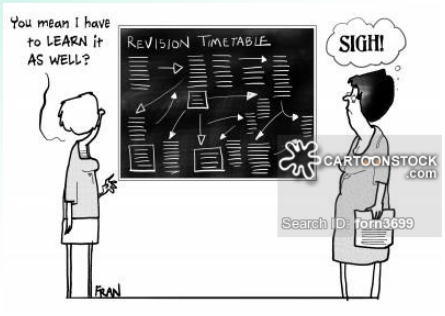
CompSci 105
Lecture 36 – Revision Week 9-12

NOTE: All exam information without guarantee and subject to change.



Exam Information

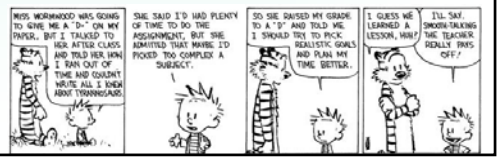
- 2 hours
- Closed book, no calculator
- 55 questions, each worth between 1 and 2 points (100 points in total)
- All multiple-choice questions (5 choices like in mid-term test)
- Part 1 (Angela): 16 points
  Part 2 (Bruce): 42 points
  Part 3 (Burkhard): 42 points (21 questions)
- All material from the lectures, assignments and labs is relevant unless specifically excluded (for example, AVL trees are not part of the exam)
- Both questions and answer choices may contain Python code

Why MCQ?

- Because multiple choice exams contain many questions, they require students to be familiar with a much broader range of material than open answer exams do
- Multiple choice exams also usually expect students to have a greater familiarity with details
- Lower risk for students – since there are more questions, misunderstanding/misreading a question has less severe results
- Faster to mark (results are out earlier)
- Research and our own experience shows that a good MCQ exam is as effective as a traditional exam

Exam Technique

- Get enough sleep before the exam
- Plan your exam – identify easy questions and do them first
  => this boosts your confidence and avoids that you loose easy points because you run out of time
- When reading the question cover up the answer choices – anticipate answer before seeing the possible answers
- If you see expected answer circle it, but check out other answers whether one of them is better
- If you can't answer a question (say, 1-2 min) come back to it later
- If you run out of time at the end, do informed guesses
- Don't panic – remember that everyone has to answer the same questions ☺



Hashing – Example 1

Given is an initially empty hash table of size 7:
`[None, None, None, None, None, None, None]`
What does the hash table look like after inserting the keys `21, 13, 20, 27, 2` in this order using the hash function `h(key) = key % 7` ?

Using linear probing: `[21, 20, 27, 2, None, None, 13]`

Using quadratic probing: `[21, 27, 2, 20, None, None, 13]`

Using double hashing with `h2(key) = 5 - key % 5` :

`[21, None, 27, None, 20, 2, 13]`

Hashing – Example 2

Modify the hash table for double hashing such that it stores for each inserted key its probe sequence:

```
class BasicHashTable:
    def __init__(self,size=7):
        self.size = size
        self.slots = [None] * self.size
        self.probeSequences = [[]] * self.size

    def put(self, key):
        hash_value = self.hash_function(key)
        if self.slots[hash_value] == None:
            self.slots[hash_value] = key
            self.probeSequences[hash_value] = [hash_value]
        else:
            currentProbeSequence = [hash_value]
            next_slot = self.rehash(hash_value,key)
            currentProbeSequence.append(next_slot)
            while self.slots[next_slot] != None and self.slots[next_slot] != key:
                next_slot = self.rehash(next_slot,key)
                currentProbeSequence.append(next_slot)
                if next_slot == hash_value:
                    return
            if self.slots[next_slot] == None:
                self.slots[next_slot] = key
                self.probeSequences[next_slot] = currentProbeSequence
```

Sorting – Example 1

What does the list `[67, 9, 55, 1, 22, 77, 11, 49]` look like after the first two passes with Bubble Sort, Selection Sort, and Insertion Sort?

Bubble Sort: `[9, 55, 1, 22, 67, 11, 49, 77]`
`[9, 1, 22, 55, 11, 49, 67, 77]`

Selection Sort: `[67, 9, 55, 1, 22, 49, 11, 77]`
`[11, 9, 55, 1, 22, 49, 67, 77]`

Insertion Sort: `[9, 67, 55, 1, 22, 77, 11, 49]`
`[9, 55, 67, 1, 22, 77, 11, 49]`

Sorting – Example 2

The Selection Sort algorithm takes $O(n^2)$ time. How can we modify the algorithm such that it finds the k largest values of a list in O(kn) time and stores them on the right of the input list?

```
def k_largest(a_list,k):
    for pass_num in range(len(a_list) - 1, len(a_list) – 1 - k, -1):
        position_largest = 0
        for i in range(1, pass_num+1):
            if a_list[i] > a_list[position_largest]:
                position_largest = i
        swap_elements(a_list, position_largest, pass_num)
```

Sorting – Example 3

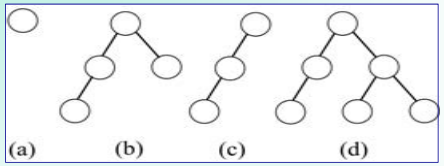
How can we change the Shell Sort algorithm to sort a list of values in decreasing order?

```
def shell_sort(a_list):
    gap = len(a_list) // 2
    while gap > 0:
        for start_position in range(gap):
            gap_insertion_sort(a_list, start_position, gap)
        print("for gap: ", gap, " - ", a_list)
        gap = gap // 2

def gap_insertion_sort(a_list, start, gap):
    for i in range(start + gap, len(a_list), gap):
        current_value = a_list[i]
        position = i
        while position >= gap and a_list[position - gap] < current_value:
            a_list[position] = a_list[position - gap]
            position = position - gap
        a_list[position] = current_value
```

Binary trees – Example 1

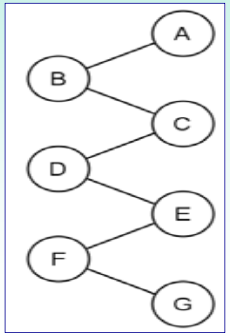
Consider the following binary trees. For each binary tree, indicate if it is complete, full and/or balanced.



(a)Complete:	yes/no	(c)Complete:	yes/no
Full:	yes/no	Full:	yes/no
Balanced:	yes/no	Balanced:	yes/no
(b)Complete:	yes/no	(d)Complete:	yes/no
Full:	yes/no	Full:	yes/no
Balanced:	yes/no	Balanced:	yes/no

Binary trees – Example 2

The following diagram shows a binary tree with the root node containing the value, A. Write the pre-order, in-order and post-order traversals of the following binary tree.



pre-order: **A B C D E F G**

in-order: **B D F G E C A**

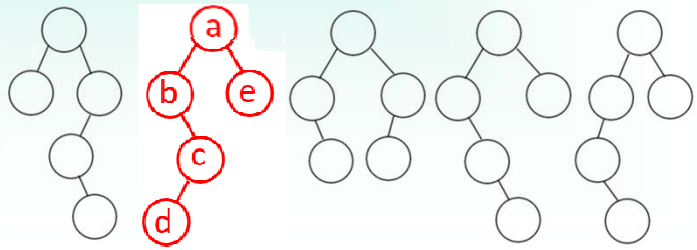
post-order: **G F E D C B A**

Binary trees – Example 3

Given a tree with the following traversal sequences:

Inorder: b d c a e
Postorder: d c b e a

What is the shape of the tree?



Binary trees – Example 4

Reconstruct a tree from its inorder and preorder traversal sequence:

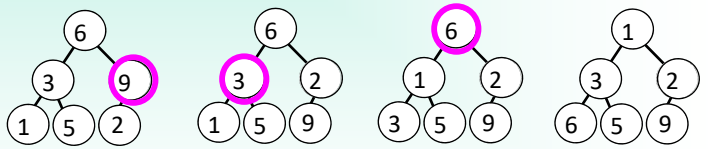
```
from ListBinaryTree import ListBinaryTree

def buildTree(inorder,preorder):
  if (len(preorder)==0):
    return None
  else:
    topElem = preorder[0]
    leftLen = inorder.index(topElem)
    leftTree = buildTree(inorder[0:leftLen],preorder[1:leftLen+1])
    rightTree = buildTree(inorder[leftLen+1:],preorder[leftLen+1:])
    return ListBinaryTree(topElem,leftTree,rightTree)

def main():
  print("Binary Tree reconstructed by abcd001:")
  inorder = input("Please enter the inorder sequence: ")
  preorder = input("Please enter the preorder sequence: ")
  if (len(inorder) != len(preorder)):
    print("Error: Input strings have different length")
    exit(-1)
  tree = buildTree(inorder,preorder)
```

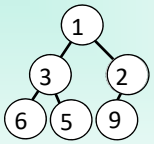
Heap – Example 1

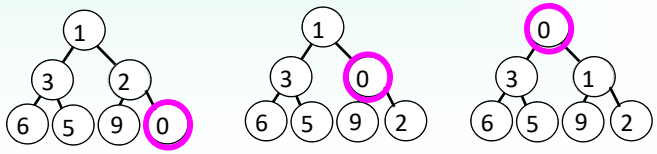
Construct a min-heap from the list [6, 3, 9, 1, 5, 2] using the O(n) method discussed in the lecture. Draw the resulting heap as a binary tree after each step.



Heap – Example 2

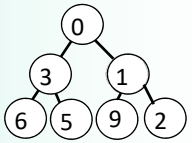
Insert the value 0 into the heap below using the method discussed in the lecture . Draw the heap as a binary tree after each step.

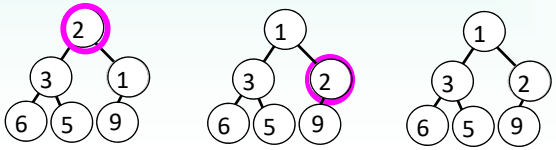




Heap – Example 3

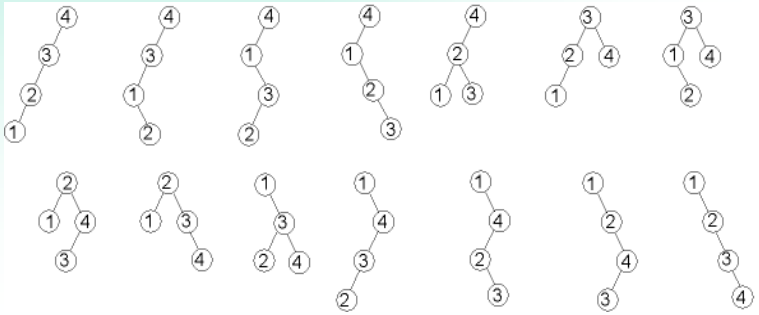
Delete the minimum (highest priority) value from the heap below using the method discussed in the lecture . Draw the heap as a binary tree after each step.





Binary Search trees – Example 1

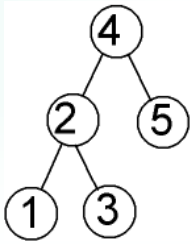
Draw all possible binary search trees obtained by inserting the values 1,2,3,4 in different orders into an initially empty tree



Binary Search trees – Example 2

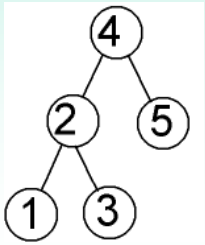
Give all possible orders of the values 1,2,3,4,5, which result in a complete binary search tree when inserted into an initially empty tree.

[4,2,1,3,5]
[4,2,1,5,3]
[4,2,3,1,5]
[4,2,3,5,1]
[4,2,5,1,3]
[4,2,5,3,1]
[4,5,2,1,3]
[4,5,2,3,1]



Binary Search trees – Example 3

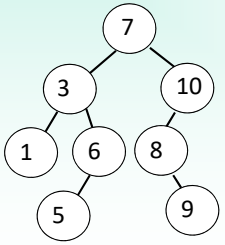
Given is a binary search tree and its level-order, inorder, preorder, and postorder traversal sequence. Assumed we insert the values into an initially empty binary search tree in the traversal order. Which of these traversal sequences will result in the original tree?



Level-order: 4 2 5 1 3 ✓
Inorder: 1 2 3 4 5 ✗
Preorder: 4 2 1 3 5 ✓
Postorder: 1 3 2 5 4 ✗

Binary Search trees – Example 4

Draw the binary search tree structure after inserting the following integer search key values into an empty binary search tree in the order given: **7, 3, 1, 6, 5, 10, 8, 9**



Binary Search trees – Example 5

What does the tree on the right look like after deleting in this order the nodes:

(1) 10

(2) 5