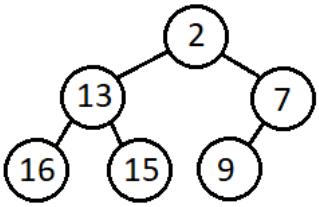
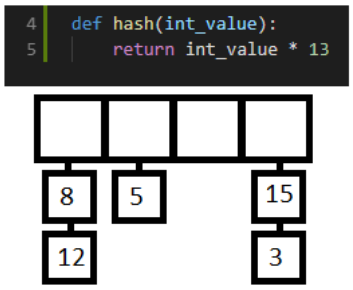
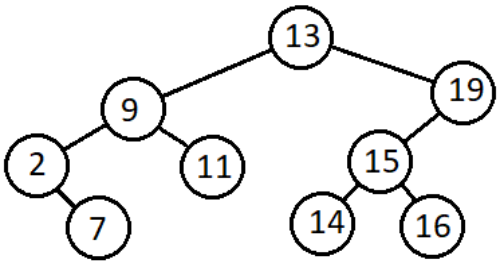
Start this exercise in groups of 4-6 students.

Images for part 1:







Part 1:

Split your group into three smaller groups (1-2 students) each of which takes one of the following problems. If two students are in a small group, each does the whole exercise.

In each exercise draw the data structure and, using a series of images and/or clear labelling on the image(s), show in as much detail (small steps) as possible what happens in each operation.

1. Look at the image of a heap
    a. What happens when the value 23 is added?
    b. What happens when the value 1 is added?
    c. What happens when the value 14 is added after the value 1 has been added?
    d. What happens when a value is removed?

2. Look at the image of a hash table
    a. What happens when the value 9 is added?
    b. What happens when the value 18 is added?
    c. What happens when the value 12 is removed?
    d. What happens when the value 7 is removed?

3. Look at the image of a binary search tree
    a. What happens when the value 21 is added?
    b. What happens when the value 10 is added?
    c. What happens when the value 17 is added?
    d. What happens when the value 19 is removed?
    e. What happens when the value 13 is removed?

Now students show the members of their larger group their solutions and explain in exact detail the steps of each operation. If more than one student did the same exercise they take turns explaining the operations (not both the same ones).
Other students should feel free to add detail if they feel something is being "jumped over".

DON'T START IMPLEMENTING ANYTHING YET!! - read the next page first
Problems for parts 2 and 3:

Read both part 2 and 3: different students will design tests for a problem and implement it.

A. *Build a data structure that uses an **array** for underlying items*
    - Make an operation that adds to the data structure
    - Make an operation that returns a string representing all the items
        - In the order in which they were added
    - Make an operation that removes all duplicates
B. *Build a data structure with a **linked list** as data storage*
    - Make an operation that adds to the data structure
    - Make an operation that returns a string representing all the items
        - In the order in which they were added
    - Make an operation that returns all values within a certain range
C. *Build a data structure with a **binary tree** as storage*
    - Make an operation that adds to the data structure
    - Make an operation that returns a string representing all the items
        - The order is not important
    - Make an operation that returns the lowest value
    - Also make an operation that returns the highest value

Read both part 2 and 3: different students will design tests for a problem and implement it.

Part 2:

Split your group into three smaller groups (1-2 students) each of which picks one of the problems. If two students are in a small group, they can cooperate.
It is also OK to only have two smaller groups and only pick two of the assignments.

Each smaller group does the following:

- Decide on the names of classes and public operations
    - Public operation are the ones that will be called from outside the class
    - Private operations are the ones that are only used internally (helpers)
- Set up these classes and operations in python code
    - Use “pass” if operation does nothing
    - If operation should return a value, return a dummy value of the correct type.
- Design tests for the operations and implement the tests in python code
    - Call the actual class and make sure everything works
        - Even though it will not return correct results
    - Make sure the tests evaluate every possibility
        - Think about “edge cases”
    - Know what you expect the tests to return
        - Or that you can easily see from the output what should be returned

Now the big group meets again and each smaller group explains their tests.

- Give detailed reasons for your “edge cases”.
    - What is the possible case
    - Why is it different from the general case
    - How does a particular test evaluate exactly this case

After each explanation the larger group discusses the tests and tries, collectively, to identify cases that these tests do not evaluate well enough. Get into details here!!

- The smaller group now updates/rewrites their tests after the group review.

Part 3:

Again split the group into smaller groups.
Each smaller group picks one of the problems that has a class definition and set of test cases from Part 2, but **not** one that any member of this smaller group worked on in Part 2.
Every student should thus be in a group that starts with implemented test cases that they didn't implement themselves.

Each group does the following:

- Design the operations, variables needed, helper classes, etc.
    - Define all this on a code level, but don't necessarily get the code working perfectly.
    - Write explanations of the operations in very good detail
        - Comments and pseudo-code
        - Diagrams
    - Write this up in a way that it can be understood without personal explanations

Now each group takes another group's design

- Seek to understand it
- Try to point out issues
    - Some case that is forgotten or left out
    - An endless loop that might occur
    - Somewhere no value will be returned
    - A change that could make the whole structure more efficient

The larger group meets again to discuss the review of all designs

- In each solution, discuss possible ways of doing it differently
    - whether or not it's necessarily better.
- Also discuss ways it might have been worse

Groups take back their design, fix it and **implement it**

- Now they have actual working code that should run correctly through tests from **part 2**

Now groups switch their code and try to point out issues (and also good things)

- Is it well readable and understandable
- Is there something that is unnecessarily complex
- Is there some hidden time complexity that might have been missed
    - Like what is the time complexity of built-in operations or python magic?

Now the groups can take their code back or the whole big group can cooperate on making each solution as good as it can be...