UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

# COS 212 Tutorial 5: Version B

- 23/03/2012
- 50minutes
- 2 questions for a total of 27 marks.

---

Name:______________________________

Student/staff Nr:______________________________

Marker (office use):______________________________

**Question 1** Heaps ........................................................................ (13 marks)

1.1 Consider an array representation for both heaps and binary search trees. As for heaps, in binary search trees the same calculations can be used to find children or parents. With this representation, what advantage does the heap structure property yield that cannot be guaranteed for binary search trees. (2)

Answer:

**Solution:** Array is compact, no open holes, space saved, etc.

1.2 Considering a max-heap, why is searching for any element other that the largest an $O(n)$ operation? (2)

Answer:

**Solution:** No assumption can be made of the relation between elements in sibling subtrees.

OR

Duplicate elements may exist.

1.3 Why is a binary tree an inappropriate structure to use when implementing heaps? (2)

Answer:

**Solution:** Housekeeping..., having to find last position to insert is a problem, having to swap with parents means that you have to keep track of predecessor/ancestors.

1.4 Transform the following array into a min-heap using `Floyd's` algorithm (i.e. bottom up). Redraw the array after `every` swap operation. (6)

`[100,52,51,20,31,23,20,1,13,16,30,1,22,1]`

**Solution:** there should be 11 swaps, 1/2 mark each, 1/2 mark if final array is a min-heap (arranged as one):

```
[100,52,51,20,31,23,1,1,13,16,30,1,22,20]

[100,52,51,20,31,1,1,1,13,16,30,23,22,20]

[100,52,51,20,16,1,1,1,13,31,30,23,22,20]

[100,52,51,1,16,1,1,20,13,31,30,23,22,20]

//From here students have a choice
option 1:
[100,52,1,1,16,51,1,20,13,31,30,23,22,20]

[100,52,1,1,16,23,1,20,13,31,30,51,22,20]

OR
option 2:
[100,52,1,1,16,1,51,20,13,31,30,23,22,20]

[100,52,1,1,16,1,20,20,13,31,30,23,22,51]



Now the swaps will be the same again regardless of 1 or 2 above
left side of the heap is now considered and 52 will swap
Using option 1
```

```
Options again:
Complete option 1:
[1,100,1,13,16,23,1,20,52,31,30,51,22,20]

[1,13,1,100,16,23,1,20,52,31,30,51,22,20]

[1,13,1,20,16,23,1,100,52,31,30,51,22,20]


==Done==

OR options 2:
[1,1,100,13,16,23,1,20,52,31,30,51,22,20]

[1,1,1,13,16,23,100,20,52,31,30,51,22,20]

[1,1,1,13,16,23,20,20,52,31,30,51,22,100]
==Done==


DOUBLE CHECK:  The order is:

20<->1
23<->1
31<->16
20<->1

Here is the split
Either 1:          OR 2:
51<->1 (left)      51<->1 (right)
51<->22            51<->20

Same again for both
52<->1
52<->13

Split:
Either:            OR
100<->1(left)      100<->1(right)
100<->13           (1 from above)  OR (2 from above)
100<->52           100<->22        100<->20
                   100<->51        100<->51
```

Answer:

1.5 Heaps need not be binary. In fact, heaps may be created where each node is allowed any number of children decided on beforehand. The general case for heaps are called `d-Heaps` where `d` indicates the maximum number of children each node is allowed to have. A binary heap can then in essence be referred to as a 2-Heap. What is the formula for calculating the index of the parent for a node at index `i` in a 6-heap? (1)

**Solution:** $\lfloor \frac{i-1}{6} \rfloor$

Answer:

## Question 2 B-Trees ............ (14 marks)

2.1 Not taking maximal block usage into account, the value of `M` in B-Trees needs to be chosen carefully. There are values for `M` that will cause errors in B-Tree operations. Which values for `M`, $M > 0$, are inappropriate? Also list and briefly discuss two problems that arise from an incorrectly chosen `M` (3)

**Solution:** 1 Mark for M cannot be even (or cannot be 1, seeing as the questions has $M > 0$)

1 Mark for each of the following problems:

- For splits, one of the siblings will always underflow if keys are divided.
- There will be one more or one less child (depending on split or merge) that cannot be incorporated into the tree.

Answer:

2.2 Consider the B-Tree in `figure 1`. Delete the keys 20 and 30, in this order from the tree. For your answer give only the final tree. (5)

**Solution:** 1 mark for tree's height becoming one less (from 3 to 2)

1 mark one less leaf from original tree (5 leaves left)

Then...

Option 1 1 mark Root 12,50,60,70

1 mark First leaf is 1,2,10,11

1 mark second leaf is 31,32

OR

Option 2 1 mark Root 10,50,60,70

1 mark first leaf is 1,2

1 mark second leaf is 11,12,31,32

Answer:

2.3 Assume the following partially given `BTreeNode` class:

```
class BTreeNode<T extends Comparable<? super T>>
{
        BTreeNode(T firstKey)
        {
                keys[0] = firstKey;
                for(int i = 0; i < M; ++i)
                        children[i] = null;
                numberOfKeys = 1;
        }

        int numberOfKeys;
        Comparable<T> keys[] = new Comparable[M-1];
```

```
    BTreeNode<T> children = new BTreeNode<T>[M];
}
```

a) Without making any assumptions about any additional methods and fields, how can one test if a `BTreeNode` is a leaf node? (1)

> **Solution:** Test if first element in children is null.

Answer:

b) Write a recursive method with the signature `void descending(BTreeNode<T> n)` which will output the keys in a B-Tree in descending order. (5)

> **Solution:** //See answer in version A, this is just the from back-to-front version.
>
> check null base case for n
>
> loop through keys from back to front (i.e. start at numkeys and work to 0)
>
> do child before key first.
>
> then key output
>
> take care of the first child.

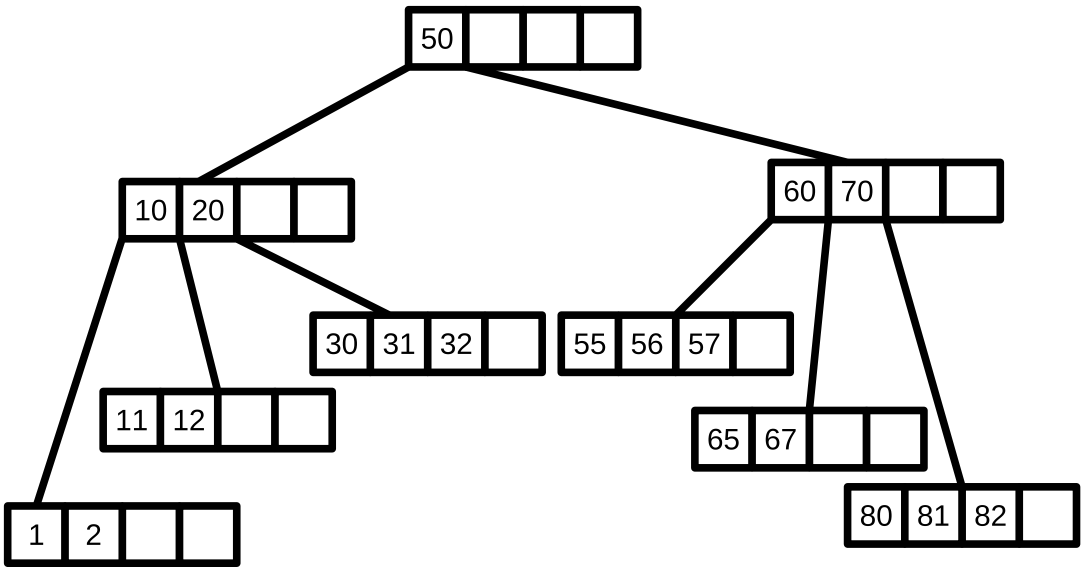


Figure 1: B-Tree