Massey University
COLLEGE OF SCIENCES

159201

Week 5

Summer 2014

Te Kunenga ki Pūrehuroa

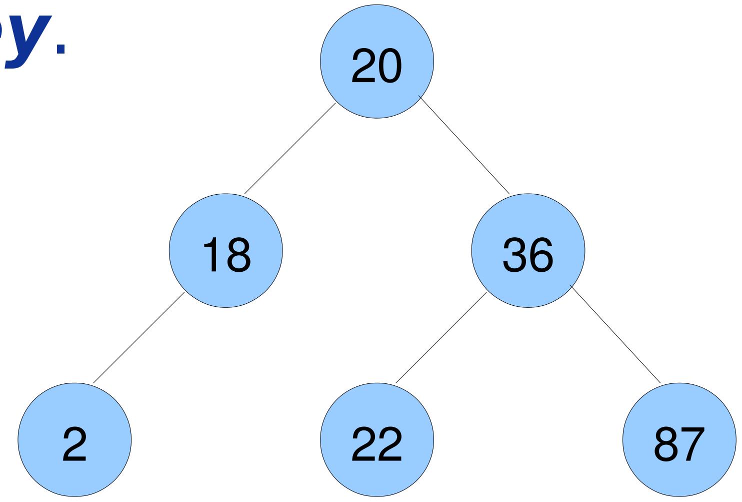
Massey University
COLLEGE OF SCIENCES

L 17

Te Kunenga
ki Pūrehuroa

Binary Search Trees

A ***binary search tree*** (BST) holds records where each record has a ***unique key***.



There are two rules for a binary search tree:

- A binary search tree is a binary tree.
- For every node:
  - All keys to the left are smaller and all keys to the right are bigger

Operations on Binary Search Trees

Insert() – how would we insert 25 into this tree?

Search() – similar to insert(). Note that a search can end in success or failure.

Search for 2 in this tree.

Search for 56 in this tree.

Sort() - use an in-order traversal to produce keys in sorted order.

Delete() - there are three situations to consider:

- The node is a leaf: delete the node (e.g. delete 2)
- The node has one child: replace the node with the child and delete the child(e.g., delete 18)
- The node has two children: see next lecture…

Massey University
COLLEGE OF SCIENCES

C++ function to insert new key

```
void Tree::Insert(int item) {
// BST must already contain at least one item
  if (item > data) {  // move to the right
     if (rightptr == NULL) {
        rightptr = new Tree(item, NULL, NULL);
     } else {
        rightptr->Insert(item);
     }
  } else if (item < data) {  // move to the left
     if (leftptr == NULL) {
        leftptr = new Tree(item, NULL, NULL);
     } else {
        leftptr->Insert(item);
     }
  } else if (item == data) {  // should be unique
     printf("Error: key is not unique");
     exit(9);
  }
}
```

Te Kunenga
ki Pūrehuroa

Massey University
COLLEGE OF SCIENCES

insert()

Notes:

We can use the same structure we used for the binary tree.

The first item into a binary tree:

```
    T = new Tree(item,NULL,NULL);
```

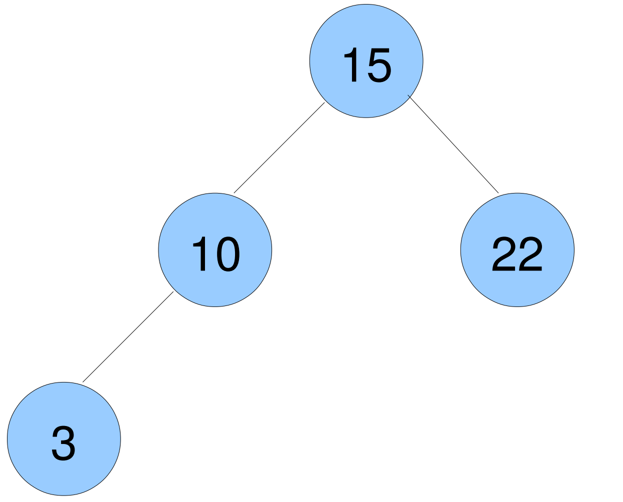
Our function inserts one C++ type (int, or float etc), but we could insert an entire record.

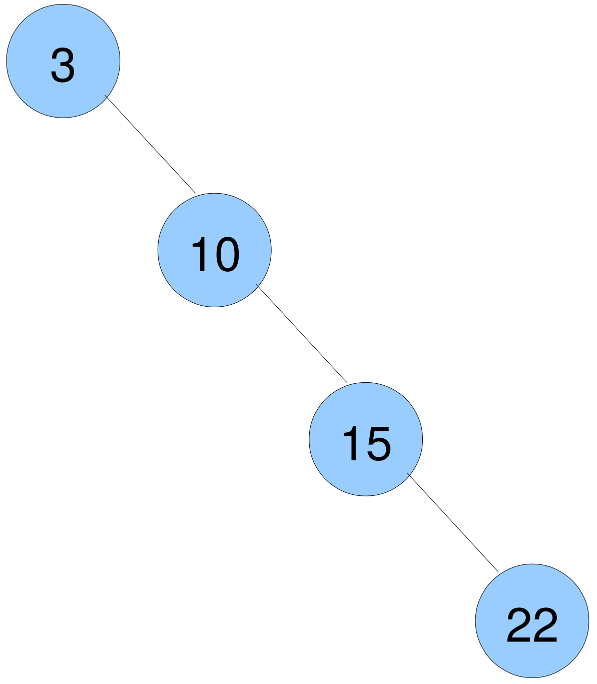
Search() would be similar. Implement it as an exercise...

Te Kunenga ki Pūrehuroa

Balance in trees

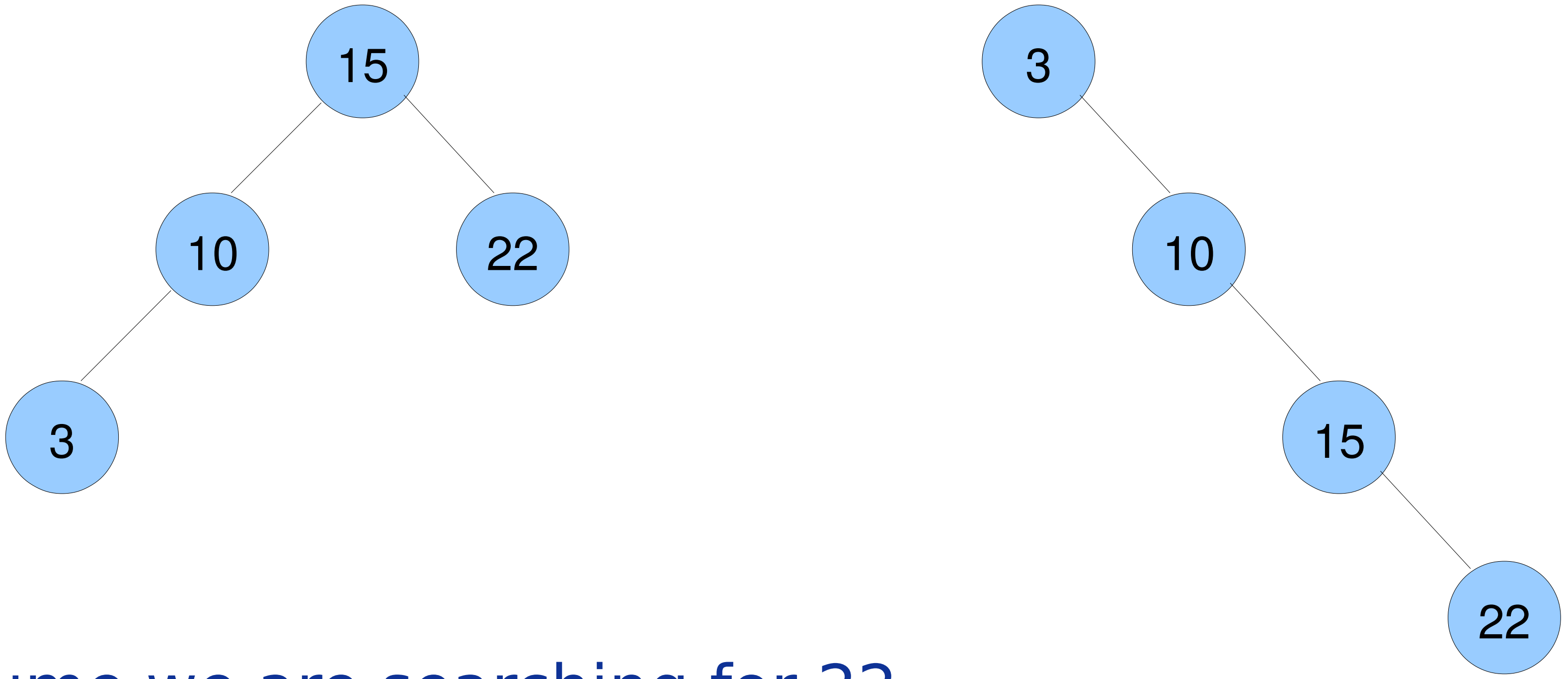
Different trees may contain the same records:





Massey University
COLLEGE OF SCIENCES

Balance in trees



Assume we are searching for 22.

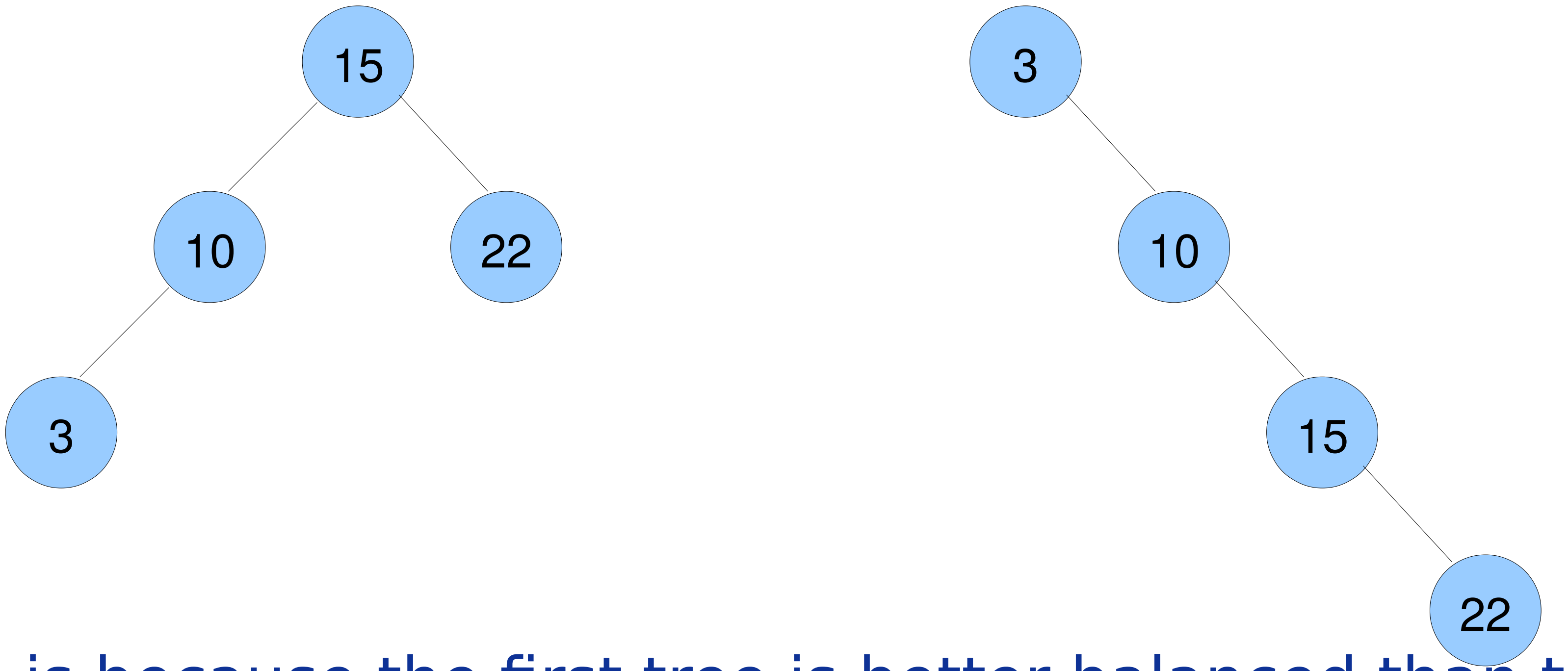
In the first tree we need to look at **2** nodes.

In the second tree we look at **4** nodes.

Which one is more efficient?

Te Kunenga ki Pūrehuroa

Balance in trees



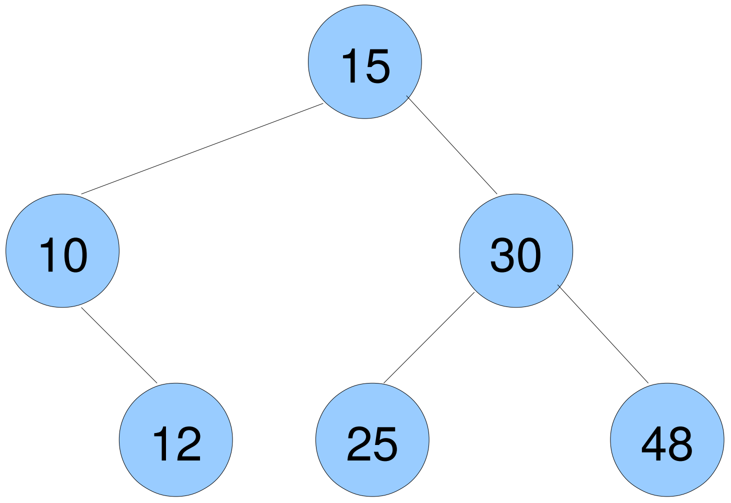
This is because the first tree is better balanced than the second one.

Although there is no precise definition of balance, we can figure out ways of finding which one is better.

E.g., we can count levels, or carry out a search etc.

Massey University
COLLEGE OF SCIENCES

Binary Search Trees: Delete

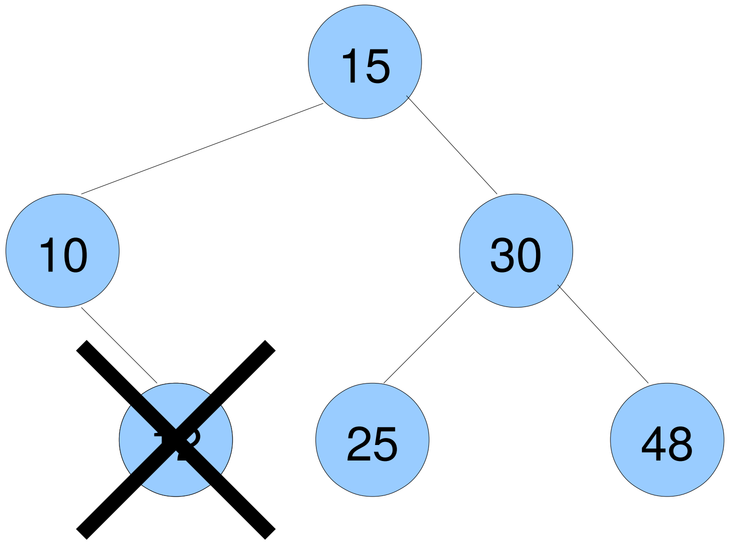


When deleting a node from a BST, it has three possible states:

Te Kunenga ki Pūrehuroa

Massey University
COLLEGE OF SCIENCES

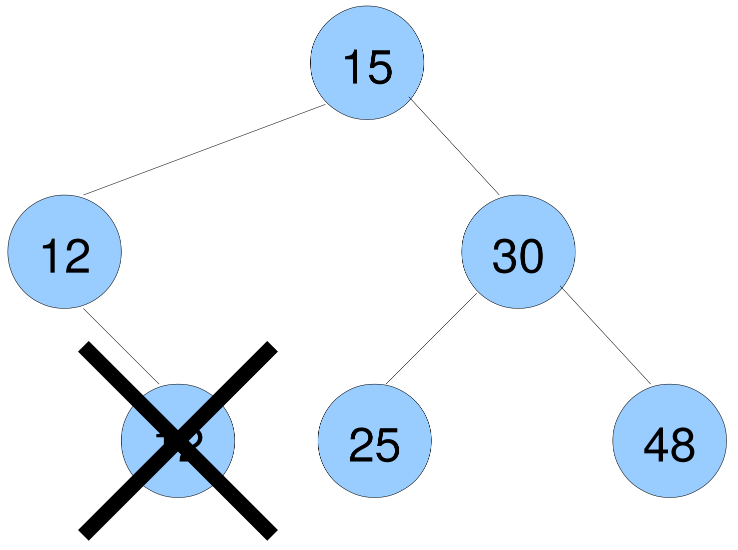
Binary Search Trees: Delete



1) Node is a leaf: e.g. 12

just delete the node and modify 10 ->right=NULL

Binary Search Trees: Delete

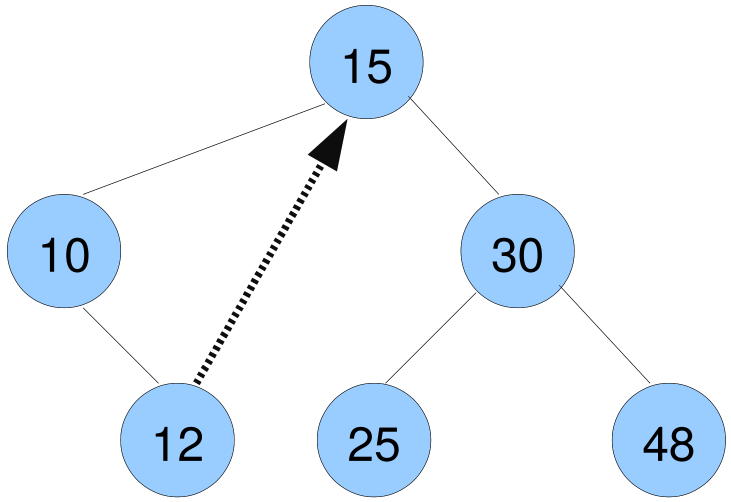


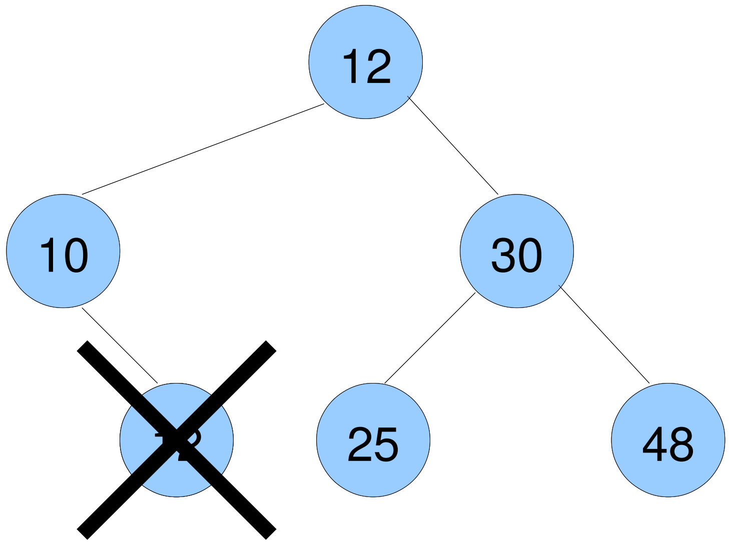
2) Node has one child, e.g. 10.

Replace the node with the child,

delete original child (and update pointer to right)

Binary Search Trees: Delete





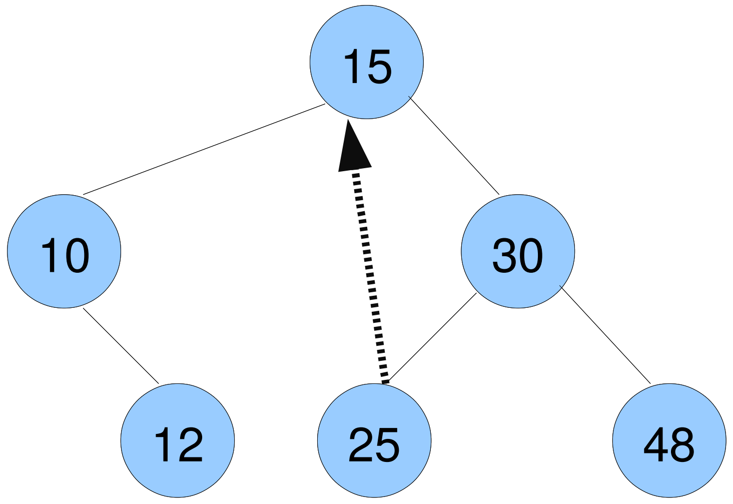
3) Node has two children (e.g., 15).

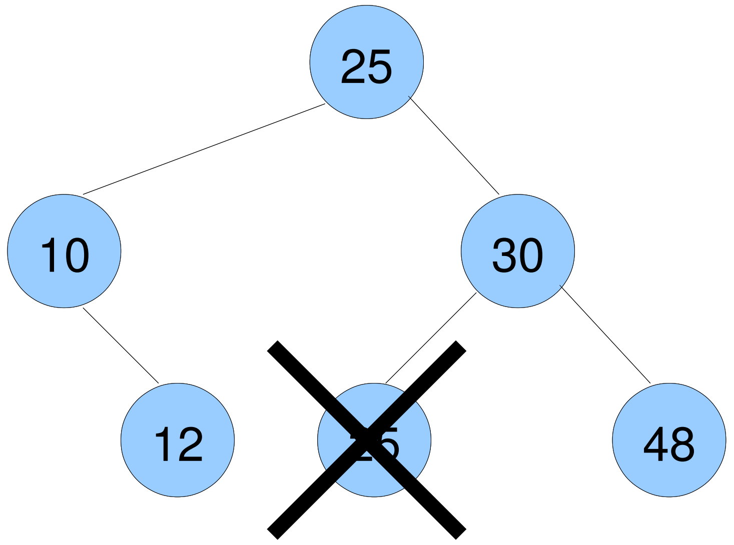
Replace the node with ***either***

i) **Largest** value from the **left** subtree

Or

Binary Search Trees: Delete

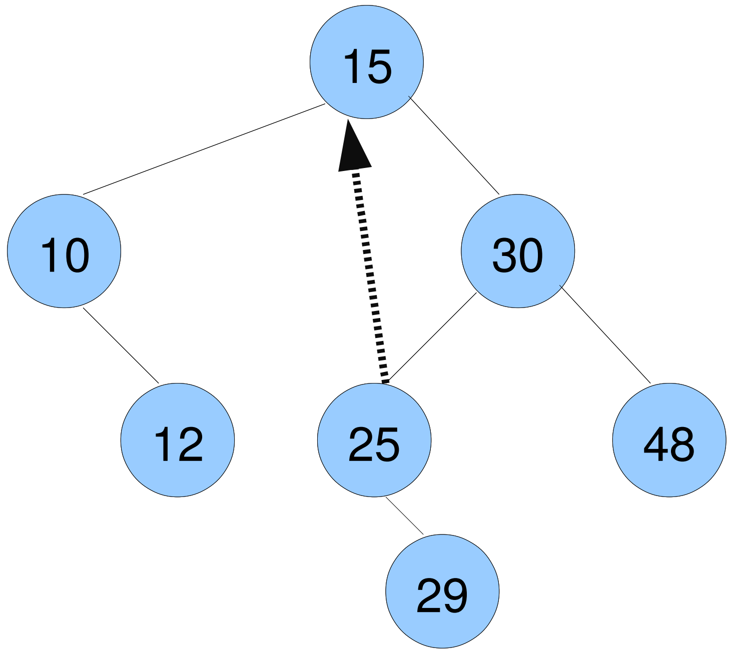


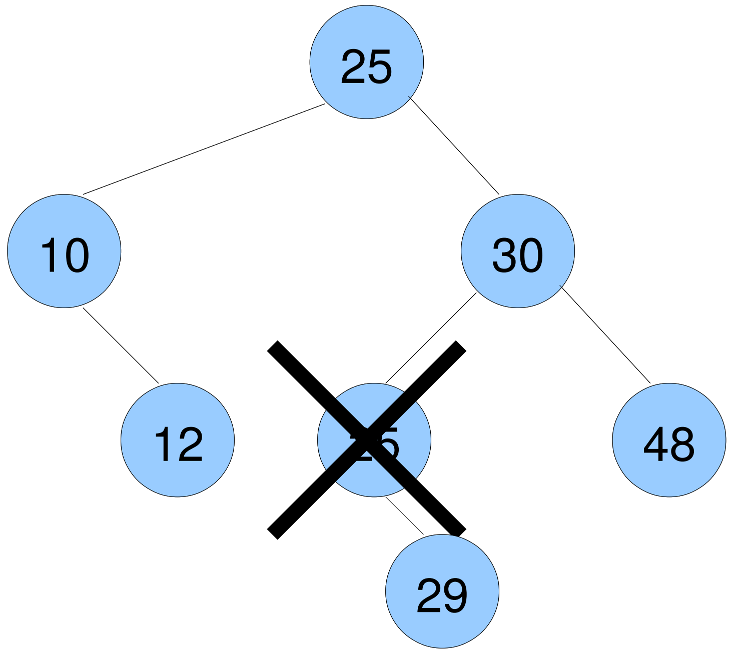


3) Node has two children (e.g., 15).

ii) **Smallest** value from the **right** subtree

Binary Search Trees: Delete





3) Node has two children (e.g., 15).

But what if 25 is not a leaf? Still copy 25 to 15, delete original 25, update pointer 30->left

Massey University
COLLEGE OF SCIENCES

Binary Search Trees: Delete

For option 3), which one is the best?

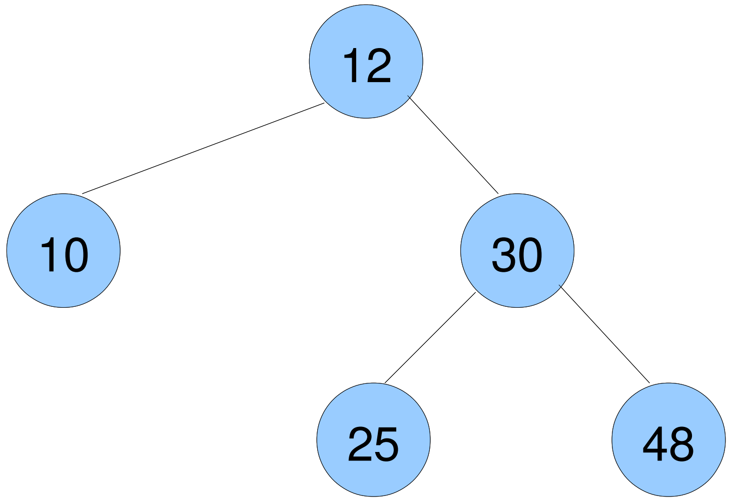
The one that results in the most balanced tree...

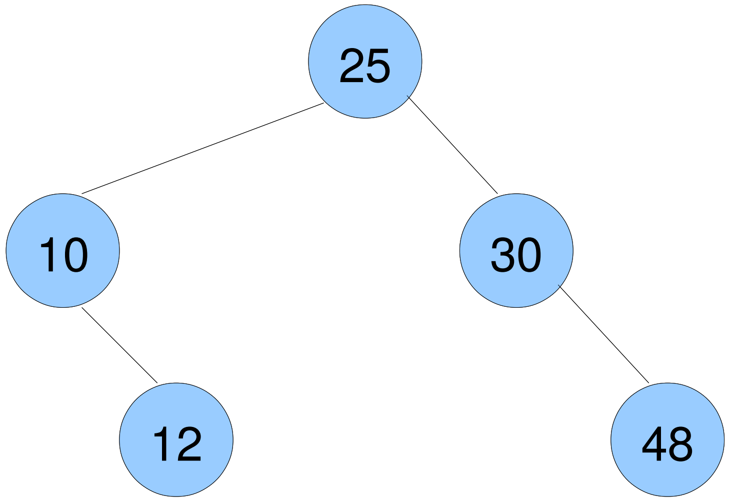
We can check for balanced states by comparing the left and the right subtrees.

One can count the nodes on each subtree...

Te Kunenga ki Pūrehuroa

Binary Search Trees: Delete



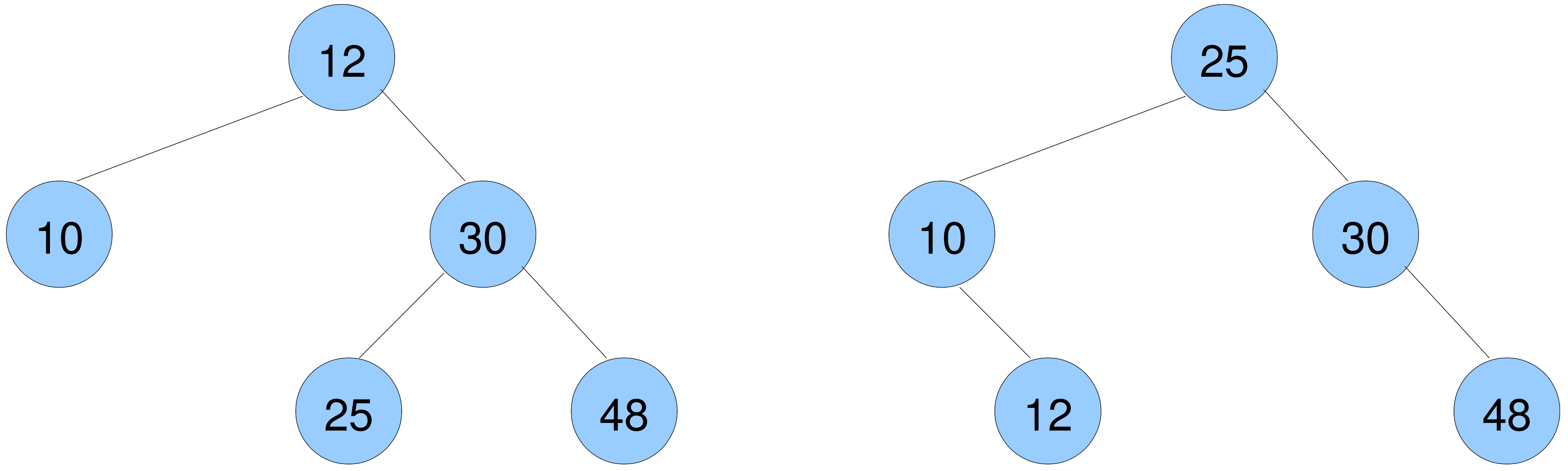


i) ii)

i) has 1 left node, 3 right nodes

ii) has 2 left nodes, 2 right nodes: clearly better

Binary Search Trees: Delete



Other criteria:

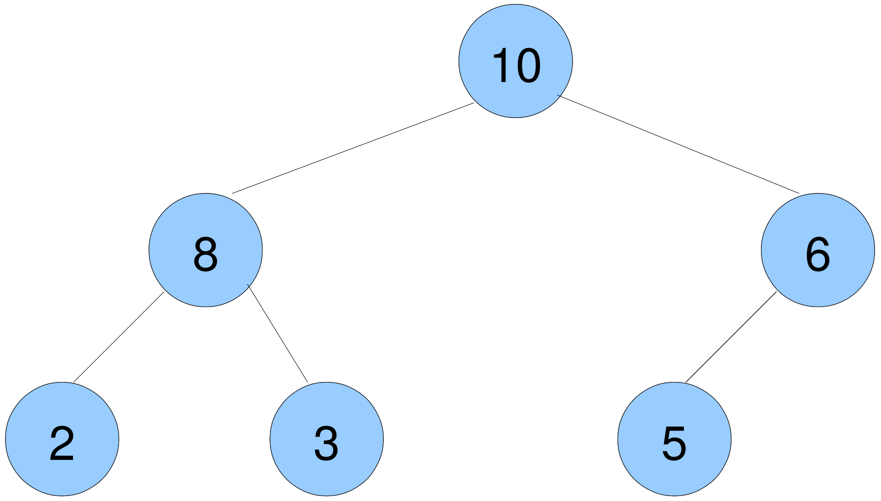
Count the levels (height) of the left and right subtrees

Massey University
COLLEGE OF SCIENCES

L 18

Te Kunenga
ki Pūrehuroa

Heaps

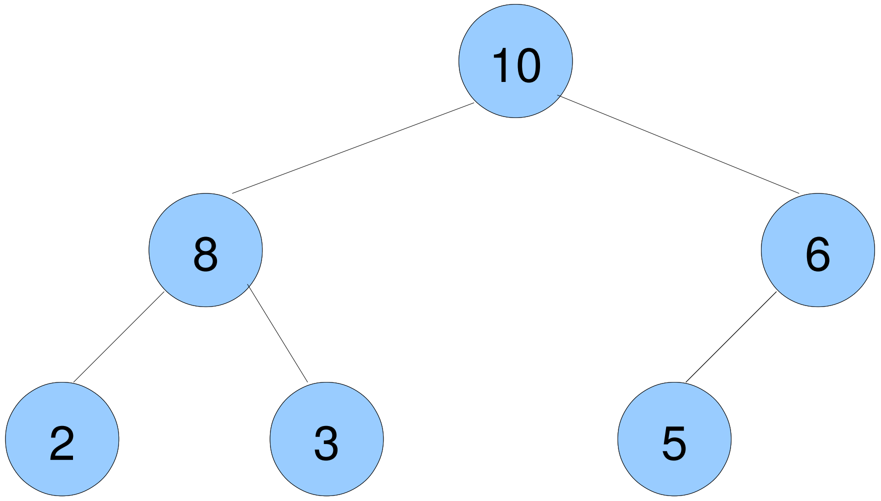


Heaps are used in algorithms where these rules help… e.g.:

Sorting

Priority queues etc…

Heaps



Heaps are trees in which the root key is larger than the keys of its children (and so on and so forth for the subtrees...).

An advantage of Heaps is that elements can be deleted or inserted very efficiently.

Massey University
COLLEGE OF SCIENCES

Heaps

A heap is a binary tree (**not** a binary search tree) such that:

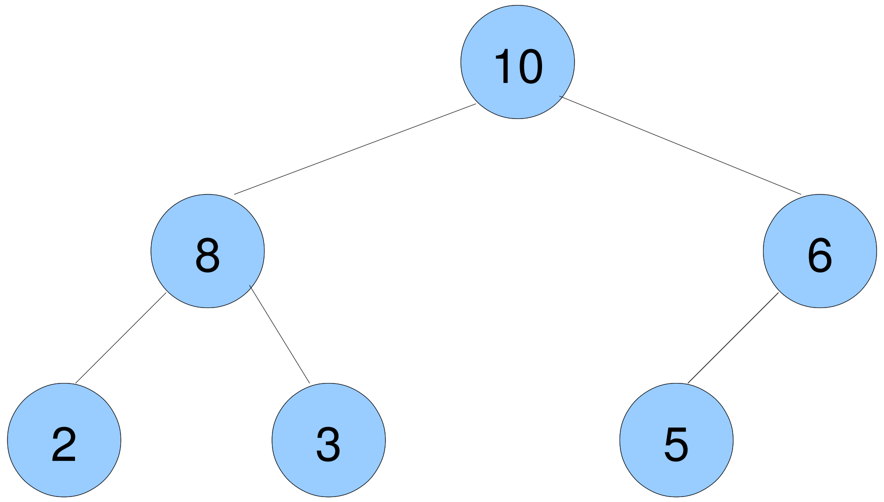
a) the value of any node ≥ values of any of its children

b) the tree is perfectly balanced (best that you can do)

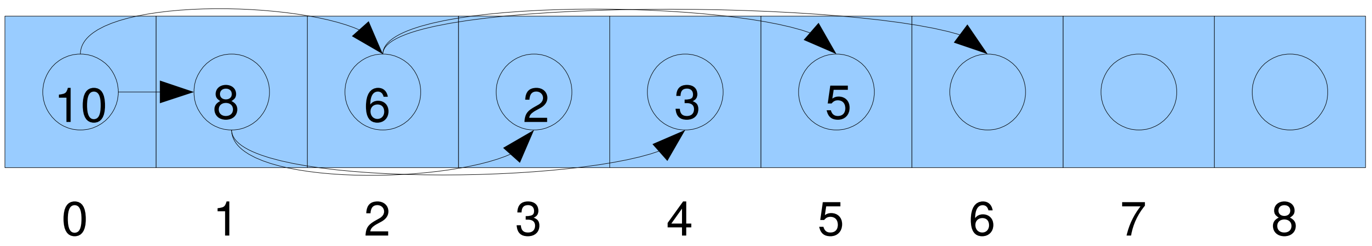
c) any new node is inserted “as far to the left as possible”

Te Kunenga ki Pūrehuroa

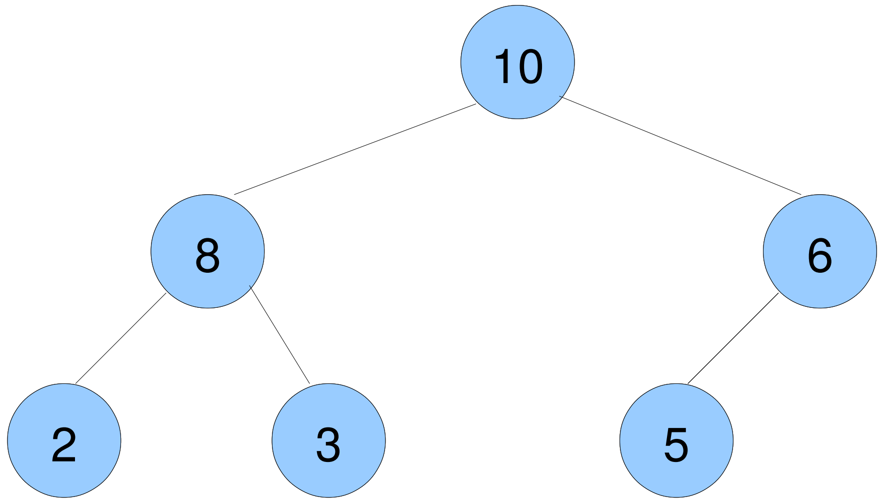
Heaps implementation



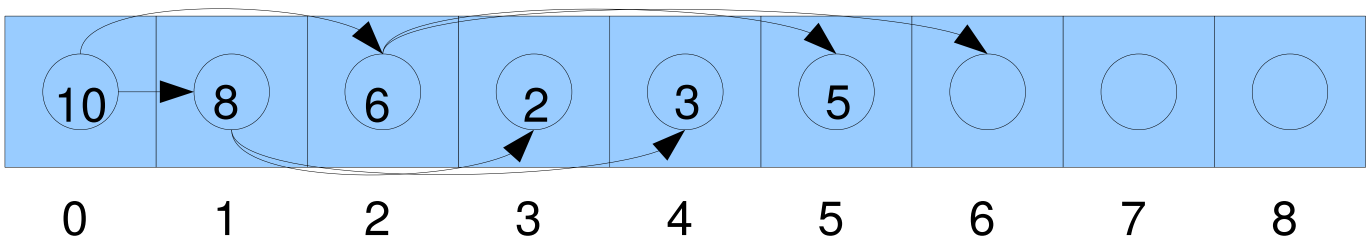
Rules b) and c) enable the translation betweeen a heap and an array (or vector).



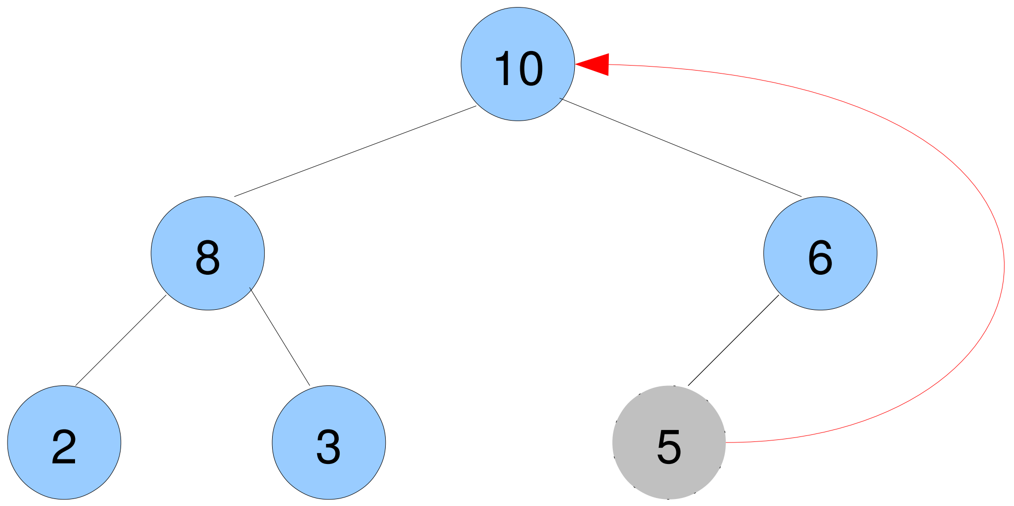
Heaps: sorting algorithms



The values [10, 8, 6] are larger than [2,3,5]



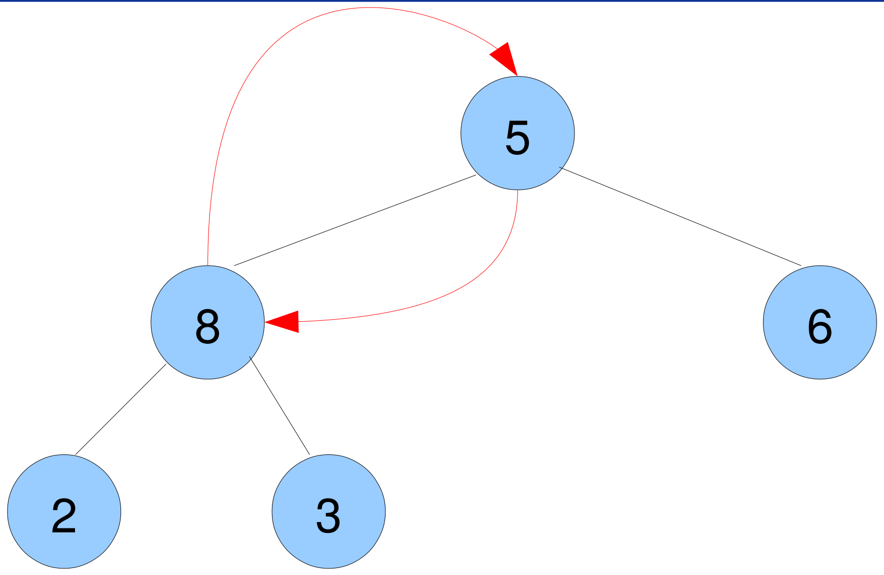
Heaps: deleting



Suppose we want to delete 10.

Use 5 to replace 10, delete the node.

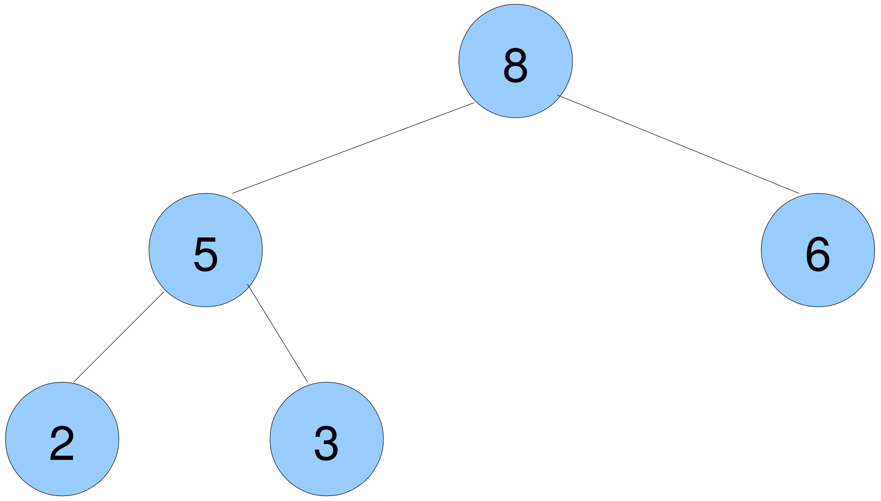
Heaps: deleting



Swap the contents until it is a Heap again

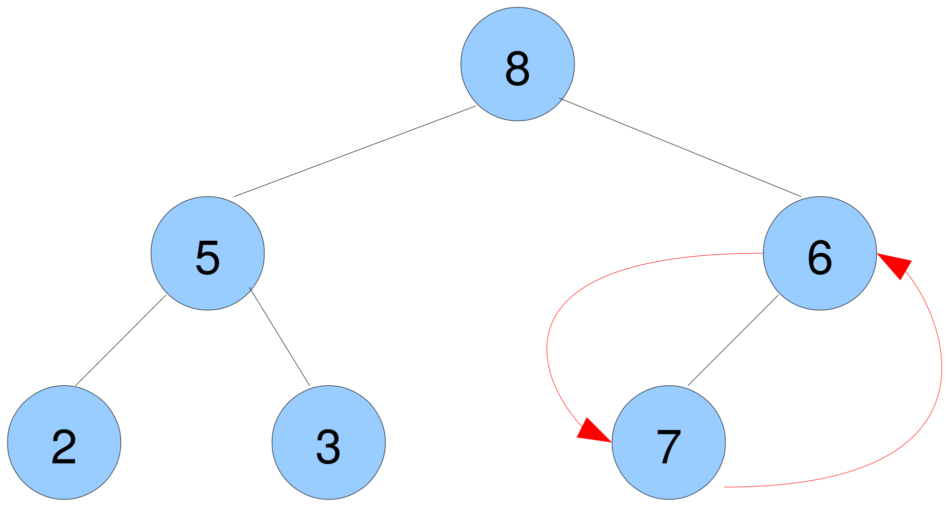
(if instead of 5 we had 1, we would continue to swap)

Heaps: adding



The opposite process, we add to the next leaf and swap it until it is a Heap.

Heaps: adding



Suppose we want to add 7...

Heaps: Implementation

```
class Heap {
   private:
      int data[10];//can change that to dynamic
      int last;//last index
   public:
      Heap(){last=-1;};   // constructor, consider
data[i]=0 an empty slot
      ~Heap() { };//destructor
      void InsertHeap( int newthing);
      bool Delete(int item);
      void PrintHeap();
};
```

Massey University
COLLEGE OF SCIENCES

Heaps: Insert

```
void Heap::InsertHeap( int newthing){
  data[last+1]=newthing;//add to the end
  last=last+1;
  if (last==0) return;//only one item in Heap
  int child_index=last;
  int par_index=0;
  bool swapping=true;
  while (swapping==true ){
    swapping=false;
    if(child_index%2==0) par_index=child_index/2-1;//right
child
    else par_index=child_index/2;//left child
    if(par_index>=0){
      if(data[child_index]>data[par_index]) {
              swap(data[child_index],data[par_index]);
              swapping=true;
              child_index=par_index;
       }
      }
    }
}
```

Te Kunenga
ki Pūrehuroa

Heaps: Delete (part 1)

```
bool Heap::Delete(int valuetodelete){
  if(last<0) {
    printf("Heap is already empty\n");
    return false;
  }
  int a;
  for(a=0;a<last;a++){
    if (data[a]==valuetodelete) break;
  }
  data[a]=data[last];
  data[last]=0;//"deleting..."
  last=last-1;// subtract last
  //now fix the heap
  int parindex=a;
  int leftindex=parindex*2+1;//left child
  int rightindex=parindex*2+2;//right child
  bool swapping=true;
```

Heaps: Delete (part 2)

```
  while((data[parindex]<data[leftindex]||data[parindex]<data[rightindex])
                && swapping==true){
    swapping=false;
    if (data[rightindex]<data[leftindex]){//follow left
      swap(data[leftindex],data[parindex]);
      parindex=leftindex;
      swapping=true;
    }
    else{//else follow right
      swap(data[rightindex],data[parindex]);
      parindex=rightindex;
      swapping=true;
    }
    leftindex=parindex*2+1;    rightindex=parindex*2+2;
    if(leftindex>last) break;
    else{
      if(rightindex>last){
        if (data[parindex]<data[leftindex])
          swap(data[parindex],data[leftindex]);
           break;
        }
    }
  }
  return true;
}
```

Te Kunenga
ki Pūrehuroa

Challenge

1) Implement a Heap using **vectors** instead of arrays. A small modification in the code we explored will suffice.

2) Use the new Heap to insert and delete new nodes in order to test your code

3) Can you think of a way to use this arrangement to sort a set of integers?

Massey University
COLLEGE OF SCIENCES

L 19

Te Kunenga
ki Pūrehuroa

Massey University
COLLEGE OF SCIENCES

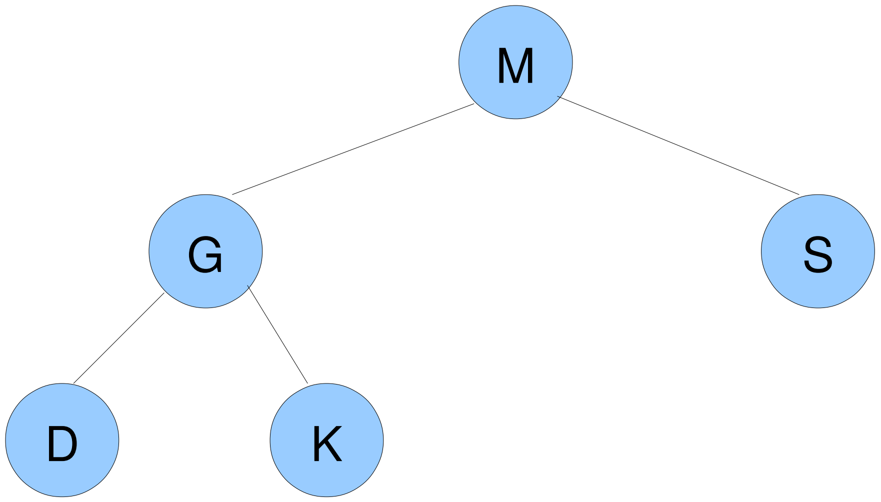
AVL Trees

AVL trees are self balancing binary search trees

They are named after Adelson-Vleskii and Landis

a) it is a ***binary search tree*** (unique keys, smaller to the left, bigger to the right)

b) for a node, the **heights** of the subtrees differ by **1 or 0**



Te Kunenga ki Pūrehuroa

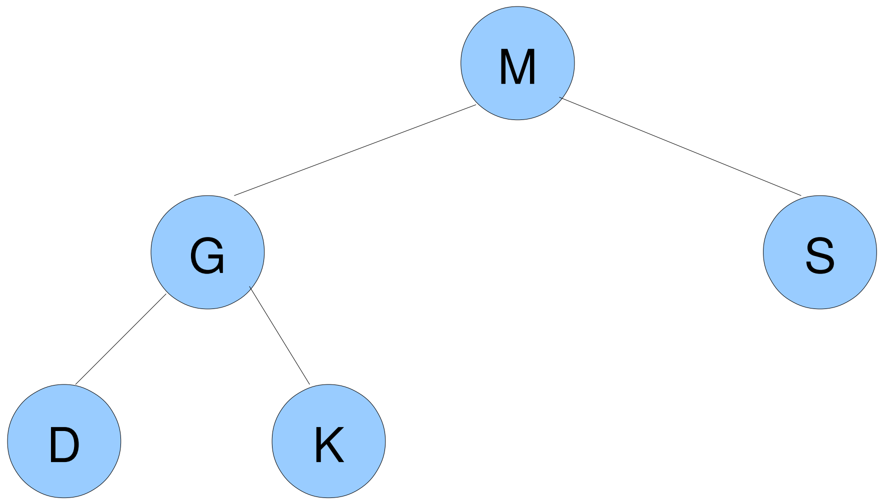
AVL Trees

Example: height of left subtree of M is 2

height of right subtree of M is 1

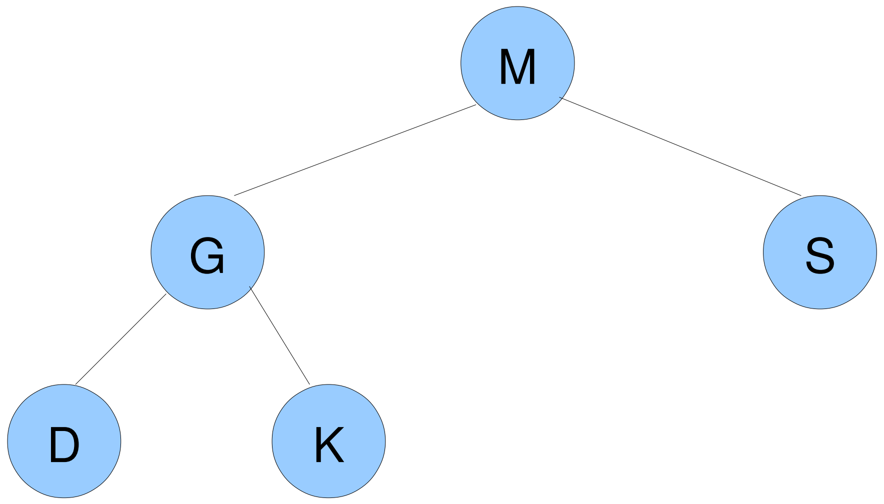
difference = 1

difference of heights of G's subtrees is 0



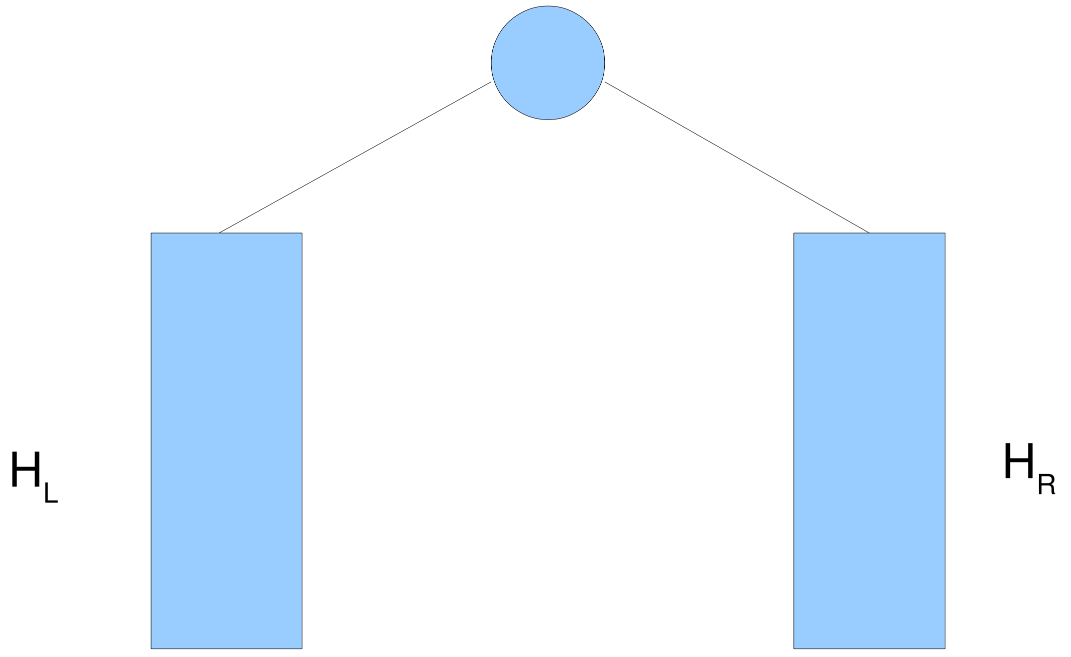
AVL Trees

All other nodes are leaves – height difference of subtrees of a leaf is always zero



AVL Trees

The diagram for a general AVL tree can be represented by:



Three possible cases: $\mathbf{H_L=H_R}$ or $\mathbf{H_L=H_R-1}$ or $\mathbf{H_L=H_R+1}$

Massey University
COLLEGE OF SCIENCES

AVL Trees

Assume that we want to add a new node, say to the *left*. Three cases arise:

new HL = HR + 1 in this case the tree is still an AVL tree

new HL = HR in this case the tree is still an AVL tree

new HL = HR + 2 in this case the tree needs to be ***rebalanced***

Two distinct cases for rebalancing

Te Kunenga ki Pūrehuroa

AVL Trees rebalancing

Case 1:

Height of left subtree of M is H + 1

Height of right subtree of M is H

If add a node:

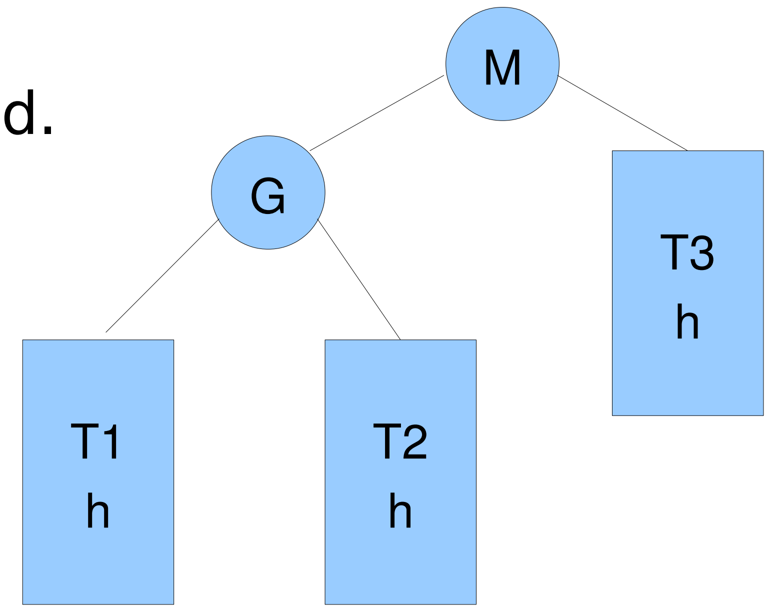
the tree needs to be rebalanced.

Note:

Prior to adding a node,

T1 and T2 and T3 had the

Same height h.

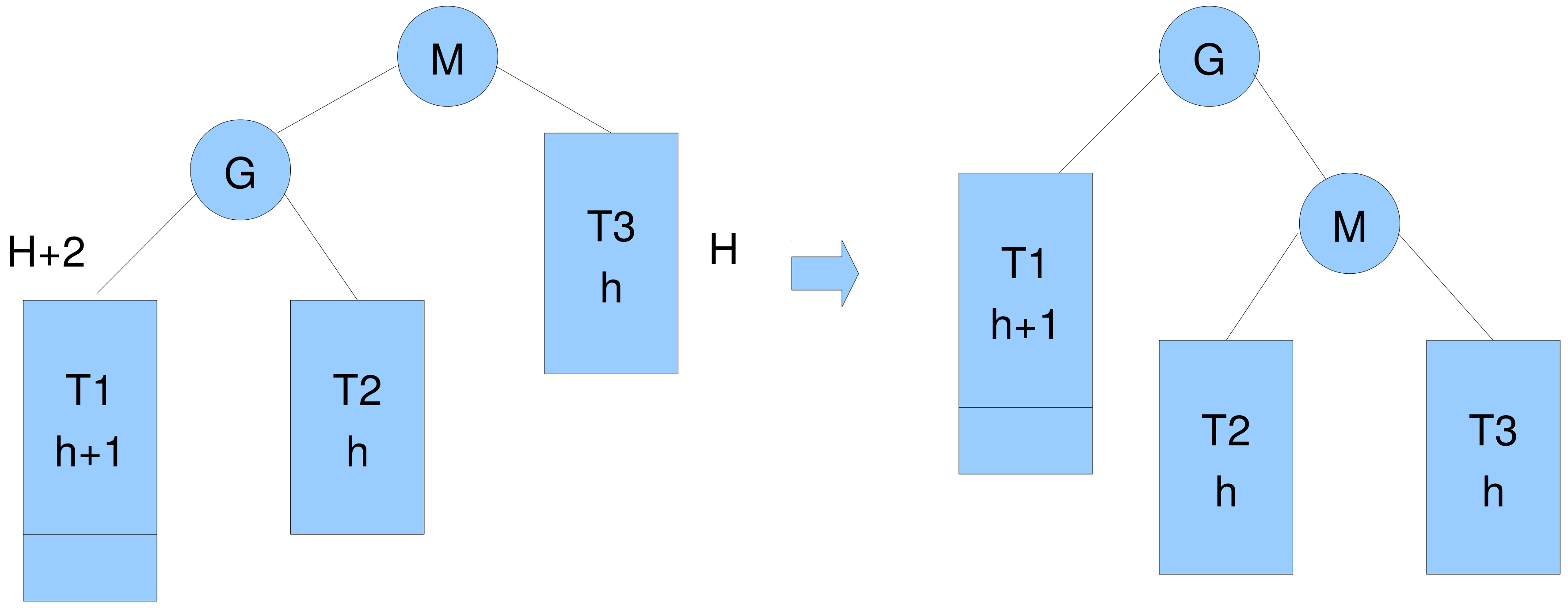


Massey University
COLLEGE OF SCIENCES

AVL Trees rebalancing

Solution for Case 1:

Make G the root (H+1 on both sides)

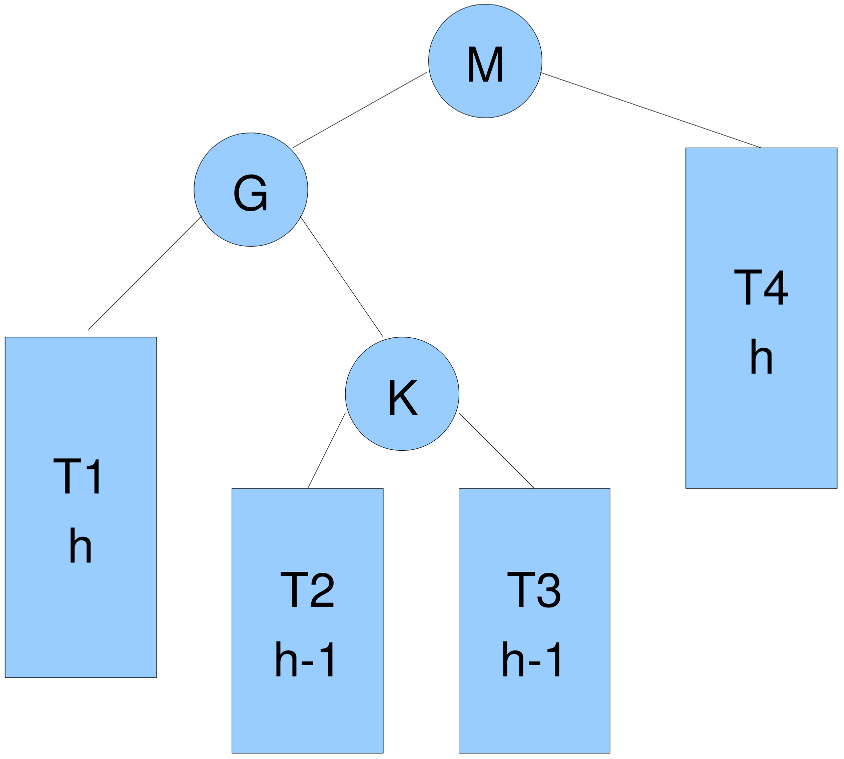


AVL Trees rebalancing

Case 2:

Add a node to T2 (or T3).

The height difference is also 1

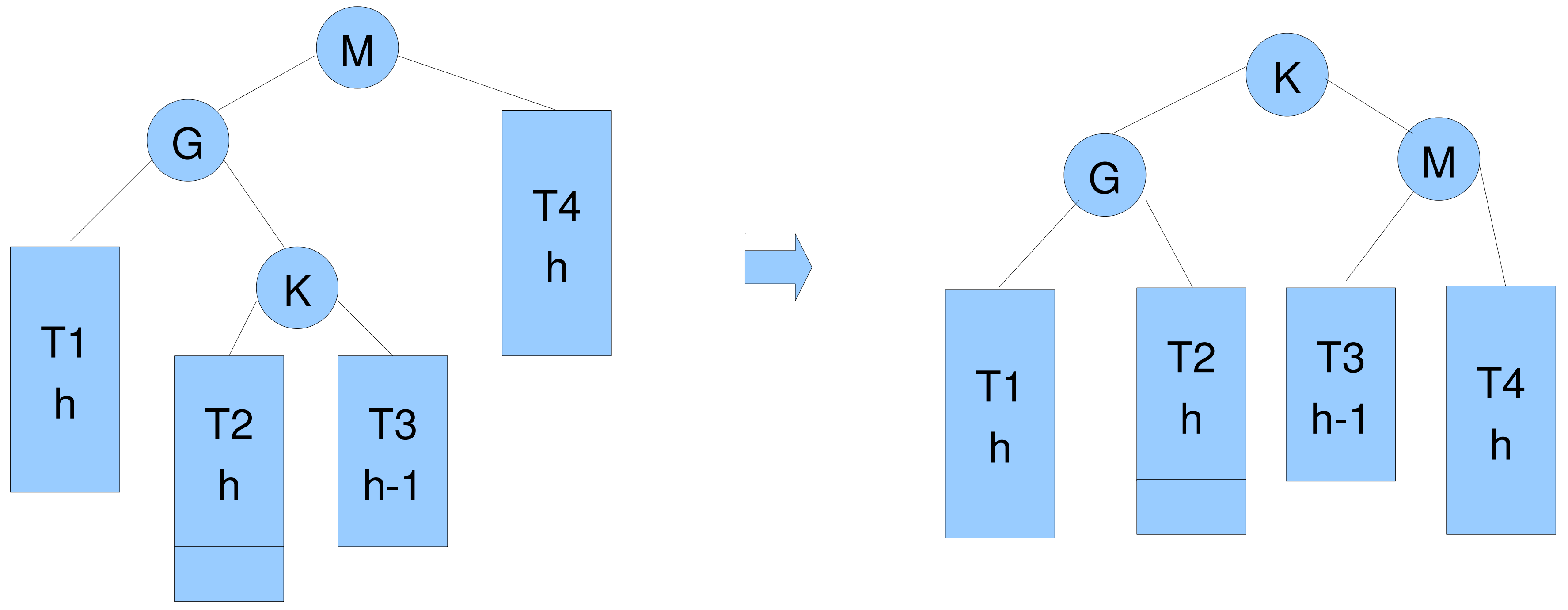


Te Kunenga ki Pūrehuroa

AVL Trees rebalancing

Solution for Case 2:

Make K the root



L 21

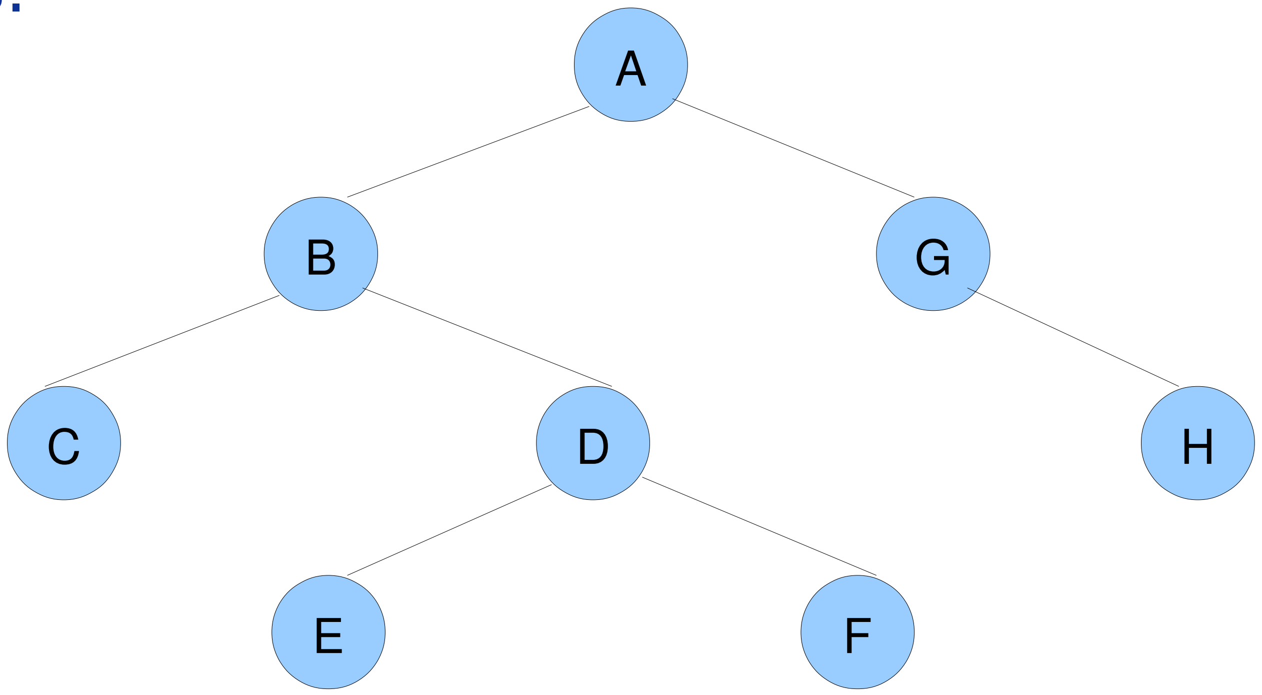
Massey University
COLLEGE OF SCIENCES

Threaded Binary Trees

Tree traversals using **recursion** are:

Slow, can cause **stack overflow**.

Another way to achieve iterative traversals is using *threads*.

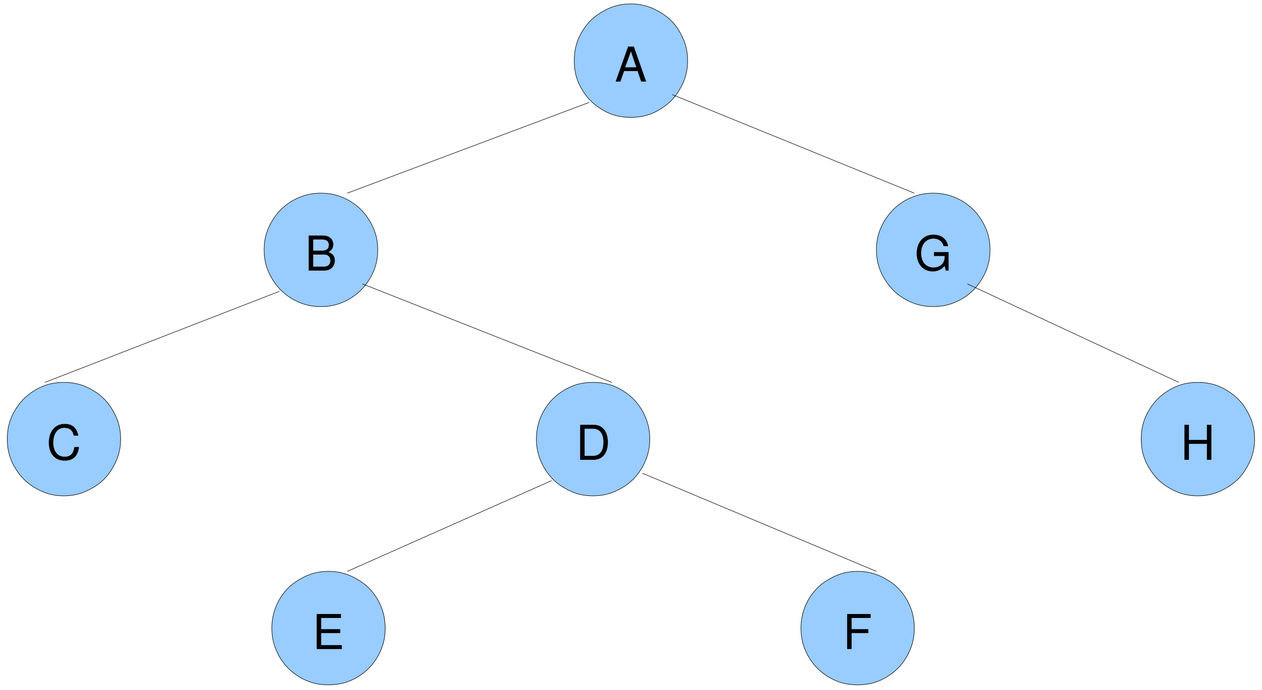


Te Kunenga ki Pūrehuroa

Massey University
COLLEGE OF SCIENCES

Threaded Binary Trees

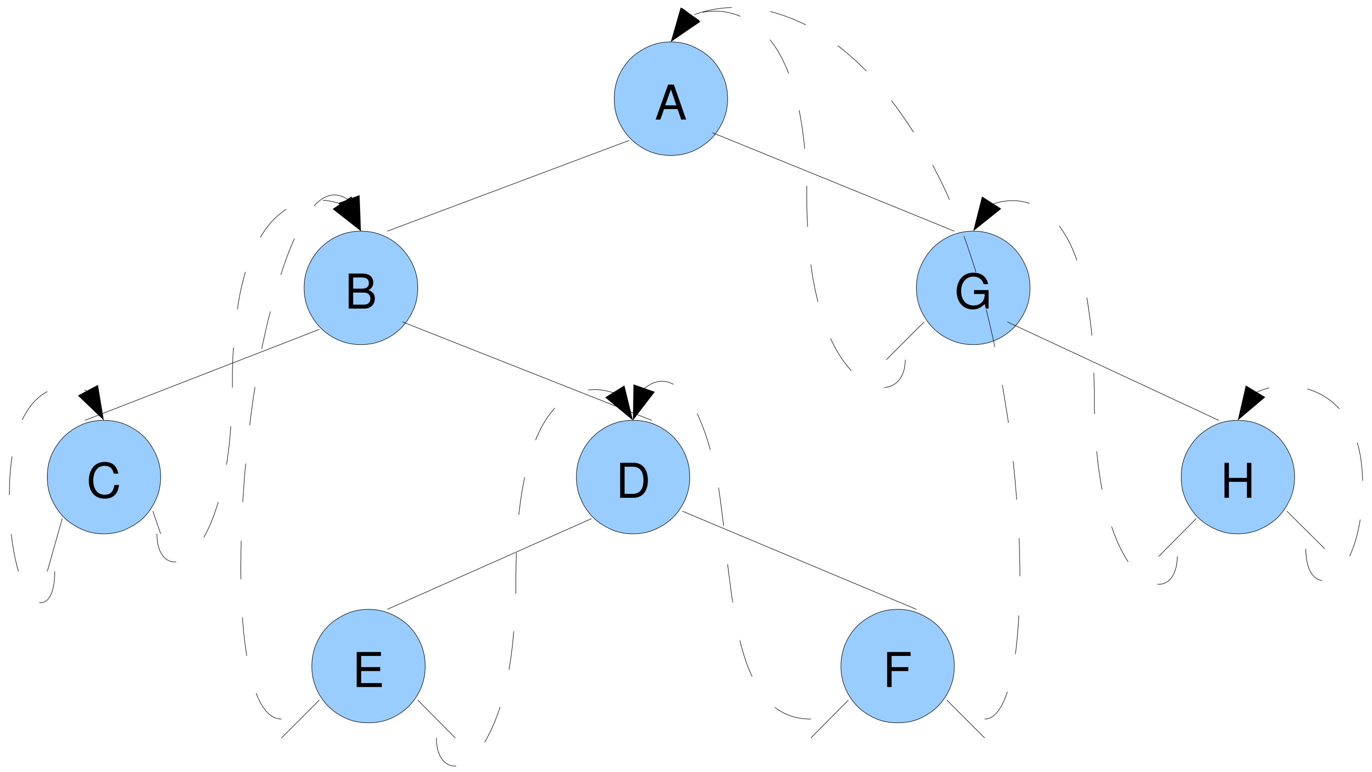
We used stacks either implicitly (*recursive*) or explicitly (*iterative* using stack with tree pointers). Extra *time* and *space* are needed to maintain the stack.



Te Kunenga ki Pūrehuroa

Threaded Binary Trees

Solution: use extra pointers to point to the next node in some order, and to the the previous pointer in some order.e.g., in-order C B E D F A G H

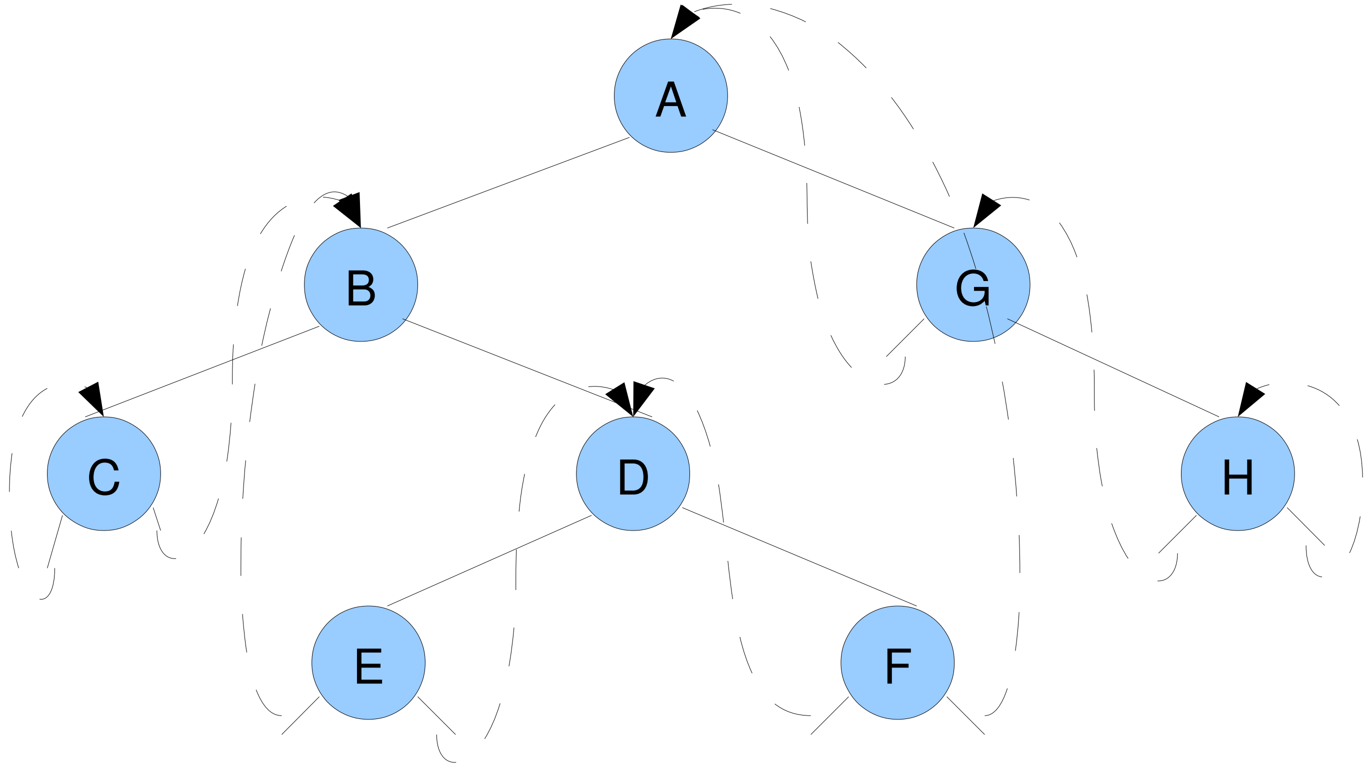


Threaded Binary Trees

These pointers can use the left-over space of NULL pointers.

Left threads point to previous node

Right threads point to next node in the traversal

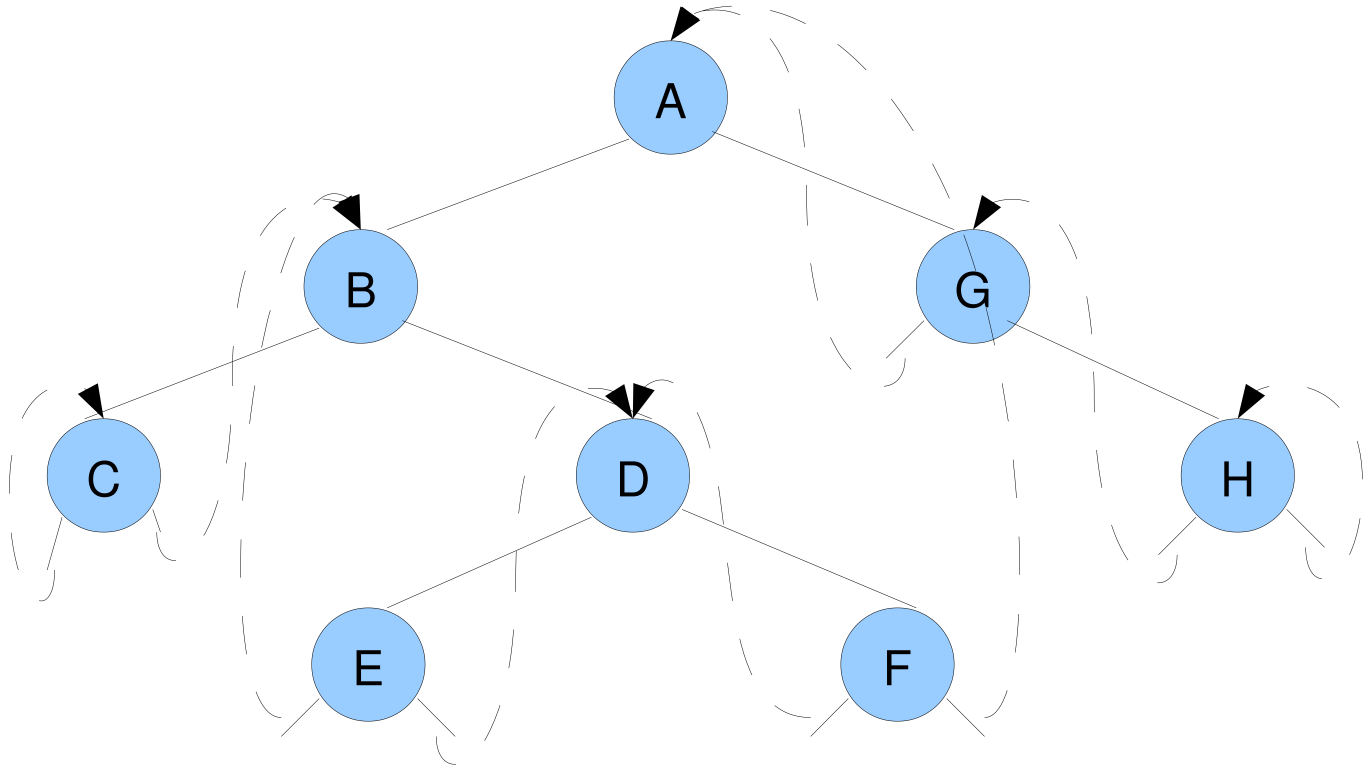


Threaded Binary Trees

E.g., node C threads points to itself and to B

Node A has no threads (it is a **branch**)

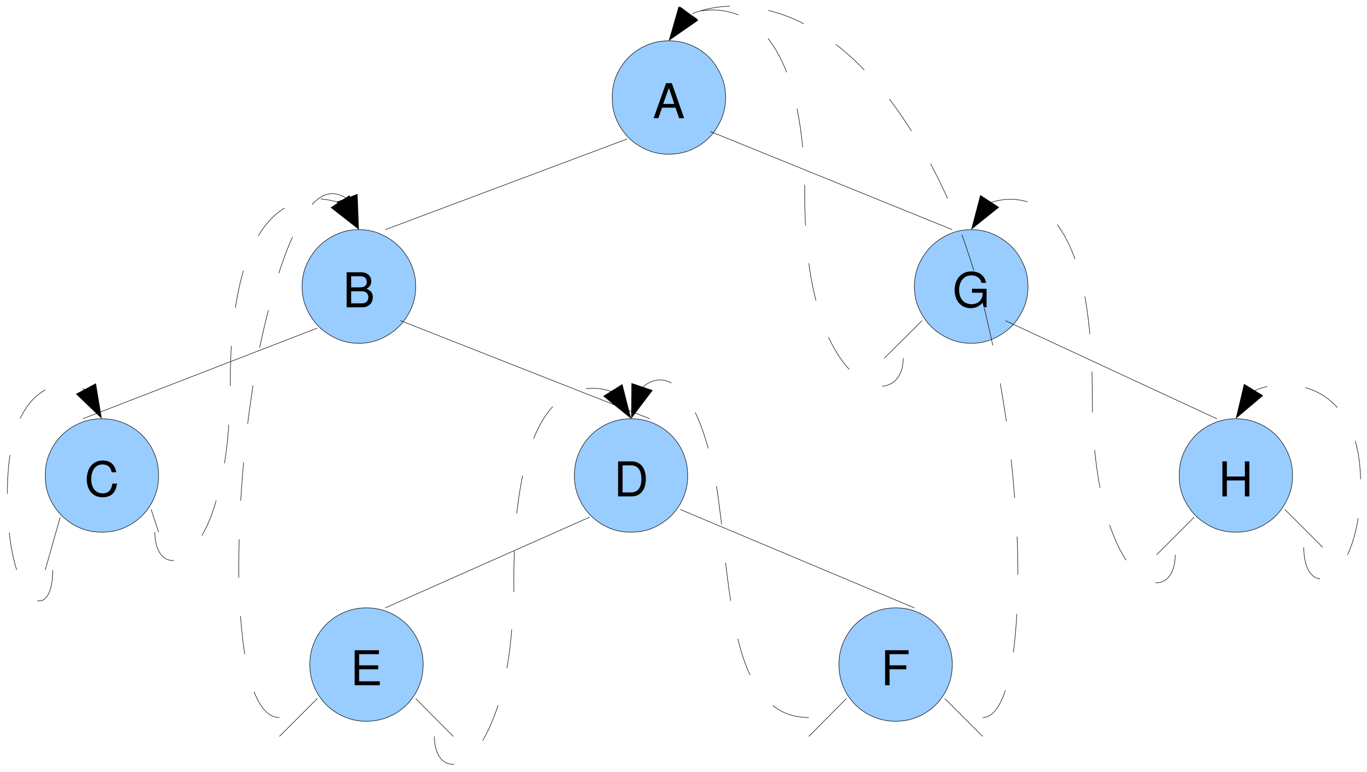
Node G thread (only one) points to A



Threaded Binary Trees

Problem: How to distinguish threads from normal pointers (branches)??

We can use something to mark them, i.e., a Boolean variable as part of the structure.



Te Kunenga ki Pūrehuroa

Massey University
COLLEGE OF SCIENCES

Threaded Binary Trees Class

```
class Tree {
private:
  char data;
  Tree *leftptr, *rightptr;
  bool lthread, rthread;  // two new variables
public:
  //*** all methods as before ***
  bool LeftThread() { return lthread; }  // two
new methods
  bool RightThread() { return rthread; }
};
```

Te Kunenga
ki Pūrehuroa

Massey University
COLLEGE OF SCIENCES

Threaded Binary Trees

If the left pointer is a thread then lthread should be set to true.

If the right pointer is a thread, then rthread should be set to true.

This is done when the node is added, so the methods have to be modified slightly.

An example of an iterative function without the use of stacks follows (in-order traversal).

Te Kunenga ki Pūrehuroa

Threaded Binary Trees in-order

```
Tree *NextNode(Tree *N) {
Tree *temp;
  if (N->RightThread()) { return N->Right(); }
  temp = N->Right();
  while (temp->LeftThread() == false) {
    temp = temp->Left();
  }
  return temp;
}
//This returns the next node in the in-order
traversal
```

In-order function:

```
void InOrder(Tree *T) {
Tree *current;
  if (T->isEmpty()) { return; }
// first get current to the starting node
  current = T;
  while (current->LeftThread() == false) {
    current = current->Left();
  }
// now perform the in-order traversal
  while (current->Right() != current) {
   // *** visit the current node ***
    current = NextNode(current);
  }
}
```

Massey University
COLLEGE OF SCIENCES

Threads…

NOTES:

1. Threads assist in writing an iterative traversal function.

2. Threads are set up for one (only) traversal algorithm.

3. Threads take up memory space and need extra time to construct.

4. Threads need to be modified if the tree changes (through the methods that modify the Boolean variables)

Te Kunenga ki Pūrehuroa

Massey University
COLLEGE OF SCIENCES

L 20

Te Kunenga ki Pūrehuroa

Massey University
COLLEGE OF SCIENCES

B-Trees

A B-Tree is a generalisation of a binary search tree.

In a B-Tree more than one element (record or key) is stored in a node (in a *multiway* fashion).

This is done to reduce storage space and access time.

B-Trees are commonly used in file systems and databases.

Te Kunenga ki Pūrehuroa

B-Trees

Definition: A B-Tree of order *m* is a general tree with the following rules:

a) the root has at least two subtrees and a key, unless it is a leaf.

b) Each leaf node holds k-1 keys (and k pointers)

with $m/2 \leq k \leq m$

c) Each leaf node holds k-1 keys

d) all leaves are at the same level

By this definition, a B-tree is always at least half full, with few levels and balanced. Let's see some examples.

Massey University
COLLEGE OF SCIENCES

B-Trees

One way to implement an *m*-order B-Tree is to have:

- One array for keys (*m*-1)
- One array for pointers (m)
- Number of keys in a node
- Leaf/nonleaf flag

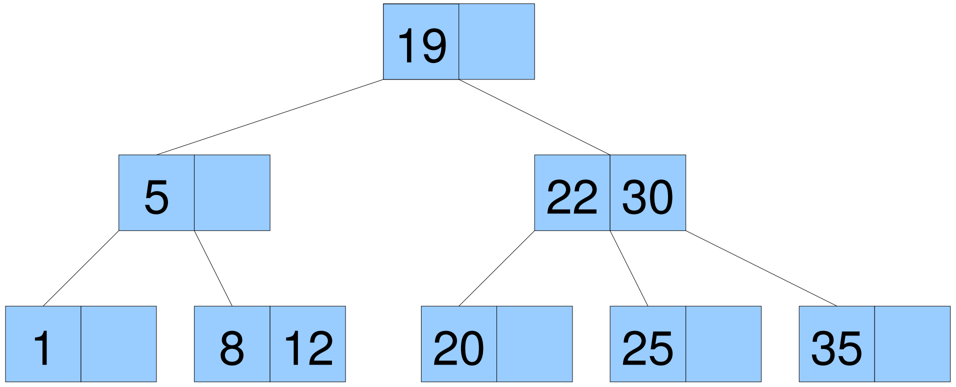
```
class BTreeNode {
private:
  int keys[M-1];
  int number_of_keys;
  BtreeNode *pointers[M];
  bool isleaf;
public:
...};
```

Te Kunenga
ki Pūrehuroa

Massey University
COLLEGE OF SCIENCES

B-Trees

Example: a B-tree with order 3



Note: - No more than 2 keys per node

- No more than 3 pointers per node

- Balanced

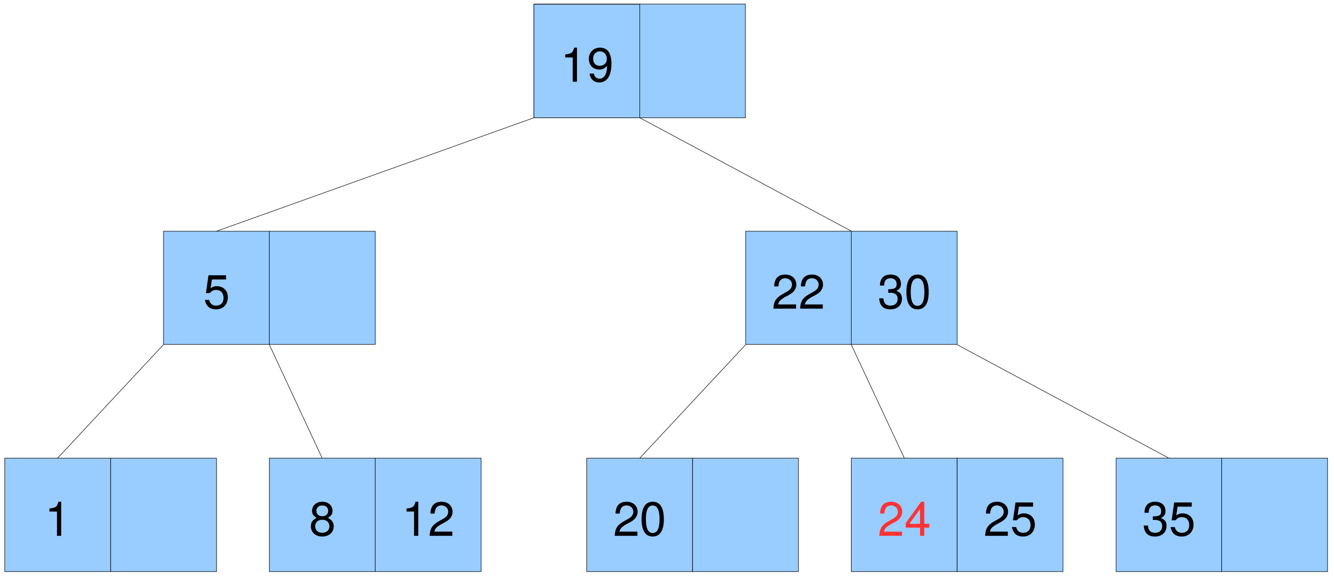
Challenge: re-create this tree:
19,5,22,30,1,8,20,25,35,12

Te Kunenga
ki Pūrehuroa

Order 3 B-Tree: insert example

Not necessarily creates a new node. Three cases:

Case 1: add to a leaf, fits the key according to value.

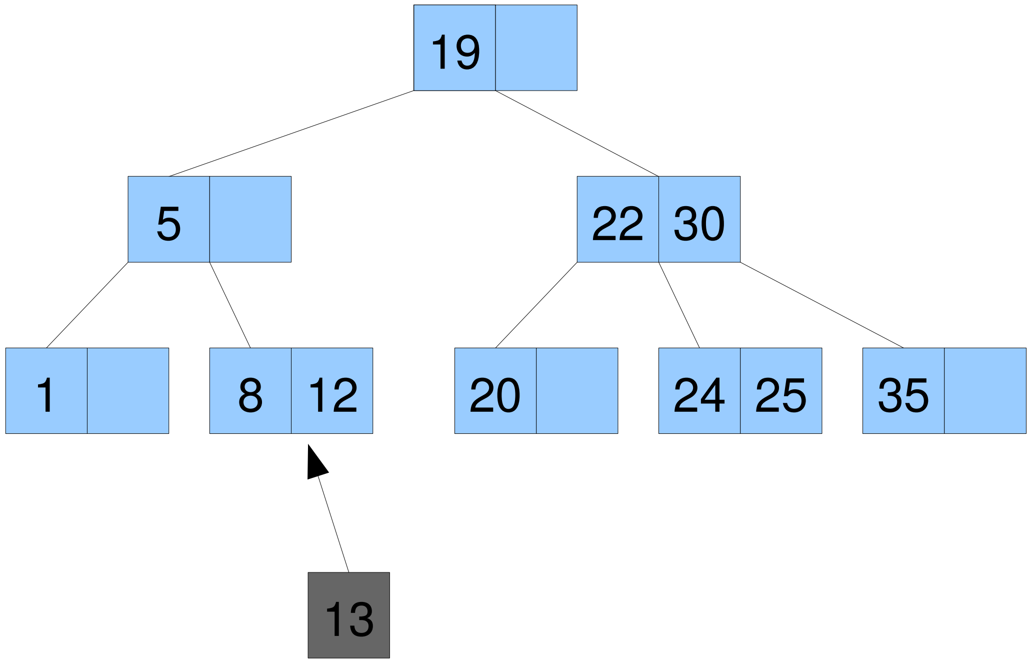


Just displaced 25 to fit 24, no new nodes.

Order 3 B-Tree: insert

Case 2: the node where the key should be is **full**.

Add 13: problem, node 8-12 is full (M-1 keys)



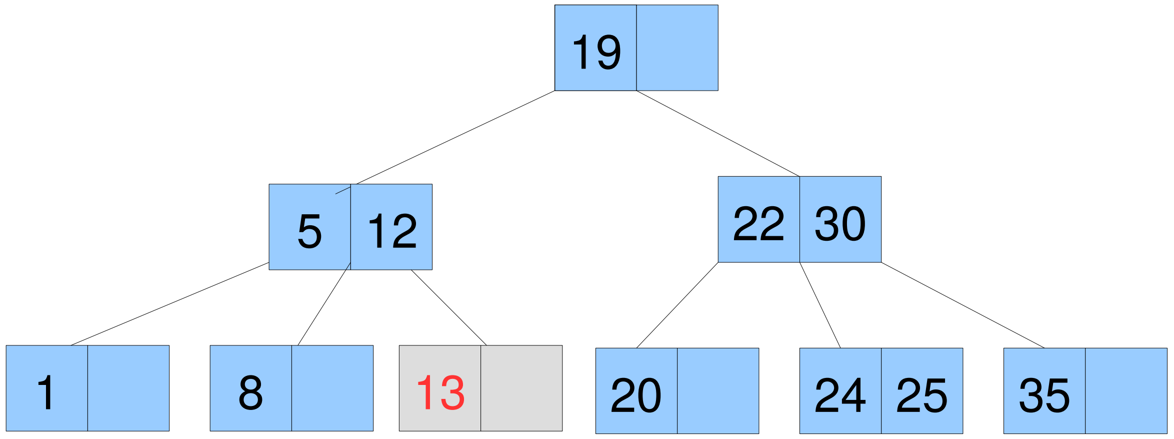
The full node needs to be split....

Analyse the keys 8-12-13, split in the middle (12)

Order 3 B-Tree: insert

Case 2: (cont.)

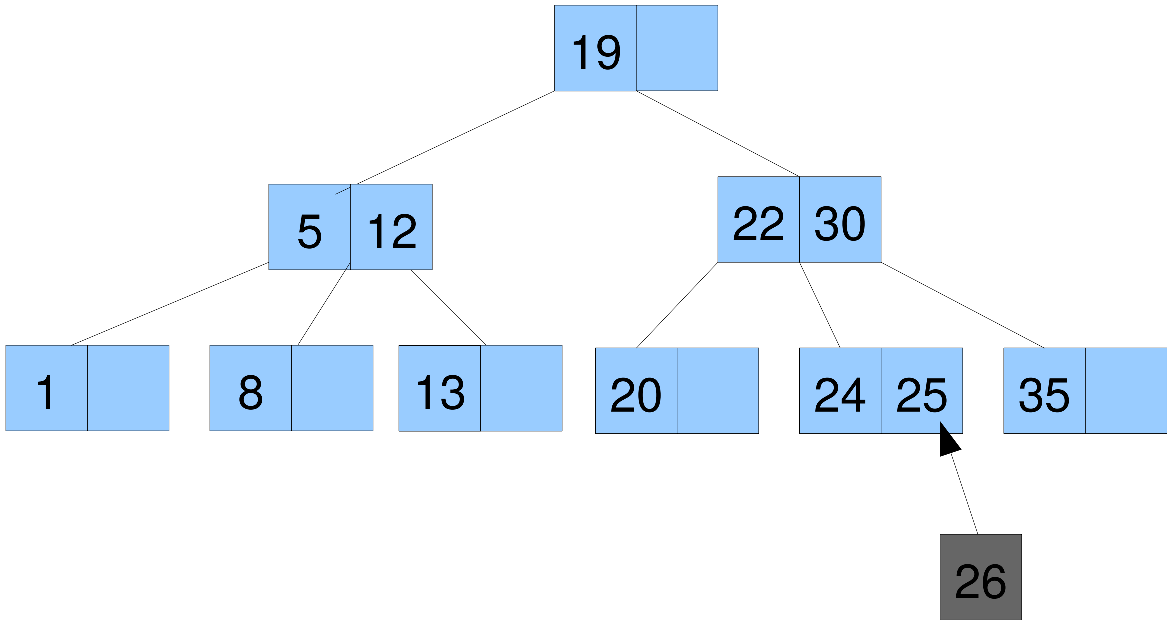
Adding 13...



Create a new node to store 13, leave 8 on the old node, copy 12 to the root, reshuffle the pointers.

Order 3 B-Tree: insert

Case 3: the root of the sub-tree is also **full**.



The full node needs to be split....

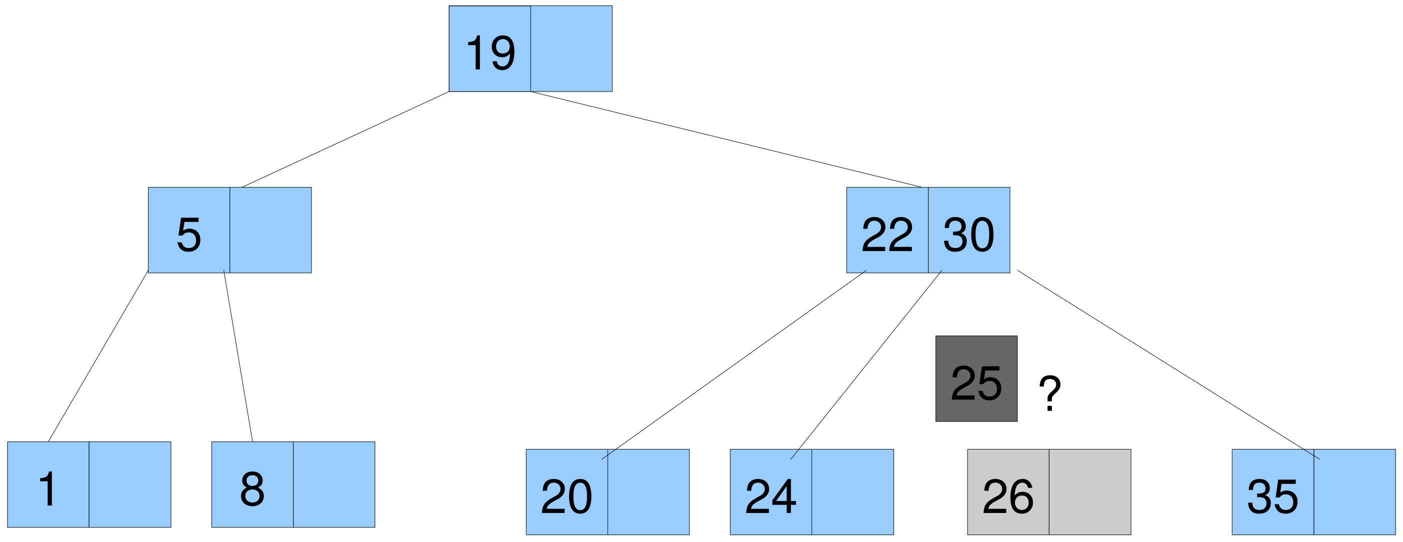
Analyse the keys 24-25-26, split in the middle (25)

Problem: not enough pointers on the root of sub-tree

Massey University
COLLEGE OF SCIENCES

Order 3 B-Tree: insert

Case 3: the root of the sub-tree is also **full**.



The full node needs to be split....

Analyse the keys 24-25-26, split in the middle (25)

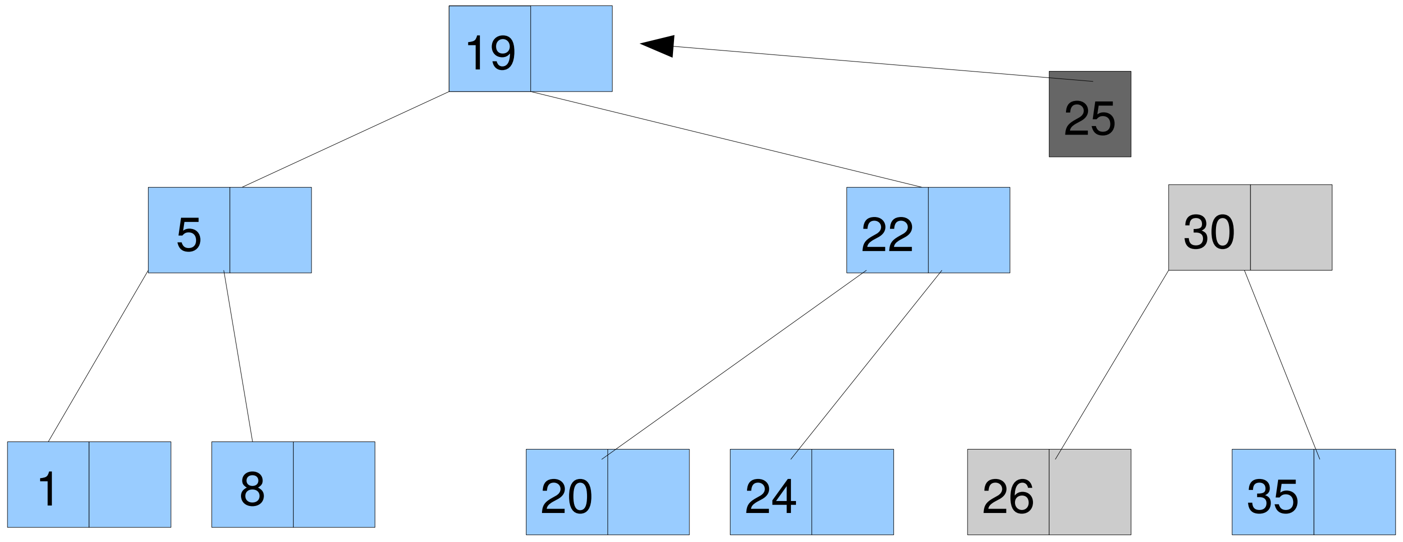
Problem: not enough pointers on the root of sub-tree

Te Kunenga ki Pūrehuroa

Massey University
COLLEGE OF SCIENCES

Order 3 B-Tree: insert

Case 3: the root of the sub-tree is also **full**.



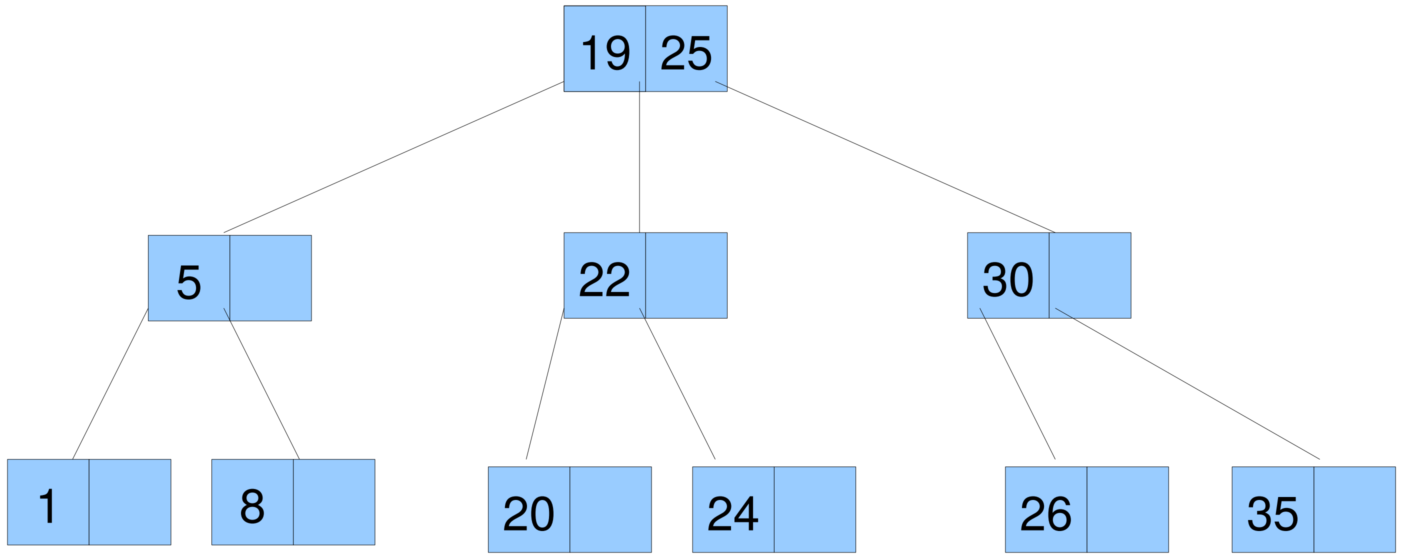
The root node (sub-tree) also needs to be split....

Analyse the keys 22-25-30, split in the middle (25)

Te Kunenga ki Pūrehuroa

Order 3 B-Tree: insert

Case 3: the root of the sub-tree is also **full**.



There is more space available on the leaves. Search is minimised as the tree is always balanced. However, there is under-utilised memory within the nodes.

Massey University
COLLEGE OF SCIENCES

Order 3 B-Tree: insert algorithm

```
BtreeInsert(key)
 node = leaf to insert key (search);
 while(true)
       if (node not full)
             insert key, increment number;
             return;
       else split node into node1 and node2
             distribute keys and pointers;
             key=middle;
             if(node not root)
                   node=its parent;
             else
                   create new root (parent of nodes 1,2)
                   copy key to it
                   distribute pointers
                   return;
```

Te Kunenga
ki Pūrehuroa

Challenge

1) What should be changed in the class and on the insert algorithm to cater for an order 4 B-Tree? And order 5? Order N?

2) Write the code to implement the insert algorithm using the B-Tree class taught on the slides. Insert elements and test your code (stress test it).

3) Analyse the B-Tree and write an algorithm for deletion of order 3 Trees. Can you generalise it for different orders?