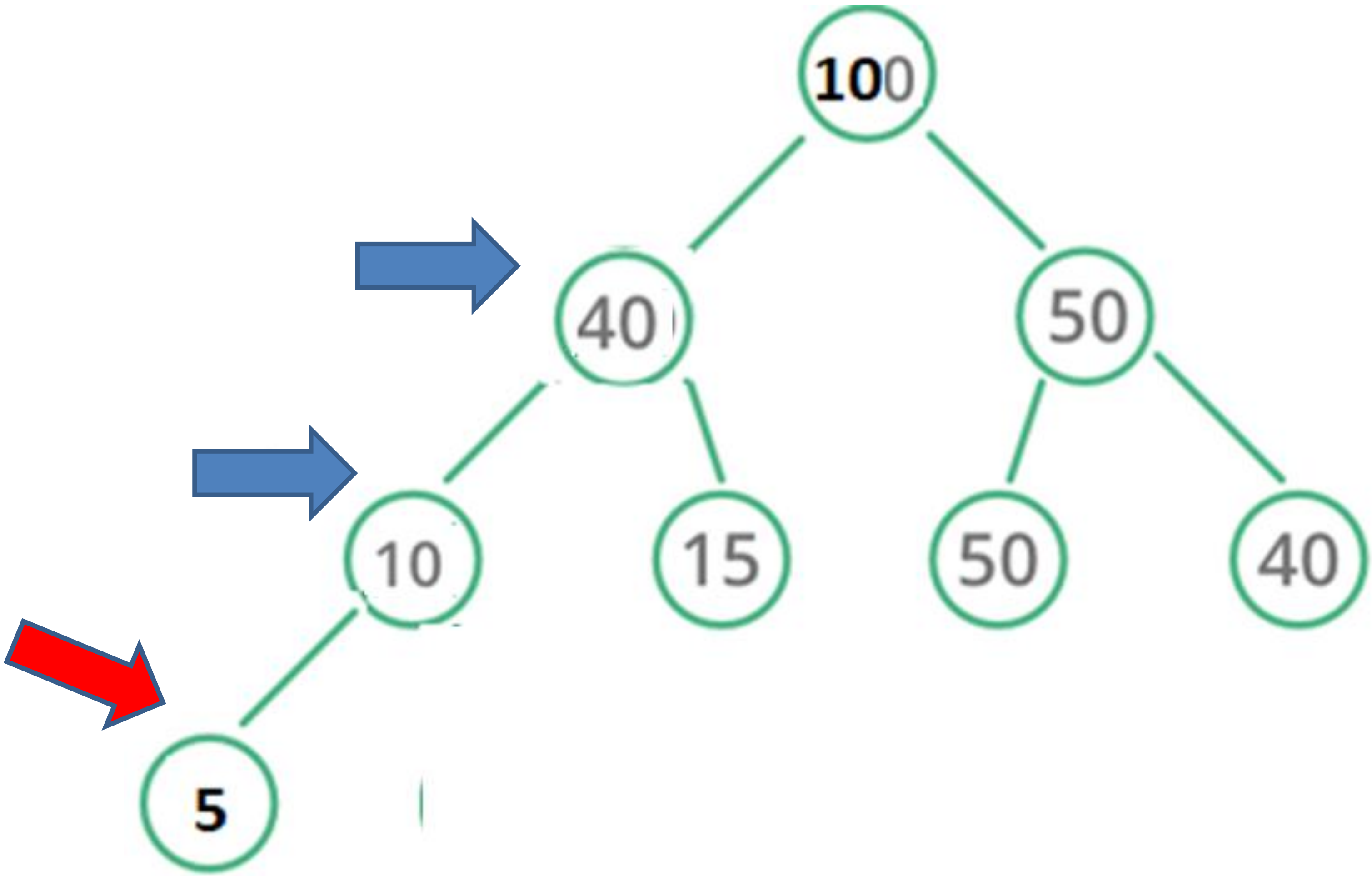
Heap tree (Delete) Example



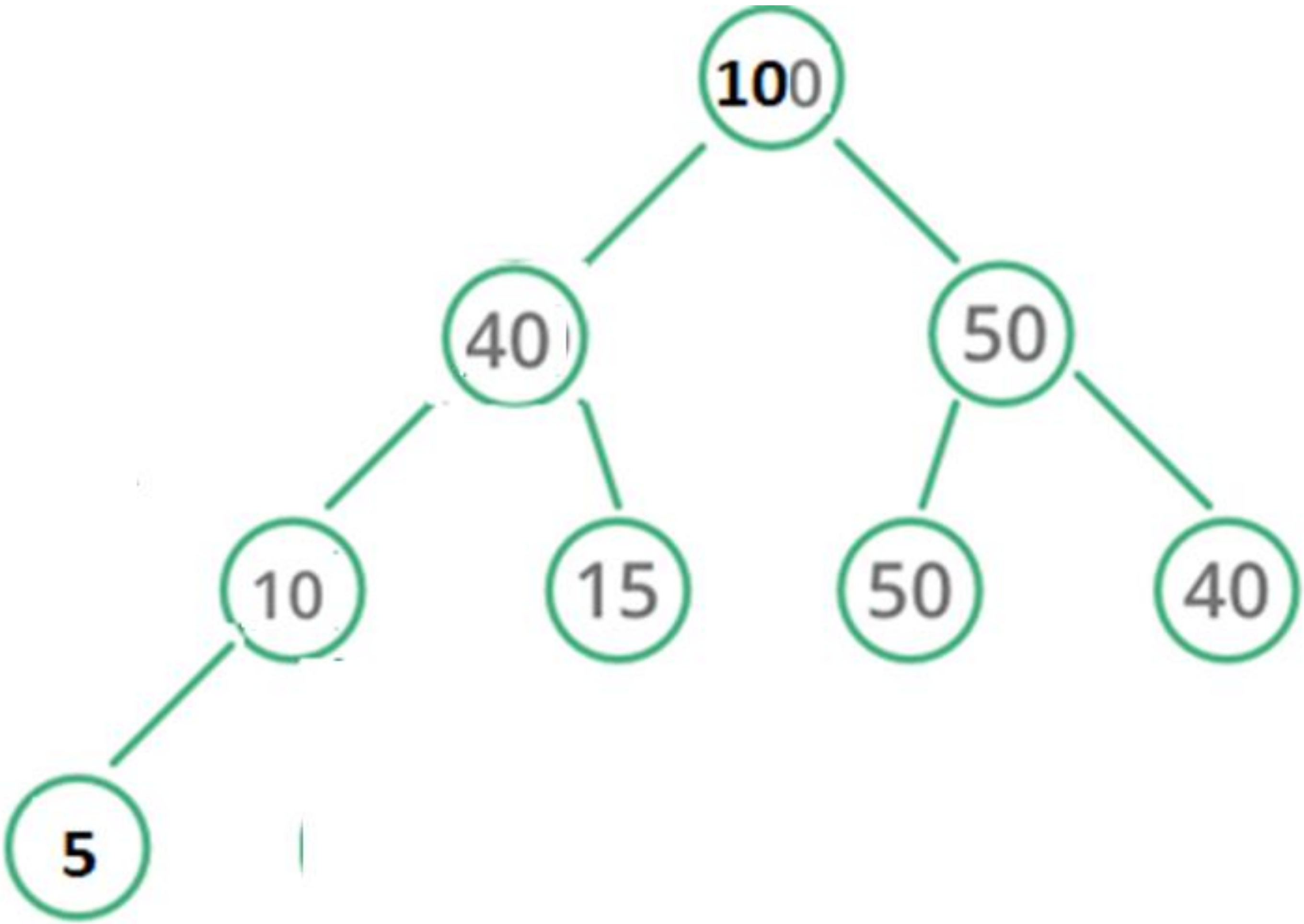
Heap tree (Delete) Example



Heap tree (Delete) Example



Heap tree (Delete) Example



Applications: Heap Sort

Step 1: Build a max/min heap tree with the given data.

Step 2: Repeat until tree is not empty

a) Delete the root node from heap and replace last node at root node.

b) Rebuild the heap after deletion.

c) Place the deleted node value in the output.

Step 3: Reverse the output if max heap was used.

Applications: Heap Sort

```
Algorithm Heap_Sort(data)
Create_Max_Heap(data)
i ← N        //N is number of elements in data
while i > 1 do
        Swap(data[1], data[i])
        i ← i-1
        j ← 1
        while j < i do
                left ← 2*j
                right ← 2*j + 1
                if data[j] < data[left] and data[left] > data[right] then
                        Swap(data[j], data[left])
                        j ← left
                else if data[j] < data[right] and data[right] > data[left] then
                        Swap(data[j], data[left])
                        j ← right
                else
                        break
```

Applications: Priority Queue

- A priority queue is a type of queue that arranges elements based on their priority values.
- Properties:
  - Every item has a priority associated with it.
  - An element with high priority is dequeued before an element with low priority.
  - If two elements have the same priority, they are served according to their order in the queue.

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
5	4	3	4	5	5	3	2	1	5

Applications: Implementation of Priority Queue

Step 1: Build a max heap tree with the given data as per priority value.

Step 2: Repeat until tree is not empty

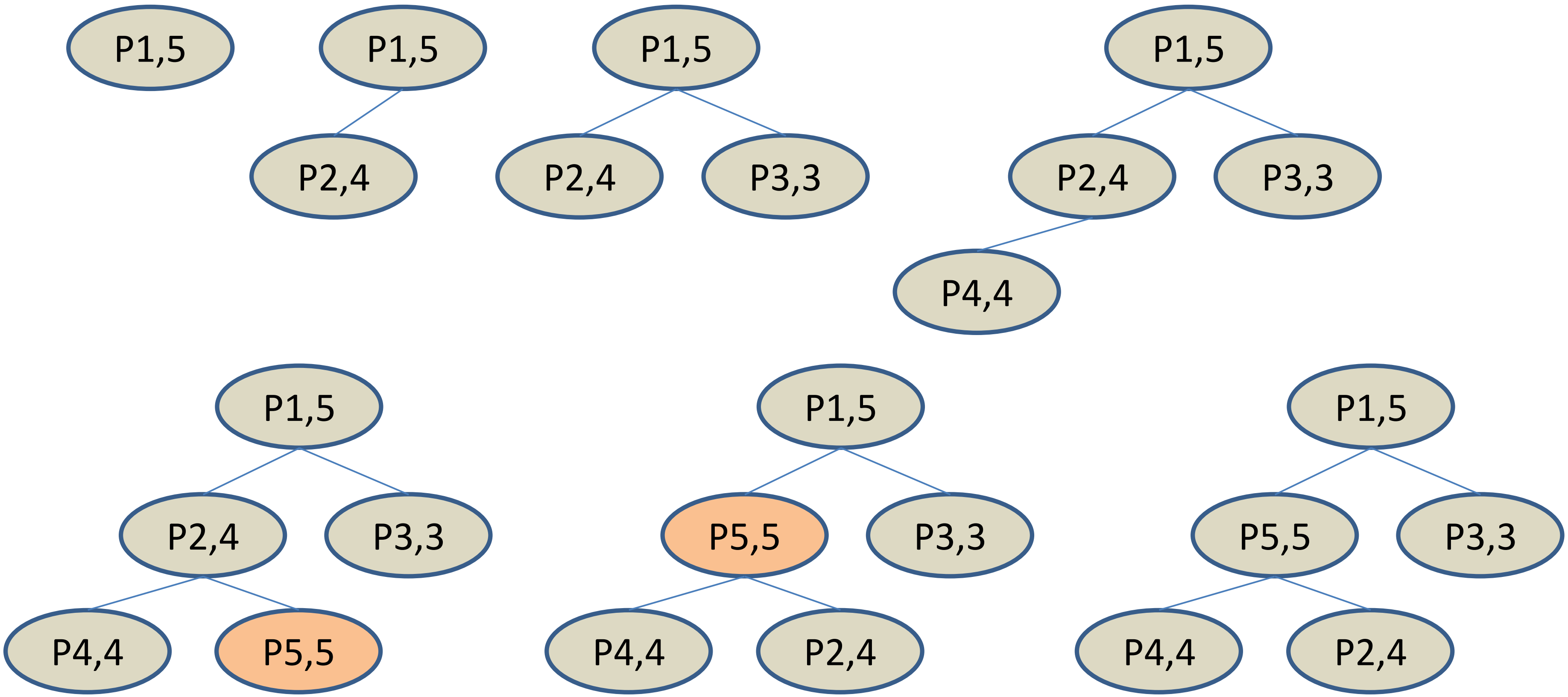
a) Delete the root node from heap and replace last node at root node.

b) Rebuild the heap after deletion.

c) Place the deleted node value in the output.

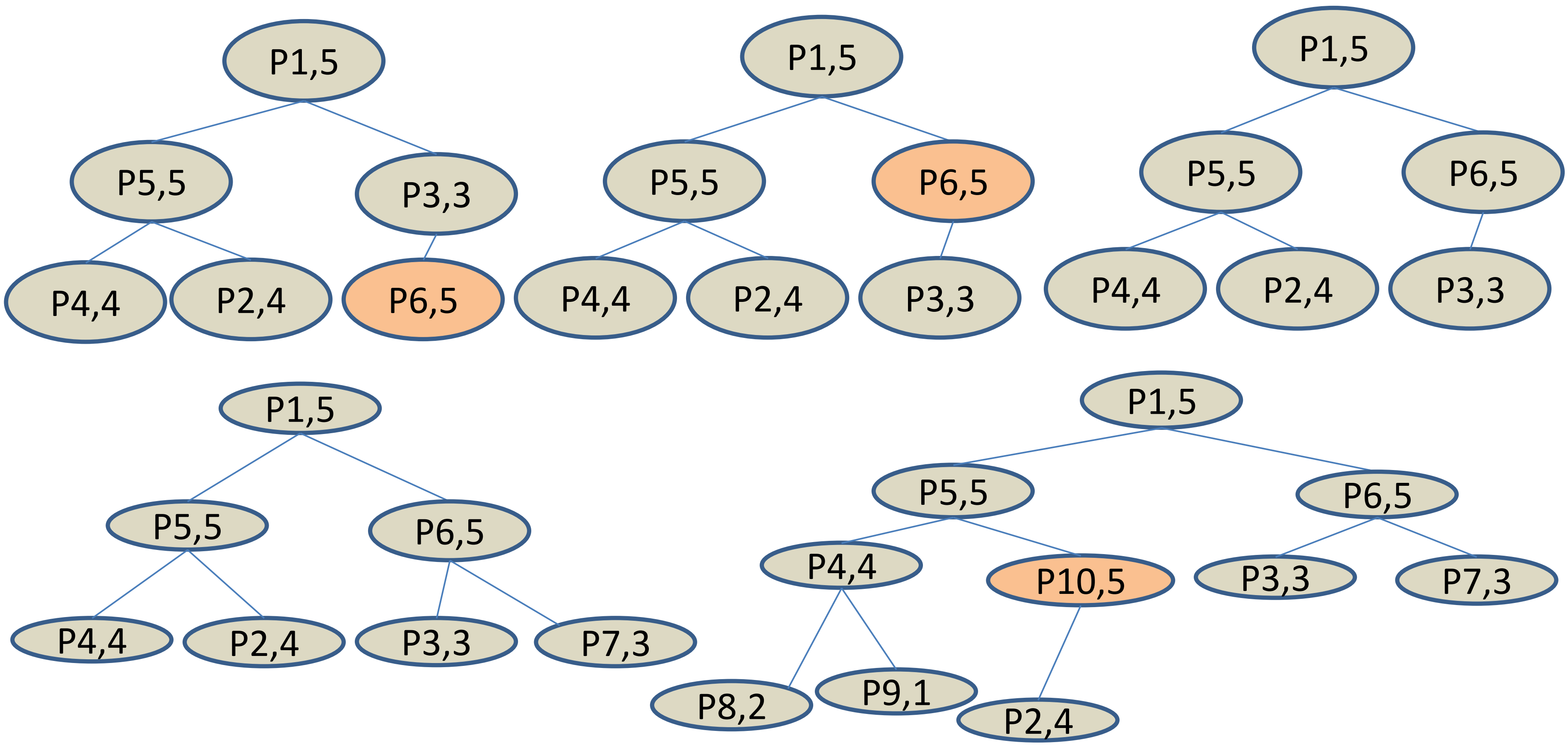
Application: Building Priority Queue

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
5	4	3	4	5	5	3	2	1	5

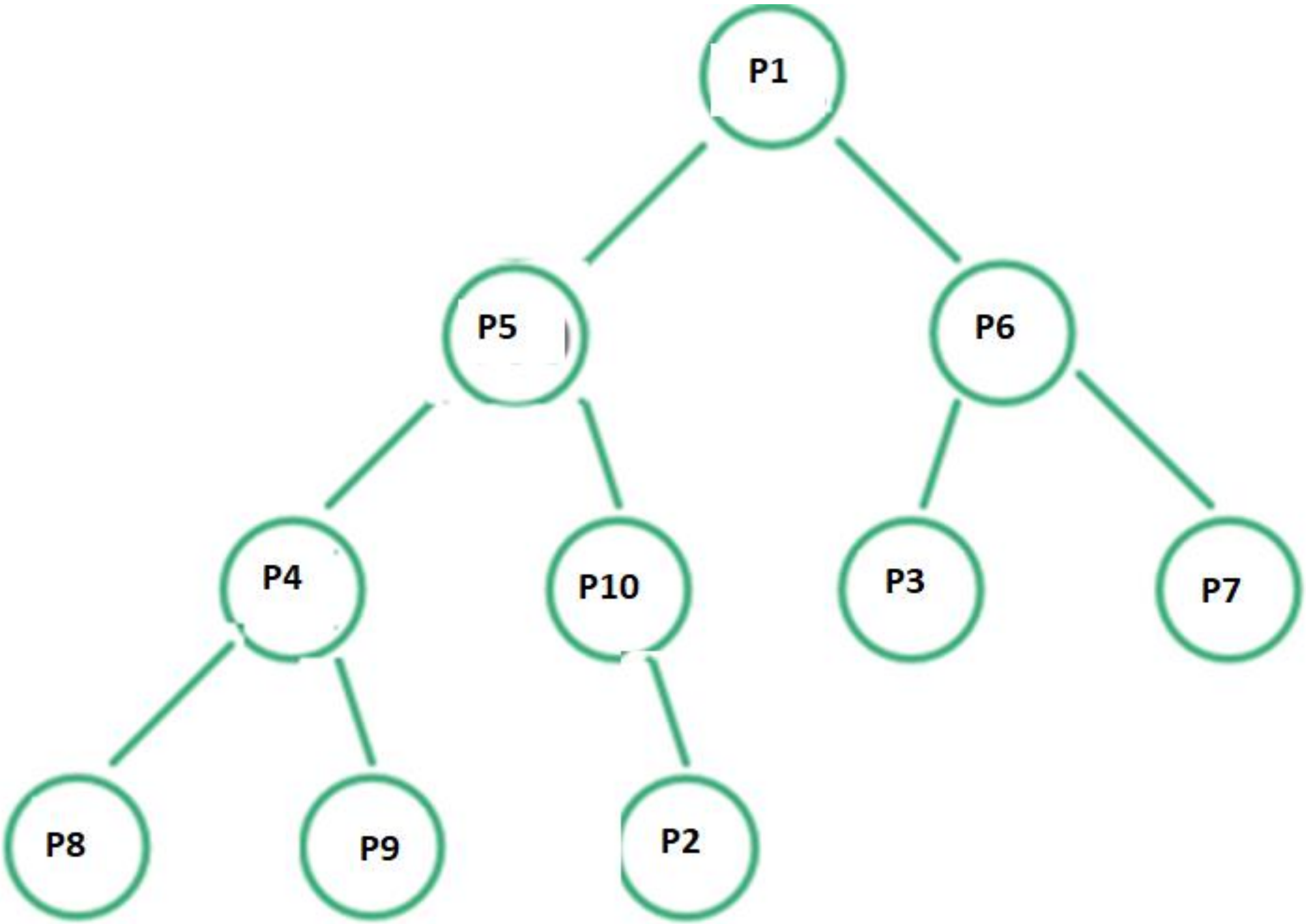


Application: Building Priority Queue

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
5	4	3	4	5	5	3	2	1	5



Applications: Implementation of Priority Queue



References

1. Algorithms Design: Foundations, Analysis and Internet Examples Michael T. Goodrich, Roberto Tamassia, 2006, Wiley (Students Edition)
2. Data Structures, Algorithms and Applications in C++, Sartaj Sahni, Second Ed, 2005, Universities Press
3. Introduction to Algorithms, TH Cormen, CE Leiserson, RL Rivest, C Stein, Third Ed, 2009, PHI

Any Question!!

Thank you!!

BITS Pilani
Hyderabad Campus

Solve Recurrence Relation Using Substitution

Two Steps

Step1

Guess the form of the solution

Step2

Use Mathematical Induction to find the constants and show that the solution works

This method works only for the cases where it is easy to guess the form of the solution

This method can be used to establish either upper or lower bounds on a recurrence

Solve Recurrence Relation Using Substitution

Example1: $T(n) = 2T(floor(n/2)) + n$ Guess: $T(n) <= cnlgn$, because the recurrence looks like $T(n) = 2\ T(n/2) + \theta(n)$ and this recurrence has a solution as $O(nlgn)$.
<u>Prove:</u> need to find an appropriate value of the constant 'c>0'.
First, assume that the bound holds for floor(n/2). So, $T(n/2) <= c\ floor(n/2)lg(floor(n/2))$.
Substituting this in the main equation, $T(n) <= 2(c\ floor(n/2)lg(floor(n/2)) + n$
$<=cnlg(n/2) +n$ $=cnlgn – cnlg2 + n$ $=cnlgn – cn + n$
$<=cnlgn$, **holds for c>=1**

Solve Recurrence Relation Using Substitution

Example: $T(n) = 2T(\text{floor}(n/2)) + n$

Guess: $T(n) \le cn\lg n$, because the recurrence looks like $T(n) = 2T(n/2) + \theta(n)$ and this recurrence has a solution as $O(n\lg n)$.

Prove: need to find an appropriate value of the constant 'c>0' and $n \ge n_0$.

First, assume that the bound holds for floor(n/2).

So, $T(n/2) \le c\,\text{floor}(n/2)\lg(\text{floor}(n/2))$.

Substituting this in the main equation, $T(n) \le 2(c\,\text{floor}(n/2)\lg(\text{floor}(n/2)) + n$

$\le cn\lg(n/2) + n$

$= cn\lg n - cn\lg 2 + n$

$= cn\lg n - cn + n$

$\le cn\lg n$, **holds for c>=1**

Second, apply induction to show that our solution holds for boundary cases on n.

Basically the boundary case is the base case for inductive proof.

This is sometimes tricky.

Lets consider that T(1) = 1, find the value of c and n.

But, when n = 1, we end up having $T(1) \le c1\lg 1 = 0$, does not hold.

So, take n0 as 4 and hence n>=4.

Check if the condition holds.

So, c>1 and n0=4 and the solution finally proved.

Solve Recurrence using Substitution

Example 2: Consider T(n) = T(floor(n/2)) + T(ceil(n/2)) + 1. Guess: T(n) <= cn and apply in the equation. So, T(n) <= c floor(n/2) + c ceil(n/2) + 1
Simplify, T(n) = cn + 1
So, it does not satisfy T(n) <= cn, for any c
However, T(n) <= cn^2 → T(n) is O(n^2). Is this a tight bound?
In fact, T(n) = O(n), **How to prove?**

Consider T(n) = T(floor(n/2)) + T(ceil(n/2)) + 1. New Guess: T(n) <= **cn - b** , **b>= 0,** and apply in the equation. So, T(n) <= (c floor(n/2) –b) + (c ceil(n/2) – b) + 1
Simplify, T(n) = (cn – 2b) + 1
So, it does not satisfy T(n) <= cn - b, for any c and b>=1
Now we achieve the prove, T(n) = O(n)

Solve Recurrence using Substitution

innovate achieve lead

Example 3: Consider T(n) = 2T(floor(n/2)) + n
Guess: T(n) <= cn and apply in the equation.
So, T(n) <= 2 (c floor(n/2)) + n

Simplify, T(n) <= cn + n

Wrong conclusion that T(n) is O(c'n), for any c'=c+1

It is wrong because we have not proved the exact form of the inductive hypothesis, i.e., T(n) <= cn

Example 4: Consider T(n) = 2 T(floor(root(n))) + lg n
Looks difficult, but lets apply some change in the variables.
Consider m = lg n → n = 2^m

So, T(2^m) = 2 T(2^(m/2)) + m

Further assume that S(m) = T(2^m) → S(m/2) = T(2^(m/2))

Thus, S(m) = 2S(m/2) + m → this we have already solved before.

S(m) is O(m lg m)

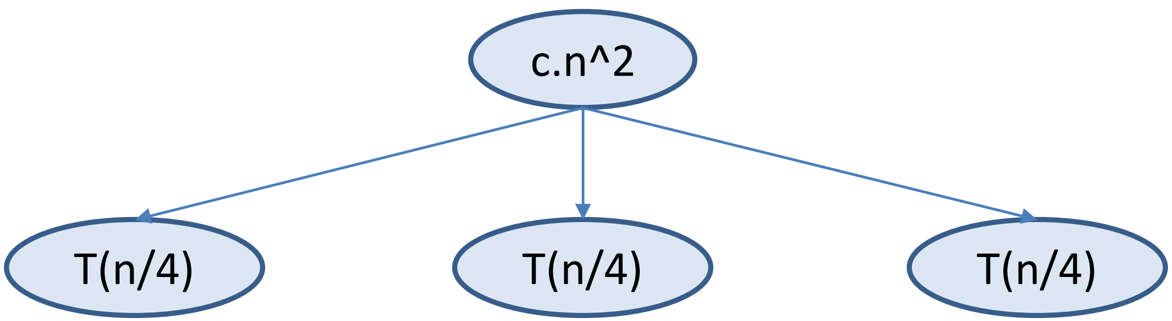
Bringing S(m) back T(n) → T(n) = T(2^m) = S(m) = O(m lg m) = O(lg n lg lg n)

Solve Recurrence by Tree Method

Recursion tree can generate good guesses that can then be verified using substitution method.

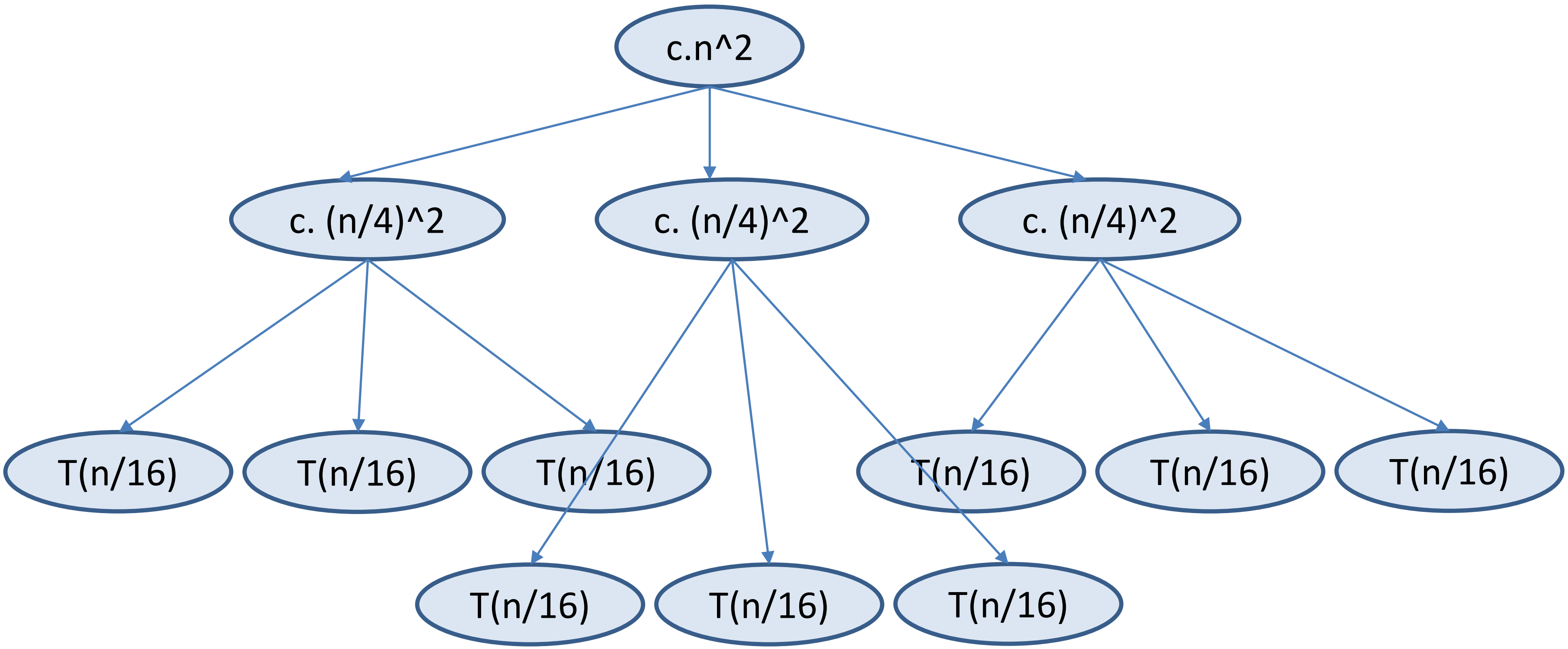
Let us take $T(n) = 3T(n/4) + \theta(n^2)$

In a recursion tree, root is the f(n) and the child nodes are the terms related to T(n/k).



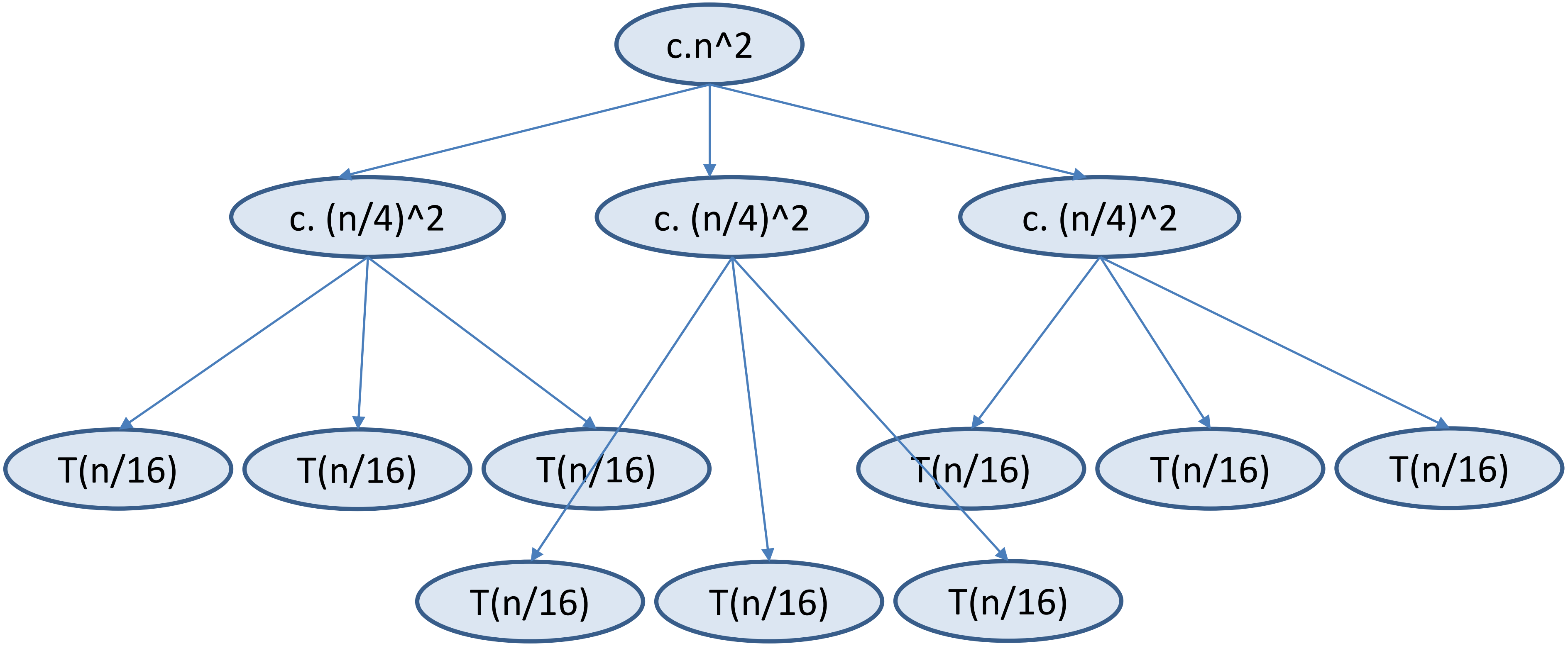
Expand each of the child node using the formula by substituting n with n/4.

Solve Recurrence by Tree Method



Continue expanding the tree till the leaf node, i.e., the node that achieves T(1) computation time → $n/4^i = 1$ → $i = \log_4 n$

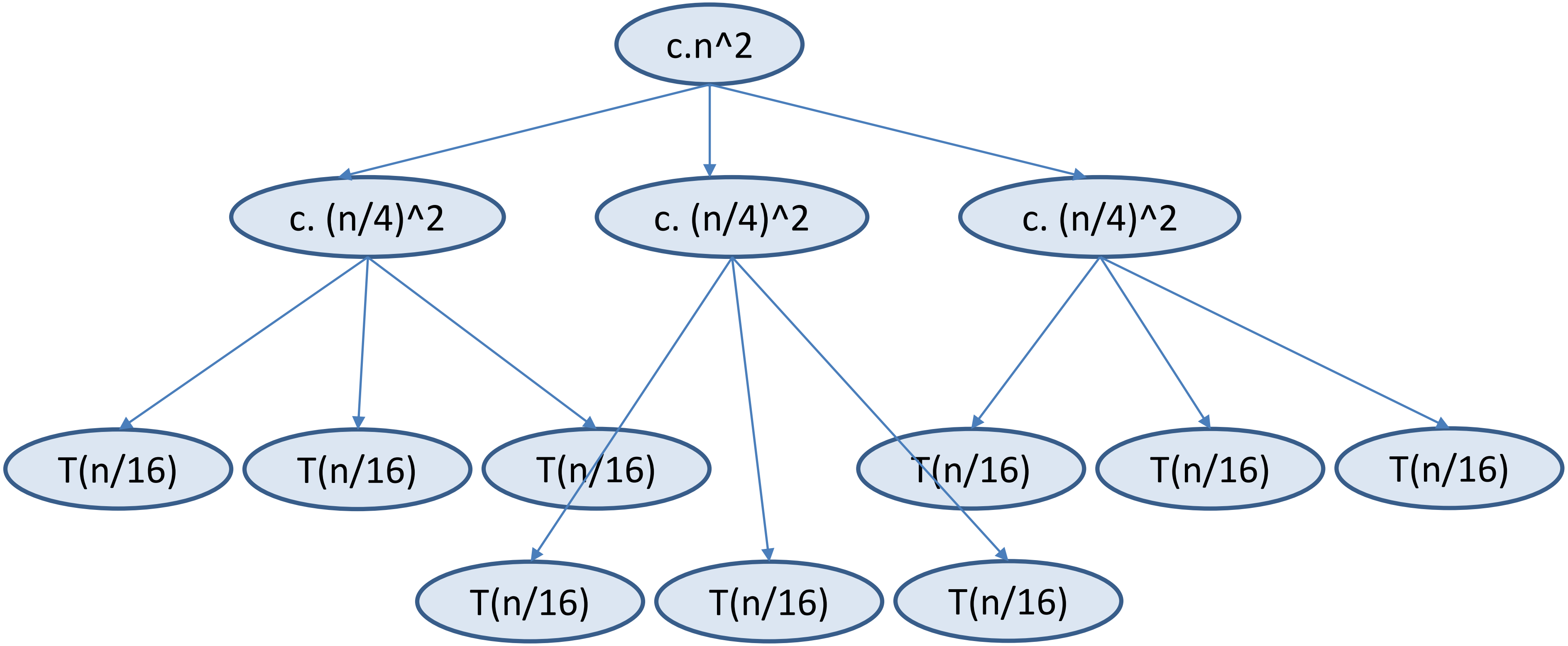
Solve Recurrence by Tree Method



So, the levels vary from i = 0, 1, 2, …, $\log_4 n$

Also, at each level i, #nodes = 3^i and these levels vary from 0 to ($\log_4 n$ - 1) because last level contains the leaf nodes

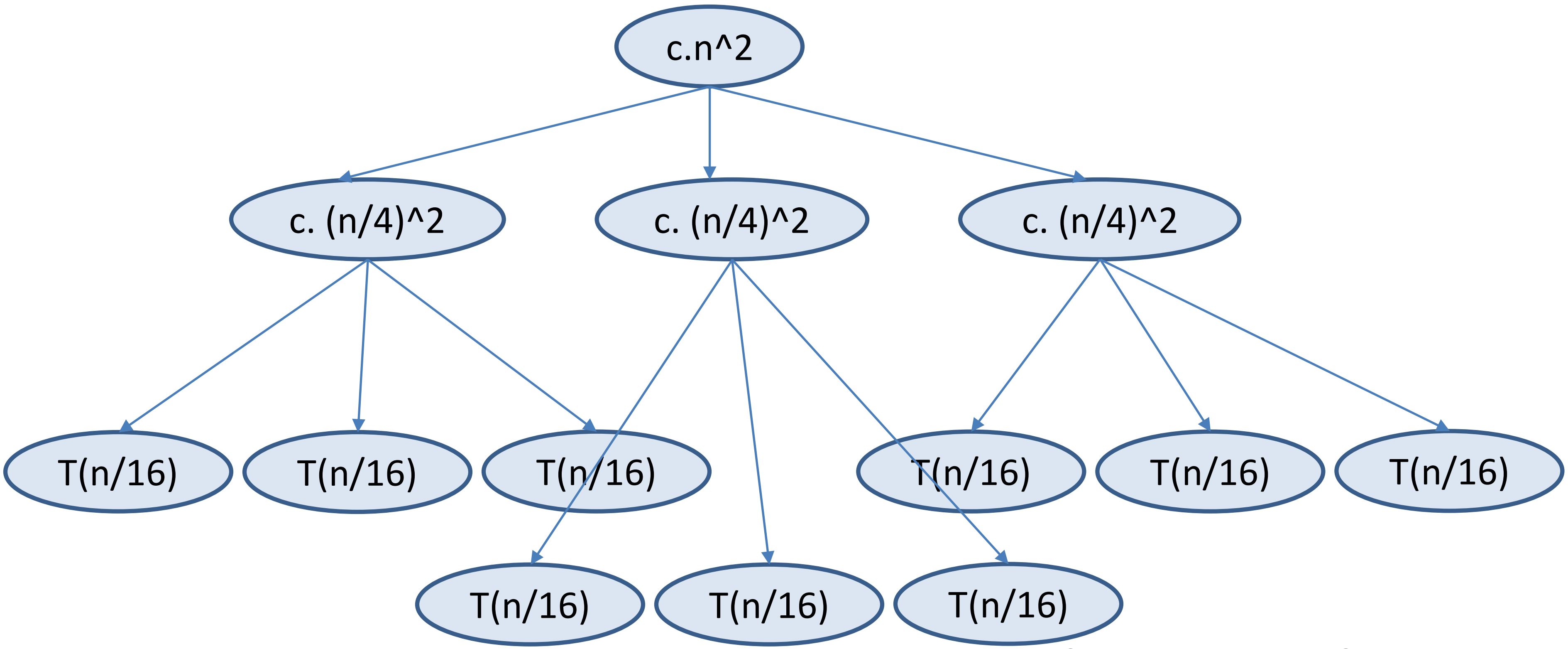
Solve Recurrence by Tree Method



Now, at a particular level i, the total cost = 3^i * (c * (n/4^i)^2) = (3/16)^i*c*n^2

At the last level, we have 3^($\log_4 n$) nodes with each node have cost of T(1) → total cost at last level = θ(n^($\log_4 3$))

Solve Recurrence by Tree Method



Adding cost across all the levels, $T(n) = cn^2 + (3/16)cn^2 + (3/16)^2cn^2 + \ldots + (3/16)^{\log 4n - 1}cn^2 + \theta(n^{(\log_4 3)})$

$=(16/13)cn^2 + \theta(n\hat{}(\log_4 3) = O(n^2)$

Solve Recurrence by Tree Method

Lets go back to our recurrence $T(n) = 3T(n/4) + \theta(n^2)$

The guess is $T(n) = O(n^2)$, i.e., $T(n) = cn^2$

Use this guess to prove the solution using substitution method.

$T(n) <= 3\ T(\text{floor}(n/4)) + cn^2$

$<= 3\ d\ (\text{floor}(n/4))^2 + cn^2$

$<= 3\ d\ (n/4)^2 + cn^{2,}$ take away floor()

$= 3/16\ d\ n^2 + cn^2$

$<= dn^2$, where $d >= (16/13)c$, (16/13) can obtained from the solution of recurrence tree