Foundations of Computer Science

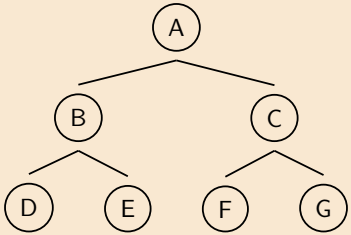
Tree traversals

Dr. Robert Harle & Dr. Jeremy Yallop

2020–2021

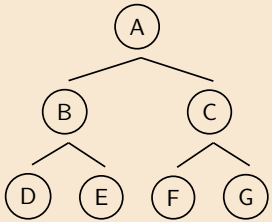
Tree traversal refers to visiting the nodes of each tree in a well-defined order.

preorder	visits the label first	(ABDECFG)
inorder	visits the label midway	(DBEAFCG)
postorder	visits the label last	(DEBFGCA)



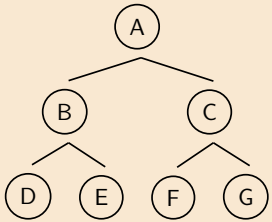
preorder visits the label first (ABDECFG)

```
In[1]: let rec preorder = function
         | Lf -> []
         | Br (v, t1, t2) ->
             [v] @ preorder t1 @ preorder t2

Out[1]:val preorder : 'a tree -> 'a list = <fun>
```



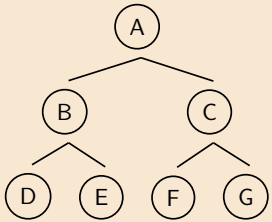
preorder visits the label first (ABDECFG)

```
In[1]: let rec preorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          [v] @ preorder t1 @ preorder t2

Out[1]:val preorder : 'a tree -> 'a list = <fun>
```



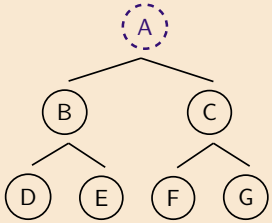
preorder visits the label first (ABDECFG)

```
In[1]: let rec preorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          [v] @ preorder t1 @ preorder t2

Out[1]: val preorder : 'a tree -> 'a list = <fun>
```



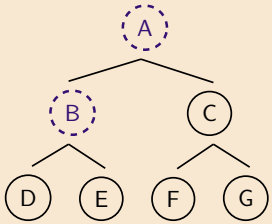
preorder visits the label first (ABDECFG)

```
In[1]: let rec preorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          [v] @ preorder t1 @ preorder t2

Out[1]: val preorder : 'a tree -> 'a list = <fun>
```



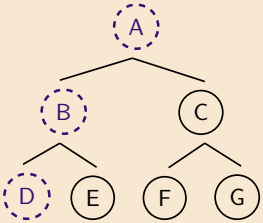
preorder visits the label first (ABDECFG)

```
In[1]: let rec preorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          [v] @ preorder t1 @ preorder t2

Out[1]: val preorder : 'a tree -> 'a list = <fun>
```



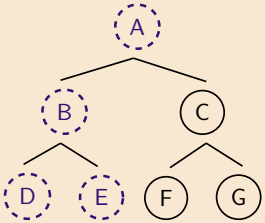
preorder visits the label first (ABDECFG)

```
In[1]: let rec preorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          [v] @ preorder t1 @ preorder t2

Out[1]: val preorder : 'a tree -> 'a list = <fun>
```



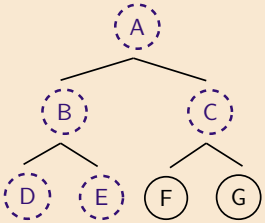
preorder visits the label first (ABDECFG)

```
In[1]: let rec preorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          [v] @ preorder t1 @ preorder t2

Out[1]: val preorder : 'a tree -> 'a list = <fun>
```



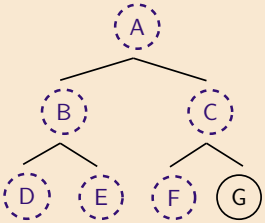
preorder visits the label first (ABDECFG)

```
In[1]: let rec preorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          [v] @ preorder t1 @ preorder t2

Out[1]: val preorder : 'a tree -> 'a list = <fun>
```



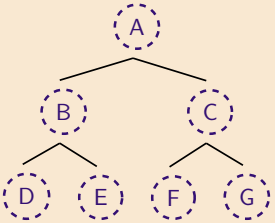
preorder visits the label first (ABDECFG)

```
In[1]: let rec preorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          [v] @ preorder t1 @ preorder t2

Out[1]: val preorder : 'a tree -> 'a list = <fun>
```



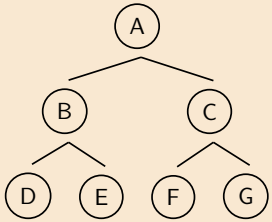
preorder visits the label first (ABDECFG)

```
In[1]: let rec preorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          [v] @ preorder t1 @ preorder t2

Out[1]: val preorder : 'a tree -> 'a list = <fun>
```



inorder visits the label midway (DBEAFCG)

```
In[2]: let rec inorder = function
         | Lf -> []
         | Br (v, t1, t2) ->
             inorder t1 @ [v] @ inorder t2

Out[2]: val inorder : 'a tree -> 'a list = <fun>
```

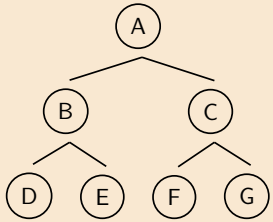


For binary search trees, this order respects the sorting constraint (left key $<$ right key)

Also imaginatively known as a treesort.

inorder visits the label midway (DBEAFCG)

```
In[2]: let rec inorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          inorder t1 @ [v] @ inorder t2

Out[2]:val inorder : 'a tree -> 'a list = <fun>
```

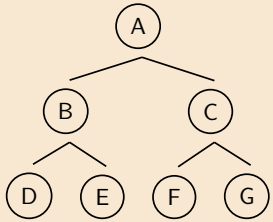


For binary search trees, this order respects the sorting constraint (left key $<$ right key)

Also imaginatively known as a treesort.

inorder visits the label midway (DBEAFCG)

```
In[2]: let rec inorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          inorder t1 @ [v] @ inorder t2

Out[2]: val inorder : 'a tree -> 'a list = <fun>
```

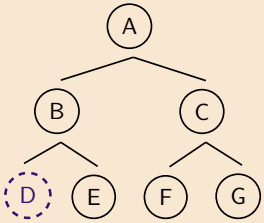


For binary search trees, this order respects the sorting constraint (left key $<$ right key)

Also imaginatively known as a treesort.

inorder visits the label midway (DBEAFCG)

```
In[2]: let rec inorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          inorder t1 @ [v] @ inorder t2

Out[2]: val inorder : 'a tree -> 'a list = <fun>
```

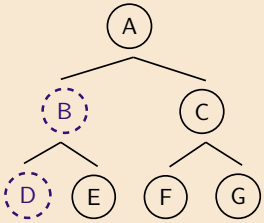


For binary search trees, this order respects the sorting constraint (left key $<$ right key)

Also imaginatively known as a treesort.

inorder visits the label midway (DBEAFCG)

```
In[2]: let rec inorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          inorder t1 @ [v] @ inorder t2

Out[2]: val inorder : 'a tree -> 'a list = <fun>
```

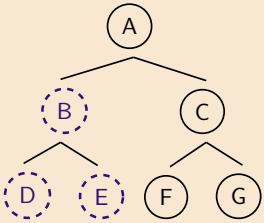


For binary search trees, this order respects the sorting constraint (left key $<$ right key)

Also imaginatively known as a treesort.

inorder visits the label midway (DBEAFCG)

```
In[2]: let rec inorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          inorder t1 @ [v] @ inorder t2

Out[2]: val inorder : 'a tree -> 'a list = <fun>
```

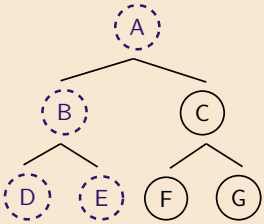


For binary search trees, this order respects the sorting constraint (left key $<$ right key)

Also imaginatively known as a treesort.

inorder visits the label midway (DBEAFCG)

```
In[2]: let rec inorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          inorder t1 @ [v] @ inorder t2

Out[2]: val inorder : 'a tree -> 'a list = <fun>
```

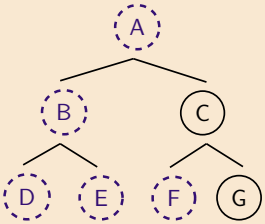


For binary search trees, this order respects the sorting constraint (left key $<$ right key)

Also imaginatively known as a treesort.

inorder visits the label midway (DBEAFCG)

```
In[2]: let rec inorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          inorder t1 @ [v] @ inorder t2

Out[2]: val inorder : 'a tree -> 'a list = <fun>
```

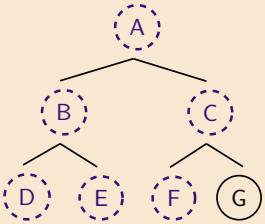


For binary search trees, this order respects the sorting constraint (left key $<$ right key)

Also imaginatively known as a treesort.

inorder visits the label midway (DBEAFCG)

```
In[2]: let rec inorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          inorder t1 @ [v] @ inorder t2

Out[2]: val inorder : 'a tree -> 'a list = <fun>
```

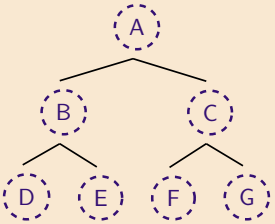


For binary search trees, this order respects the sorting constraint (left key $<$ right key)

Also imaginatively known as a treesort.

inorder visits the label midway (DBEAFCG)

```
In[2]: let rec inorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          inorder t1 @ [v] @ inorder t2

Out[2]: val inorder : 'a tree -> 'a list = <fun>
```

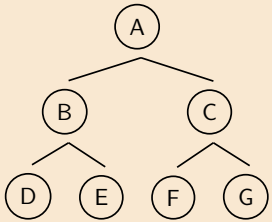


For binary search trees, this order respects the sorting constraint (left key $<$ right key)

Also imaginatively known as a treesort.

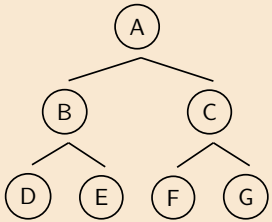
postorder visits the label last (DEBFGCA)

```
In[3]:let rec postorder = function
        | Lf -> []
        | Br (v, t1, t2) ->
            postorder t1 @ postorder t2 @ [v]

Out[3]:val postorder : 'a tree -> 'a list = <fun>
```



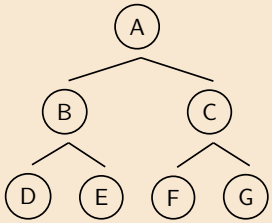
postorder visits the label last (DEBFGCA)

```
In[3]: let rec postorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          postorder t1 @ postorder t2 @ [v]

Out[3]: val postorder : 'a tree -> 'a list = <fun>
```



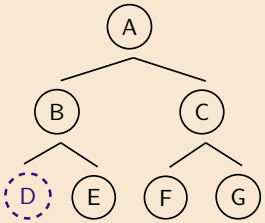
postorder visits the label last (DEBFGCA)

```
In[3]: let rec postorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          postorder t1 @ postorder t2 @ [v]

Out[3]: val postorder : 'a tree -> 'a list = <fun>
```



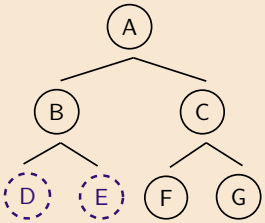
postorder visits the label last (DEBFGCA)

```
In[3]: let rec postorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          postorder t1 @ postorder t2 @ [v]

Out[3]: val postorder : 'a tree -> 'a list = <fun>
```



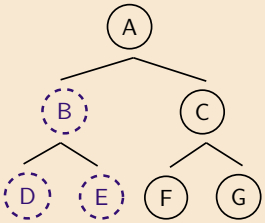
postorder visits the label last (DEBFGCA)

```
In[3]: let rec postorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          postorder t1 @ postorder t2 @ [v]

Out[3]: val postorder : 'a tree -> 'a list = <fun>
```



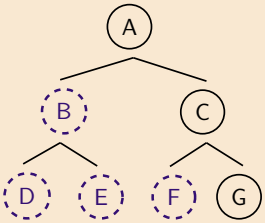
postorder visits the label last (DEBFGCA)

```
In[3]: let rec postorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          postorder t1 @ postorder t2 @ [v]

Out[3]: val postorder : 'a tree -> 'a list = <fun>
```



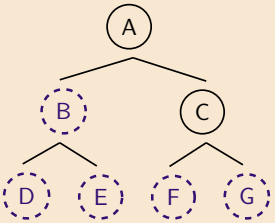
postorder visits the label last (DEBFGCA)

```
In[3]: let rec postorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          postorder t1 @ postorder t2 @ [v]

Out[3]: val postorder : 'a tree -> 'a list = <fun>
```



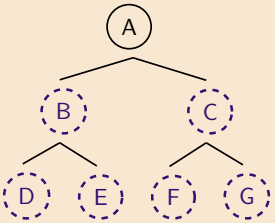
postorder visits the label last (DEBFGCA)

```
In[3]: let rec postorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          postorder t1 @ postorder t2 @ [v]

Out[3]: val postorder : 'a tree -> 'a list = <fun>
```



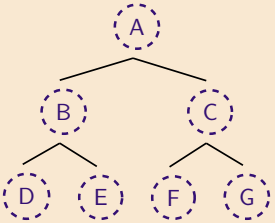
postorder visits the label last (DEBFGCA)

```
In[3]: let rec postorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          postorder t1 @ postorder t2 @ [v]

Out[3]: val postorder : 'a tree -> 'a list = <fun>
```



postorder visits the label last (DEBFGCA)

```
In[3]: let rec postorder = function
       | Lf -> []
       | Br (v, t1, t2) ->
          postorder t1 @ postorder t2 @ [v]

Out[3]: val postorder : 'a tree -> 'a list = <fun>
```



Tree traversal refers to visiting the nodes of each tree in a well-defined order.

preorder, inorder and postorder are **depth-first** traversal algorithms.

The other possibility is **breadth-first** by going across the levels of the tree.