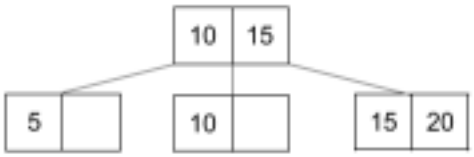
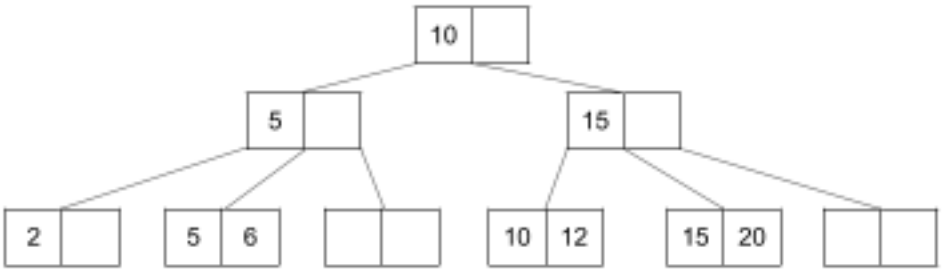
CS W186 - Spring 2024
Exam Prep Section 3
B+ Trees

1 B+ Trees

Given the following (degree $d = 1$) B+ tree:



(a) Draw what the tree looks like after adding 2, 6, and 12 in that order.



For the following questions, consider the tree directly after 2, 6, and 12 have been added.

(b) What is the maximum number of inserts we can do without changing the height of the tree? Answer: **11 inserts**.

The maximum number of keys for a fixed height h is given by $2d \cdot (2d + 1)^h$. We have $d = 1$ from the question, and $h = 2$ (we must remember h is the number of non-root layers). Plugging these numbers in gives us $2 * 3^2 = 18$.

Now, we already have 7 keys in the tree, so we can insert $18 - 7 = 11$ more.

(c) What is the minimum number of inserts we can do that will change the height of the tree?

Answer: **4 inserts** (e.g. 16, 17, 18, 19).

The idea is to repeatedly insert into the fullest nodes; this is hopefully apparent from understanding how and when B-trees split.

2 More B+ Trees

Consider the following schema:

```
CREATE TABLE FineWines (
    name CHAR(20) NOT NULL,
    years_aged INTEGER NOT NULL,
    price INTEGER NOT NULL
);
```

Suppose that we have built an index over <years aged, price>, and suppose this index is two levels deep (in addition to the root node), and data is stored by *list of references* in separate data pages, each of which can hold hundreds of records.

(a)
```
SELECT * FROM FineWines
 WHERE years_aged = 5
   AND price = 100
```

What is the worst case number of page I/Os to execute this query on an unclustered index if there is 1 matching record? 2 matching records? 3 matching records?

Answer: **4, 5, 6 page I/Os**.
As the B-tree is two levels deep, we must do 3 page reads to traverse the index tree: root node, layer 1 node, layer 2 (leaf) node. As the query is an *exact* index key match, and we are storing by *list of references*, all pointers to records for this key will be on the same leaf node.
However, since the index is unclustered, each record might be stored on its own page. Thus, we would need 1 additional page read for each matching record.

(b) What is the worst case number of page accesses to execute this query on a clustered index if there is 1 matching record? 2 matching records? 3 matching records?

Answer: **4, 5, 5 page I/Os**.
As before, the tree index traversal is 3 page I/Os. Since the index is now clustered, records are stored next to each other on pages.
However, there is no guarantee that they won't cross a page boundary. So, two matching records might be on two neighbouring pages: at the end of one, and at the beginning of another. Thus we need up to 2 additional page reads for $\geq$ 2 matching records.

(c)
```
SELECT * FROM FineWines
 WHERE price = 100
```

What is the worst case number of page I/Os to execute this query if there are 150 data pages?

Answer: **150 page I/Os**.
Since this query *does not match* the available index, we just have to read every page.

(d)
```
DELETE FROM FineWines
      WHERE years_aged = 1
        AND price = 5
```

Suppose this query deletes exactly one record. How many page I/Os are needed to execute this query? (Assume no tree rebalancing is required.)

Answer: **6 page I/Os**.
(3) Read index pages (traversing index tree). (1) Read data page. (1) Write data page. (1) Write leaf page in index (to delete the key from the index).