Durham University

COMP1081 Algorithms & Data Structures

Revision lecture – Part 3

Dr. Anish Jindal
anish.jindal@durham.ac.uk

Assessment: Exam Structure

- **2 hours**
- **Four sections (named)**
  - Eg: Selection and data structures (Dr. Anish Jindal)
- **Equal weightage**
- **Independent of each other**

- **Check model exam paper**
- **Check past exams**
  - Key difference for PART 3: You will NOT be asked to design algorithms and prove theorems/properties/etc.

Part 3

- Algorithms for sorting, searching and selection
- Binary Search Trees
- AVL trees
- Heaps
- Lower bounds for sorting and selection

Content

1. Even more sorting (BucketSort, RadixSort)
2. Binary Search
3. Selection (QuickSelect, Median-of-Medians)
4. Binary Search Trees
5. Balanced BSTs (AVL trees. **Red-Black not examined**)
6. Heaps
7. Lower Bounds

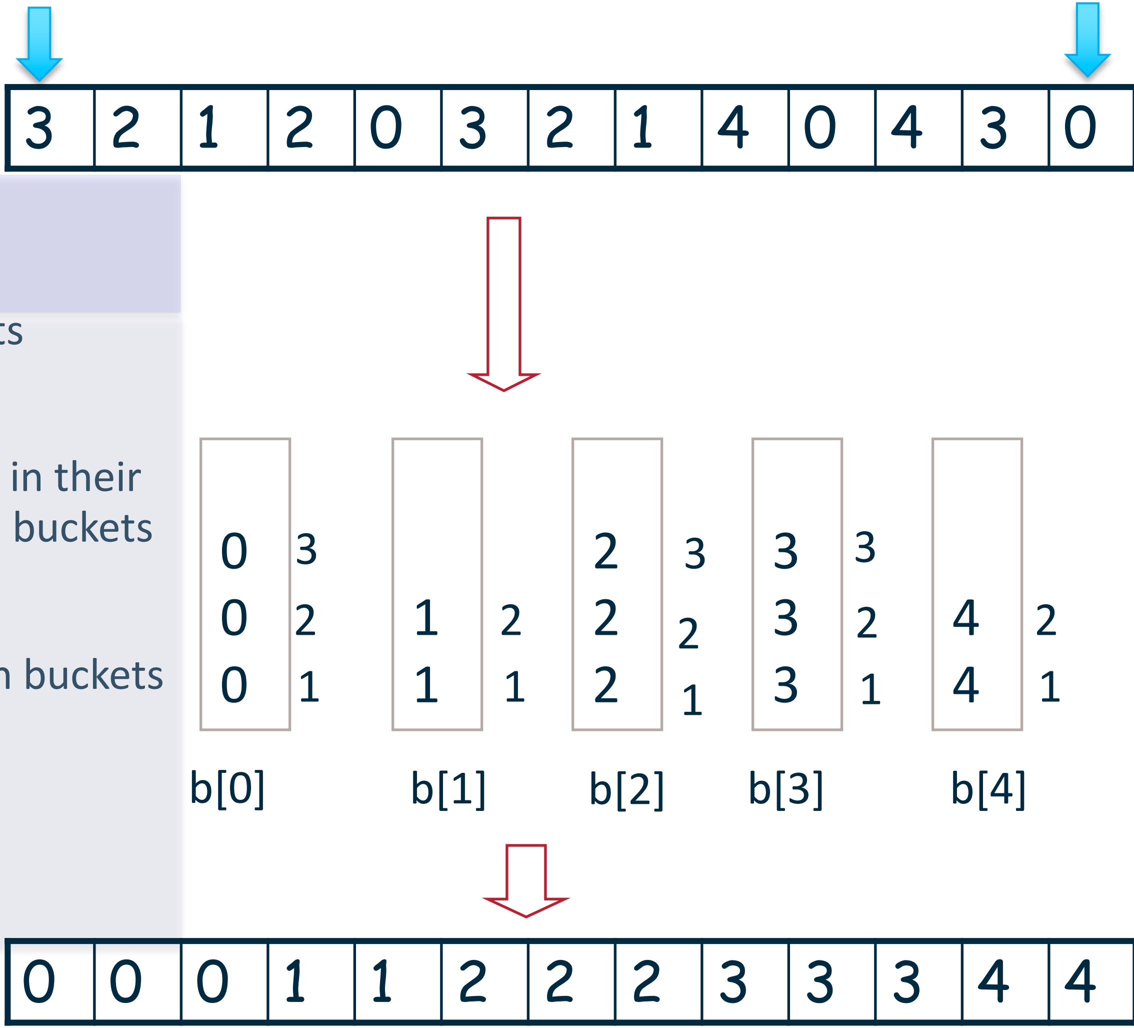
Even more sorting (BucketSort, RadixSort)

- General lower bound $\Omega(n \log n)$ for comparison-based sorting
  - what does it say and mean?
- BucketSort, RadixSort
  - How/why do they work?
  - What are their running times?
  - What are the assumptions?
  - Do they “beat” the lower bound?

BucketSort

Algorithm BucketSort(S)
(values in S are between 0 and k-1)

```
for j = 0 to k-1 do        // initialize k buckets
    b[j] = 0
end for
for i = 0 to n-1 do          // place elements in their
    b[S[i]] = b[S[i]]  + 1      // appropriate buckets
end for
i = 0
for j = 0 to k-1 do        // place elements in buckets
    for r = 1 to b[j] do        // back in S
      S[i] = j
      i = i + 1
    end for
end for
```



Durham University

RadixSort: example

Consider this input:

67, 23, 90, 6, 43, 22, 18, 75, 49, 12, 36

First pass (right-most digit) gives the following buckets:

		12	43			36			
90		22	23		75	6	67	18	49
0	1	2	3	4	5	6	7	8	9

. . . which, in turn, gives the "new" array

90, 22, 12, 23, 43, 75, 6, 36, 67, 18, 49

RadixSort: example..

Second round now works on this new array

90, 22, 12, 23, 43, 75, 6, 36, 67, 18, 49

but now considers the next digit from the right (here: left-most):

	18	23		49					
(0)6	12	22	36	43		67	75		90
0	1	2	3	4	5	6	7	8	9

Dump this table and get

6, 12, 18, 22, 23, 36, 43, 49, 67, 75, 90

Binary Search

- What does it do?
- How does it work?
- How long does it take?
- How is it analysed?

Example: Solution

Find the value 33 from the **sorted array** as below:

Recursive binary search

```
int search (int A[1..n], int left, int right, int x)
{
  if (right == left and A[left] != x)
      handle error; leave function

  p = middle-index between left and right

  if (A[p] == x) then
      return p

  // here come the recursive calls (if x not yet found)
  if (x > A[p]) then
      return search(A,p+1,right,x)  // in right half
  else // x < A[p]
      return search(A,left,p-1,x)  // in left half
}
```

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

left ↑ (index 1) right ↑ (index 15)

Selection

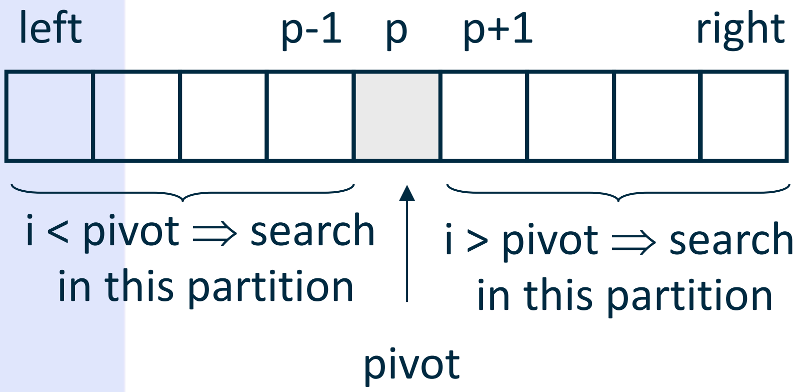
- QuickSelect and Median-of-medians
  - What problems are they for?
  - How do they work?
  - How do they differ?
  - How long does it take?

QuickSelect (int A[1...n], int left, int right, int i)

```
if (left == right) then
    return A[left]
else
    // rearrange/partition in place
    // return value "pivot" is index of pivot element
    // in A[] after partitioning
    pivot = Partition (A, left, right)

    // Now:
    // everything in A[left...pivot-1] is smaller than pivot
    // everything in A[pivot+1...right] is bigger than pivot
    // the pivot is in correct position w.r.t. sortedness

    if (i == pivot) then
        return A[i]
    else if (i < pivot) then
        return QuickSelect (A, left, pivot-1, i)
    else        // i > pivot
        return QuickSelect (A, pivot+1, right, i)
    end if
end if
```

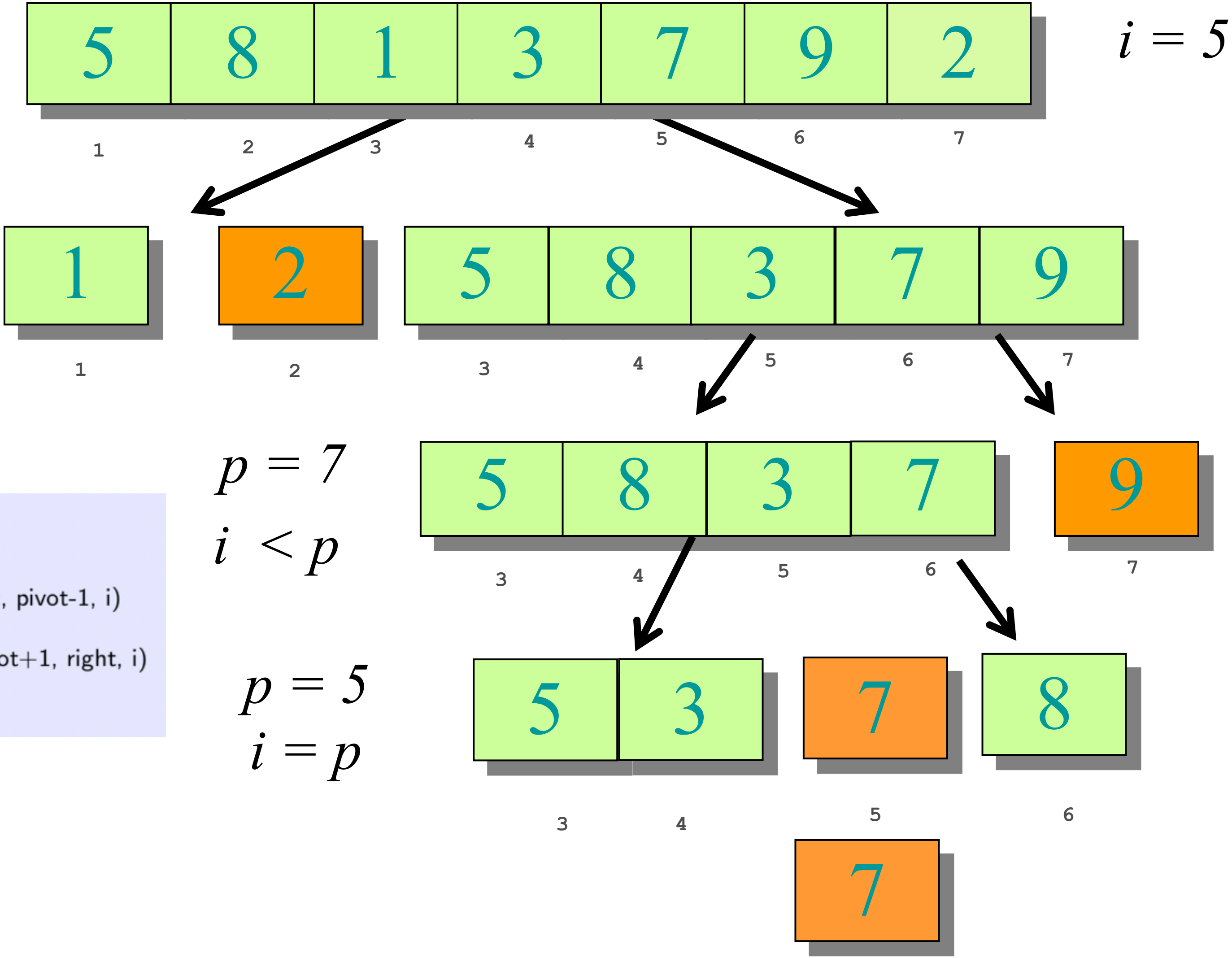


Complete example

$i = 5$

$p = 2$

$i > p$




```
if (i == pivot) then
    return A[i]
else if (i < pivot) then
    return QuickSelect (A, left, pivot-1, i)
else        // i > pivot
    return QuickSelect (A, pivot+1, right, i)
end if
end if
```

Durham University

Median-of-Medians: the algorithm

SELECT(i, n)

Finding pivot

1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.
2. Recursively SELECT the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.
3. Partition around the pivot x. Let $k = \text{rank}(x)$.
4. **if** $i = k$ **then return** x
   elseif $i < k$
   then recursively SELECT the ith smallest element in the lower part
   else recursively SELECT the $(i–k)$th smallest element in the upper part

Same as Quick-Select

Example

A = {12, 34, 0, 3, 22, 4, 17, 32, 3, 28, 43, 82, 25, 27, 34,   2 ,19 ,12 ,5 ,18 ,20 ,33, 16, 33, 21, 30, 3, 47}

$i=11$ – 11th smallest element

1. Divide the array into groups of 5 elements

12	4	43	2	20	30
34	17	82	19	33	3
0	32	25	12	16	47
3	3	27	5	33	
22	28	34	18	21	

Binary Search Trees (BSTs)

- What are they, what's the point?
- BST property (examples)
- The standard operations (how they work)
  - Insertion
  - Search
  - Deletion
  - Traversals
  - …

A **binary search tree** (BST) is a tree in which no node has more than two children (not necessarily exactly two).

The one additional crucial property of BSTs:

BST property

You must build and maintain the tree such that it's true for **every node** v of the tree that:

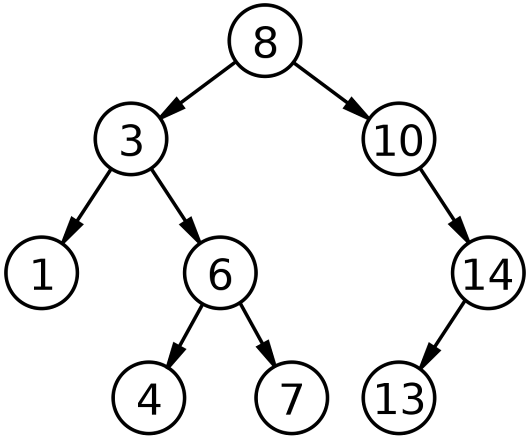
- **all elements in its left sub-tree are "smaller" than** v
- **all elements in its right sub-tree are "bigger" than** v

Smaller and bigger refer to the value. The left/right sub-tree refers to the tree rooted in a node's left/right child.

Just saying "left child smaller and right child bigger" not sufficient!

Your operations that modify the tree must take care not to destroy this property!

Examples



- in-order: 1,3,4,6,7,8,10,13,14
- pre-order: 8,3,1,6,4,7,10,14,13
- post-order: 1,4,7,6,3,13,14,10,8

For BSTs, the in-order traversal gives the elements in sorted order!

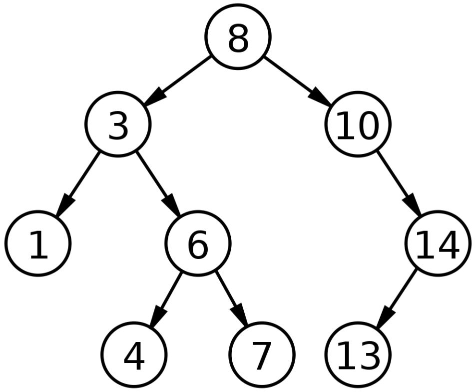
Durham University

Inserting into a BST

To be called on root of tree.

- If match, return (don't insert again).
- If new key is smaller than that of current node, insert on left.
- Otherwise, insert on right.

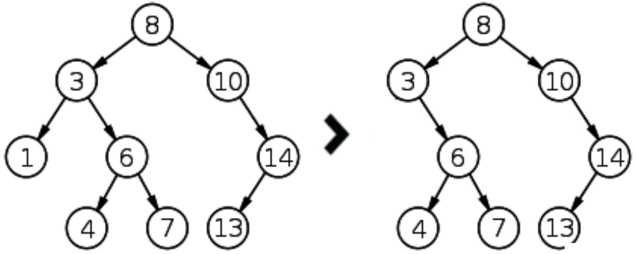
```
root.insert(6)
root.insert(4)
root.insert(7)
root.insert(14)
root.insert(13)
```



Deleting from a BST:

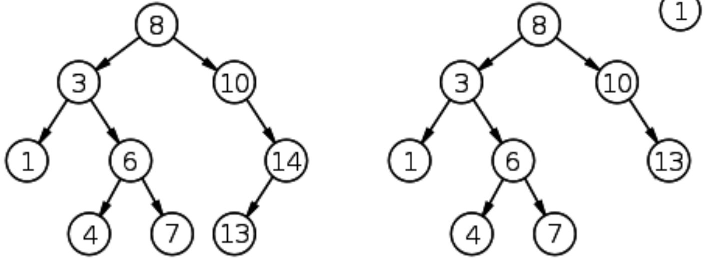
- **no child**

```
root.delete(1)
```



- **one child**

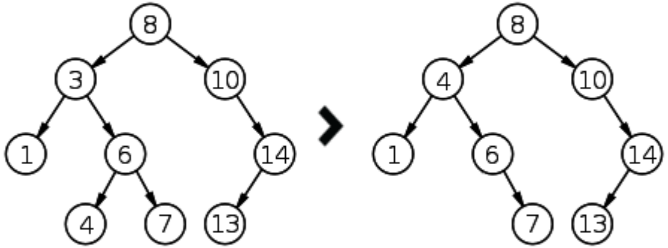
```
root.delete(14)
```



- **two children**

starting from that node, take one step to the right, then always go left

root.delete(3)



Balanced Trees: AVL

- What are they, what's the point?
- Key property (height)
- Re-balancing BSTs: Rotations
- Fix-up procedures after insertion/deletion
- Complexity of operations

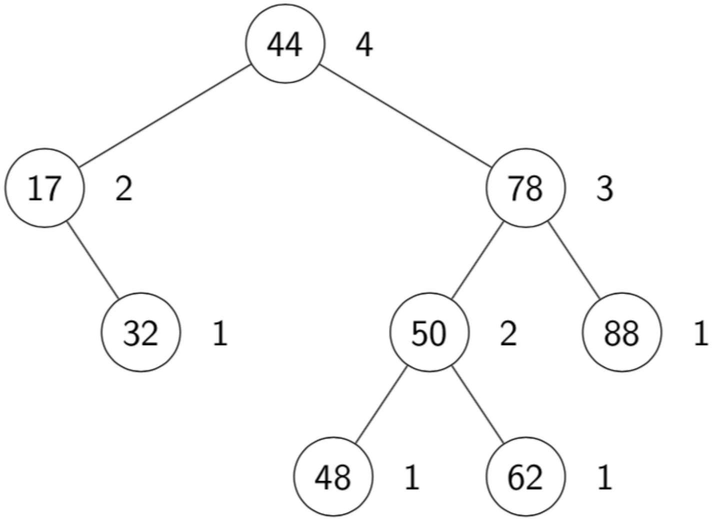
AVL Trees

An **AVL tree** is a self-balancing BST with the following additional property:

> **Height-balance property**
>
> **For each node v, the heights of v's children differ by at most 1.**

Will use the definition of height of a node:

- height of Null is 0 and height of a proper leaf is 1
- height of a parent = max height of a child +1.



AVL rotations

There are 4 cases:

Let the node that needs rebalancing be α.

(require single rotation) :

1. Insertion into left subtree of left child of α.
2. Insertion into right subtree of right child of α.

(require double rotation) :

3. Insertion into right subtree of left child of α.
4. Insertion into left subtree of right child of α.

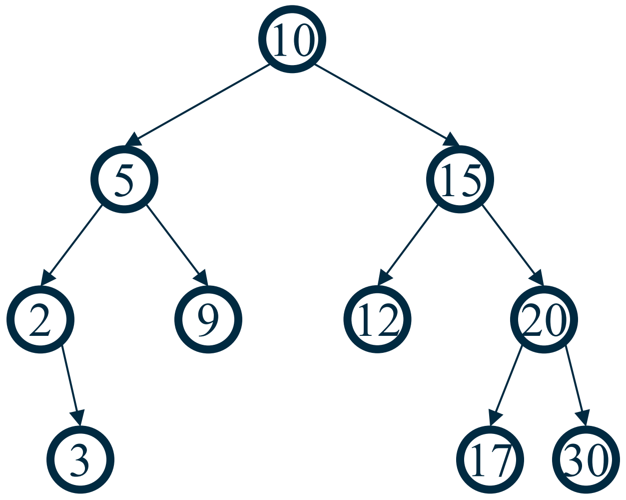
Rotations:

- Right
- Left

- Left-Right
- Right-Left

Insert the following in the given AVL Tree.

Insert(18)



Durham University

Heaps

- What are they, what's the point?
- Min-heaps vs max-heaps
- Heap property
- Representation tree vs array
- Heapify (why? how? how long?)
- BuildHeap (why? how? how long?)
- HeapSort (why? how? how long?)

Heap properties

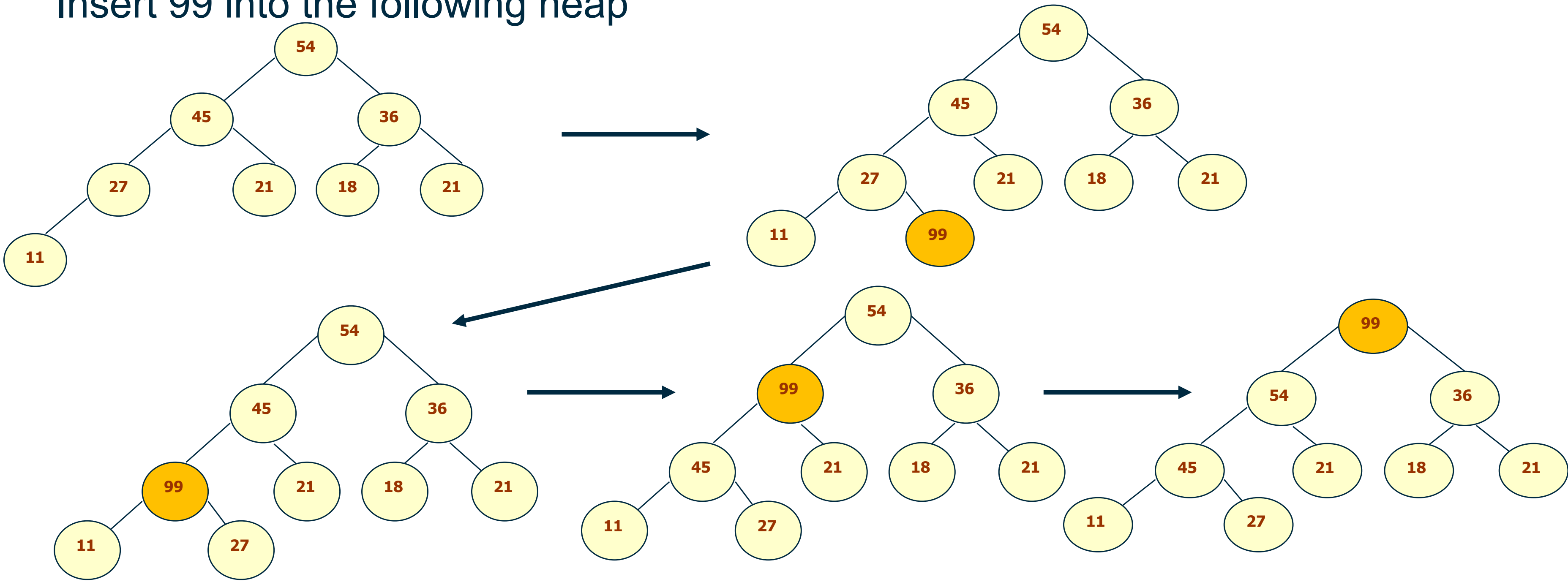
It is a binary tree with the following properties:

- *Property 1:* it is a complete binary tree
- *Property 2:* the value stored at a node is greater or equal to the values stored at the children
(**heap property**)

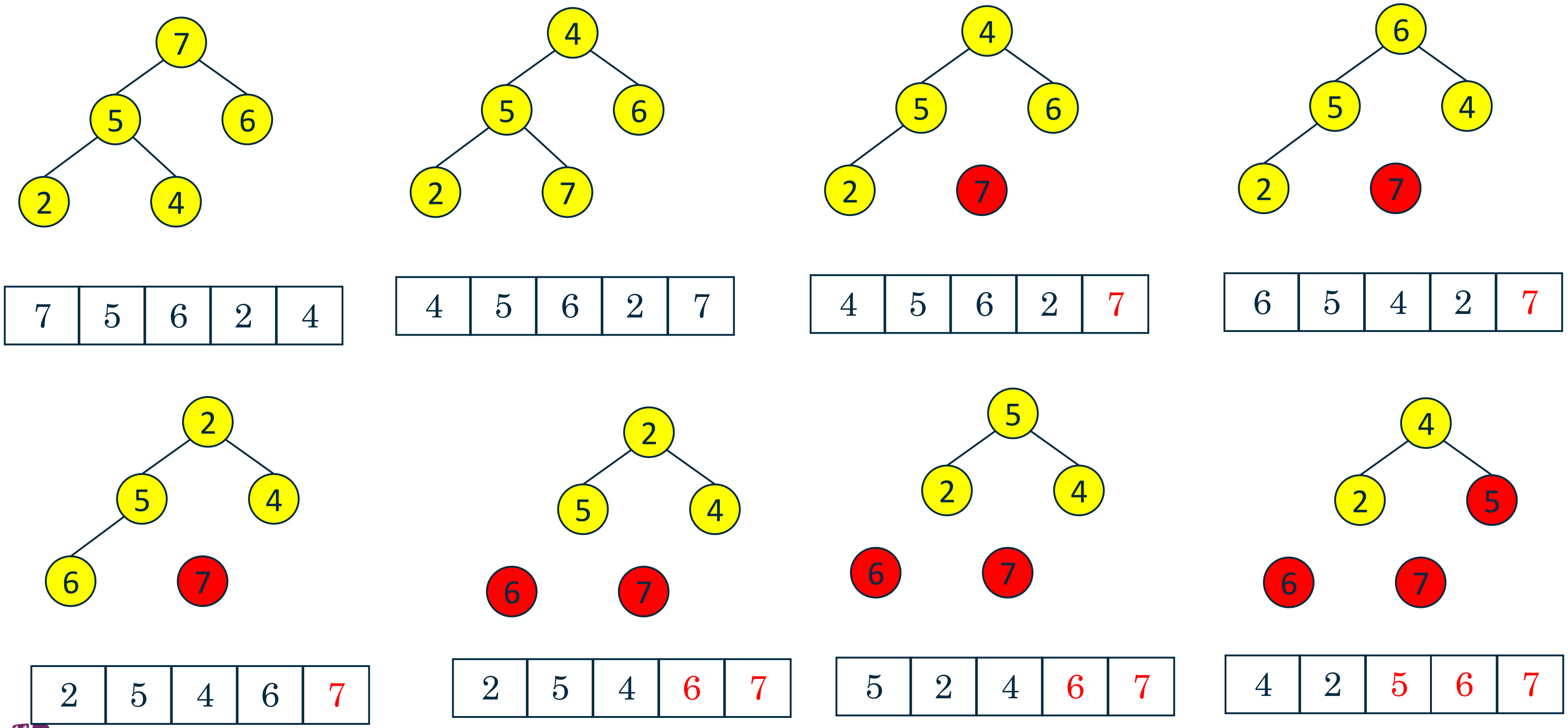
- heap property: for all nodes v in the tree,
v.parent.data >= v.data
- This is for max-heaps
(for min-heaps, v.parent.data =< v.data)

Example (simpler)

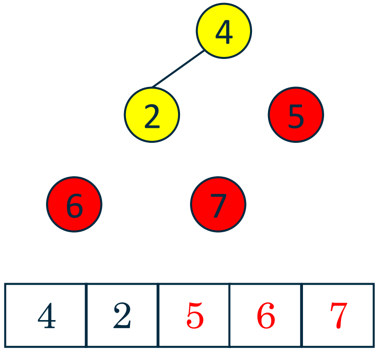
Insert 99 into the following heap

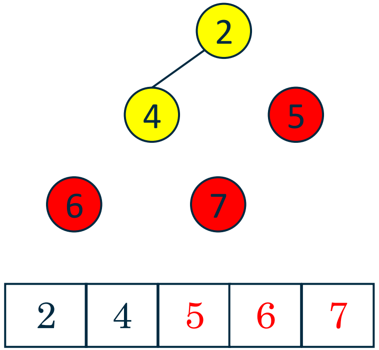


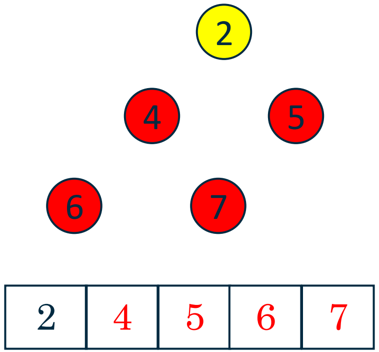
Example (Heapsort)



Example (Heapsort)..







Lower bounds

- What's the point?
- Decision trees
  - What are they? (def'n, examples)
- Adversaries
  - What are they? How can they be designed?
  - Examples of exact bounds (max / 2nd largest / min and max)

Model exam questions

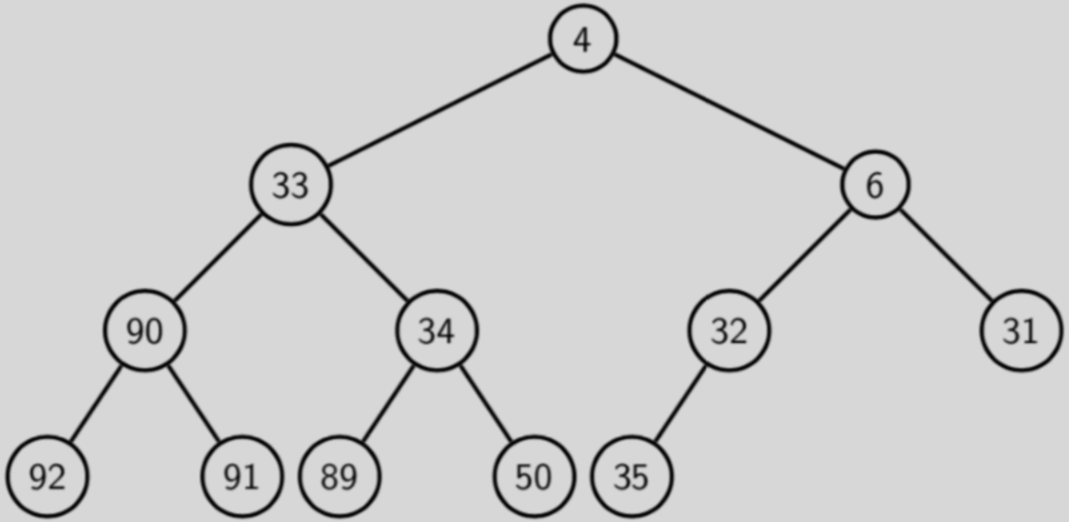
(a) Consider the following min-heap represented by an array:

$$A = [4, 33, 6, 90, 34, 32, 31, 92, 91, 89, 50, 35]$$

Justify the claim that A is a min-heap by drawing it as a tree and briefly explaining the min-heap property. **[3 Marks]**

Solution (a)

It is a min-heap because every node has a smaller key than any of its childern [1 mark].



[2 marks]

(b) Consider an arbitrary AVL tree. For each of the following two cases, does one always return to the original AVL tree after performing the operations? Justify your answers.

i. Insert an element which is not in the AVL tree and then delete this element. **[4 Marks]**

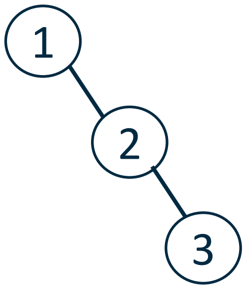
ii. Delete an element from the AVL tree and then insert it. **[4 Marks]**

Solution (b)

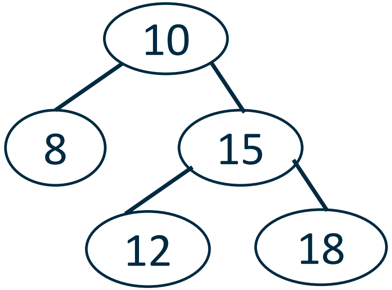
i. No [1 mark]. If the insertion causes a rotation then the tree changes. One example is the tree with root 1 and a leaf 2 - inserting 3 makes the tree change the root to 2, which stays the root after 3 is deleted. [3 marks]

ii. No [1 mark]. If you delete the root and then insert it, it will be inserted as a leaf. One example is the tree with root 2 and leaves 1 and 3. [3 marks]

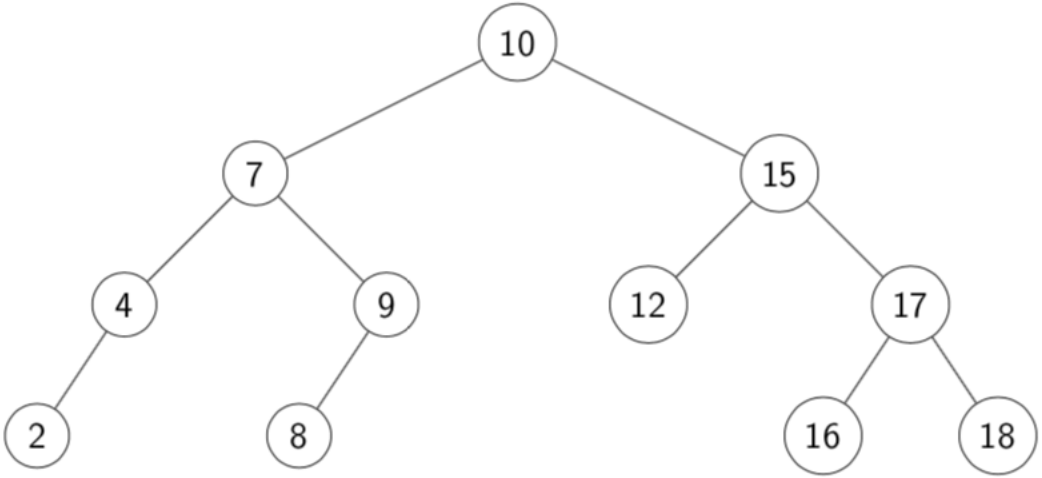
(i)



(ii)



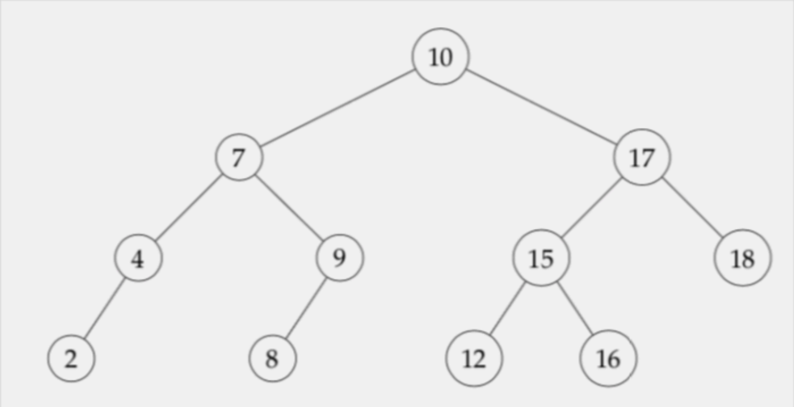
(c) Consider the following BST:



Draw the BSTs obtained from it first by performing left rotation on 15 and then (on the resulting tree) right rotation on 10. **[6 Marks]**

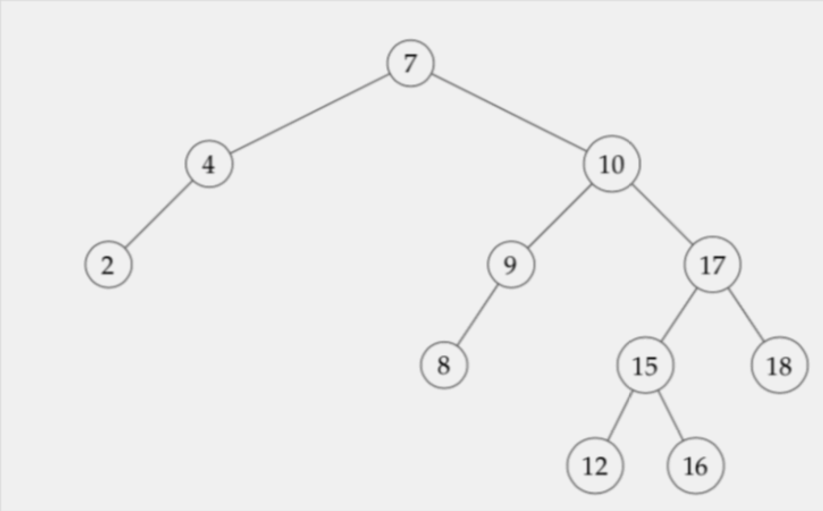
Solution (c)

Tree after left rotation on 15:



[3 marks]

Tree after right rotation on 10:



[3 marks]

Durham University

(d) i. Manually run the RadixSort algorithm with base-2 representation on the following array (where all numbers are given in base-2):

$$(11001)_2, (11101)_2, (101)_2, (10000)_2, (1011)_2, (11110)_2.$$

[5 Marks]

ii. Is Median-of-Medians an optimal selection algorithm (in some specific sense)? **[3 Marks]**

Solution (d) (i)

i.

Everything is done base-2, so I'll drop the subscript in all numbers.

First pass (right-most digit) gives the following buckets:

	(0)1011
	(00)101
11110	11101
10000	11001

. . . which, in turn, gives the "new" array

$$10000, 11110, 11001, 11101, (00)101, (0)1011$$

Now the same for the second digit from the right.

(00)101	
11101	
11001	(0)1011
10000	11110

. . . which, in turn, gives the "new" array

$$10000, 11001, 11101, (00)101, 11110, (0)1011$$

Now the same for the third digit from the right.

(0)1011	11110
11001	(00)101
10000	11101

. . . which, in turn, gives the "new" array

$$10000, 11001, (0)1011, 11101, (00)101, 11110$$

Solution (d) (i)..

Now the same for the fourth digit from the right.

	11110
	11101
(00)101	(0)1011
10000	11001

...which, in turn, gives the "new" array

$$10000, (00)101, 11001, (0)1011, 11101, 11110$$

Finally, the same for the fifth digit from the right.

	11110
	11001
(0)1011	11001
(00)101	10000

...which gives the sorted array.

$$(00)101, (0)1011, 10000, 11001, 11101, 11110$$

[5 marks] in total – [1 mark] for each correct pass.

Solution (d) (ii)

ii. The running time of MoM is $O(n)$[1 mark] (as proved in lectures), which is asymptotically optimal [1 mark] because any selection algorithm, given an array of n elements, must inspect every element in the array (or else the required i-th smallest element can be the one not inspected), which takes $\Omega(n)$ time [1 mark].

Durham University

That's it. Please complete MEQs.

All the best for the exam!!

Thank you!