ANSWERS

A4.6 Binary search trees

Trees and graphs · A level · OCR H446 1.4.2, AQA 7517 4.2.5.1, Eduqas A500QS 1.1 · about 30 min

BugBotLab

What this lesson is about

Building, inserting and searching, O(log n) against O(n), trees stored in arrays, and deletion in outline.

- The rule
- Building a tree by inserting
- Searching
- How fast is it?
- A tree in arrays
- Deleting, in outline

Questions

6 marks in all

1. The keys 40, 20, 60, 10, 30 are inserted into an empty binary search tree in that order. Put the keys compared when searching for 30 in order. [1 mark]

Number the lines 1 to 3 to put them in the right order.

___ `20`
___ `30`
___ `40`

Answer:

```
40
20
30
```

30 is less than 40, so go left to 20; more than 20, so go right, where 30 was inserted.

2. What is the time complexity of searching a balanced binary search tree of n keys? [1 mark]

- ☐ **A** O(log n)
- ☐ **B** O(1)
- ☐ **C** O(n)
- ☐ **D** $O(n^2)$

Answer: A. Each comparison discards about half of what is left, so the number of comparisons grows with $\log_2 n$.

3. Keys are inserted into a binary search tree already in ascending order. What happens? [1 mark]

- ☐ **A** Every node gets only a right child, and searching becomes O(n)
- ☐ **B** The tree stays balanced
- ☐ **C** Every node gets only a left child, and searching becomes O(1)
- ☐ **D** The insert fails

Answer: A. Each new key is larger than everything so far, so it goes right every time: the tree is a linked list.

4. A node to be deleted from a binary search tree has two children. What replaces it? [1 mark]

- ☐ **A** Its in-order successor, the smallest key in its right subtree
- ☐ **B** Its left child, always
- ☐ **C** The root of the tree
- ☐ **D** The largest key in its right subtree

> **Answer:** A. The successor is larger than everything on the left and smaller than everything else on the right. The in-order predecessor also works.

5. What does this program print? [1 mark]

```
key = [50, 30, 70, 60, 80]
left = [1, -1, 3, -1, -1]
right = [2, -1, 4, -1, -1]
i = 0
path = []
while i != -1:
    path.append(key[i])
    if 65 < key[i]:
        i = left[i]
    else:
        i = right[i]
print(path)
```

> **Answer:**
>
> ```
> [50, 70, 60]
> ```
>
> Searching for 65: right from 50 to 70, left from 70 to 60, then right from 60 to -1, so 65 is not in the tree.

6. A binary search tree holds 1023 keys and is perfectly balanced, with every level full. What is the most comparisons a search can take? [1 mark]

> **Answer:** 10. 1023 = 2^10 - 1, so there are 10 levels, and a search makes at most one comparison per level.

The task: a search tree in arrays

Store a binary search tree in three lists that grow together: `key`, `left` and `right`. A node is an index into them: `key[i]` is its key, and `left[i]` and `right[i]` hold the indexes of its children, or -1 for no child. The root is index 0. Write `insert(k)` that appends `k` as a new node with -1 in both pointers and links it into the tree, sending a key left when it is smaller than a node's key and right otherwise. Insert every key in `KEYS`, in order. Print `left:` followed by the values in `left` separated by single spaces, then `right:` and the values in `right` the same way. Then write `search(k)`, which walks down from the root. Print `search 45:` followed by every key it compared, separated by single spaces, then `found` or `not found`, and the same for 55.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

KEYS = [50, 30, 70, 20, 40, 60, 80, 35, 45, 65]
key = []
left = []
right = []

def insert(k):
    key.append(k)
    left.append(-1)
    right.append(-1)
```

The hint students can ask for: A new key goes in the next free index with no children. Then start at the root and follow left or right pointers, comparing as you go, until the pointer you want to follow is -1: that pointer is the one to set. Search follows exactly the same path, noting each key it compares, until it finds the key or reaches -1.

A solution

```
from bugbot import *
connect()

KEYS = [50, 30, 70, 20, 40, 60, 80, 35, 45, 65]
key = []
left = []
right = []

def insert(k):
    key.append(k)
    left.append(-1)
    right.append(-1)
    new = len(key) - 1
    if new == 0:
        return
    i = 0
    while True:
        if k < key[i]:
            if left[i] == -1:
                left[i] = new
                return
            i = left[i]
        else:
            if right[i] == -1:
                right[i] = new
```

```
                return
            i = right[i]

def search(k):
    compared = []
    i = 0 if key else -1
    while i != -1:
        compared.append(str(key[i]))
        if k == key[i]:
            return compared, True
        if k < key[i]:
            i = left[i]
        else:
            i = right[i]
    return compared, False

for k in KEYS:
    insert(k)
print("left:", " ".join(str(p) for p in left))
print("right:", " ".join(str(p) for p in right))
for target in [45, 55]:
    compared, found = search(target)
    print(f"search {target}:", " ".join(compared), "found" if found else "not found")
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.