COMP 2402 Abstract Data Types and Algorithms

Study Session Questions

December 28, 2025

Learning Objectives

- You should be confident analyzing array-based structures, including `ArrayList`, `Queue/Deque`, `DualArrayDeque`, and `RootishArrayStack`.
- You should be confident analyzing linked list structures, including `SLL/DLL`, `SEList`, and `SkipList`.
- You should be confident analyzing tree-based structures, including `RandomBSTs`, `Scapegoat Trees`, `2-4/Red-Black Trees`, and `Treaps`.
- You should be confident analyzing other core structures, including `Heaps`, `Graphs`, `PriorityQueues`, `Tries`, and `HashTables`.
- You should be comfortable with runtime and amortized analysis for these data structures and sorting algorithms.

I. Array-Based Lists

- Contiguous blocks of memory - easy O(1) access to values
- Adding or removing in the middle of a list requires shifting elements - variable runtimes
- Cannot expand or shrink without allocating a new array and copying values to new array - expensive!

1. If we resize a backing array in an `ArrayStack` via `grow()`, what conclusion can be drawn?
    A. At least $\frac{n}{2}$ `add()` operations have occured since then
    B. At least $\frac{2n}{3}$ `remove()` operations have occured since then
    C. At least $\frac{2n}{3}$ `add()` operations have occured since then
    D. At least $\frac{n}{2}$ `remove()` operations have occured since then
    E. We cannot bound the number of `add()` nor `remove()` operations
2. Recall that we shrink the backing array a when $3n < a.\texttt{length}$. If we are currently about to shrink the array...
    A. At least $\frac{n}{2}$ `add()` operations have occured since then

B. At least $\frac{2n}{3}$ `remove()` operations have occured since then

C. At least $\frac{2n}{3}$ `add()` operations have occured since then

D. At least $\frac{n}{2}$ `remove()` operations have occured since then

E. We cannot bound the number of `add()` nor `remove()` operations

3. What is the minimum number of elements a `RootishArrayStack` of block size 9 can hold without needing to shrink?

A. 28

B. 29

C. 36

D. 37

E. 45

F. 46

4. In a `RootishArrayStack`, what will a call to `get(10)` (get the element with index 10) return?

A. `blocks.get(0)[10]`

B. `blocks.get(5)[1]`

C. `blocks.get(3)[4]`

D. `blocks.get(4)[0]`

E. `blocks.get(4)[1]`

F. `blocks.get(4)[4]`

5. A `DualArrayDeque` uses two `ArrayStacks`, `front` and `back`. What's the implementation for `get(i)` ?

A. A: `front.get(i)`

B. Either A or B depending on the value of i and `front.size()`

C. B: `front.get(front.size() - i - 1)`

D. Either A or C depending on the value of i and `front.size()`

E. C: `back.get(i - front.size())`

F. Either B or C depending on the value of i and `front.size()`

6. Explain amortization in your own words.

II. LinkedList Structures

- Rather than O(1) access we must walk through the list, one at a time, until the ith element is reached
- Easier insertion and deletion

7. Which of the following methods are usually **much faster** on a LinkedList versus an ArrayList? Select all that apply.
   - ◯ read the first element of a list n times
   - ◯ read a newly-randomly-indexed element of the list n times
   - ◯ insert n elements at the end of a list
   - ◯ insert n elements at the beginning of a list
   - ◯ insert n elements at index $\frac{size}{2}$ (which doesn't change)
   - ◯ none of these

8. Consider a SEList with $b = 5$. The last element of the last block has index 14. How many blocks *could* the list have?
   - A. two
   - B. three
   - C. four
   - D. five
   - E. six
   - F. seven

9. In a Doubly-Linked List, how is traversal direction optimized in `getNode(i)` ?
   - A. The list will randomly traverse from the head or tail to amortize speed
   - B. The list always traverses from head to save space
   - C. The list always traverses from tail to save space
   - D. A recursive traversal is implemented
   - E. The direction (head or tail) depends on the value of i

10. Which of the following statements are true with regards to SkipLists? Select all that apply.
    - ◯ The expected time complexity of `get(i)` is $O(\log n)$.
    - ◯ The worst-case time complexity of a find operation is $O(n^2)$.
    - ◯ The level/height of a node in a SkipList is determined by random chance.
    - ◯ The "sentinel" node in a SkipList always holds the largest element.
    - ◯ The maximum level of any node grows as $O(\log n)$.
    - ◯ The expected time complexity of finding the last element is $O(1)$.

11. Explain the role of the dummy node. In particular, what are dummy.next and dummy.prev?

Part III - Trees -

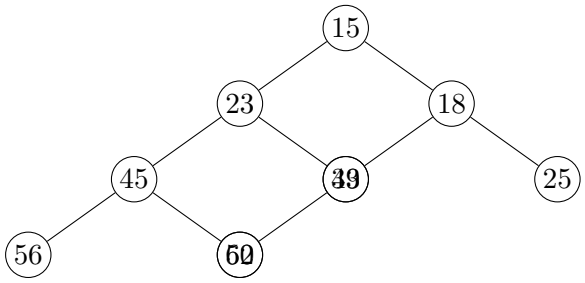
> "If you want to know why the tree grows upside down, ask the computer scientists who introduced this convention. (The conventional wisdom is that they never went out of the room and so they never saw a real tree.)"

12. Match each type of tree with its corresponding description.

A. Treap	**1.** randomized balancing with expected $O(\log n)$ performance using priorities
B. Scapegoat Tree	**2.** maintains balance by rebuilding entire subtrees when imbalance is detected
C. Red-Black Tree	**3.** binary implementation of a 2-4 tree, guaranteed $O(\log n)$ height
D. 2-4 Tree	**4.** fewest tree levels by allowing multi-key nodes, multiple splits per step

13. If we `add(19)`, then `removeMin()` to the heap below, what will the heap look like afterwards as an array?

`15, 23, 18, 45, 49, 33, 25, 56, 62, 50`



A. `[19, 18, 25, 45, 50, 33, 49, 56, 62, 23]`

B. `[23, 18, 45, 49, 33, 25, 56, 62, 50, 19]`

C. `[18, 19, 25, 45, 23, 33, 49, 56, 62, 50]`

D. `[18, 23, 19, 25, 33, 45, 49, 50, 56, 62]`

E. `[18, 19, 45, 23, 33, 25, 56, 62, 50, 49]`

F. none of these

14. Match each type of tree traversal with its implementation.

A. preorder	**1.** Visit root, then left subtree always, then right subtree.
B. inorder	**2.** Visit left subtree, root, then right subtree.
C. postorder	**3.** Visit from the leaves to the root, prioritizing left subtrees first
D. breadth-first	**4.** Visit level by level, from root to leaves, prioritizing left subtrees

15. What is the time complexity to rebuild a subtree of size s in a scapegoat tree?

A. $O(1)$

B. $O(\log s)$

C. $O(s \log s)$

D. $O(s)$

E. $O(s^2)$

F. $O(2^s)$

16. In a 2-4 tree, what happens when inserting a key causes a node to exceed its maximum of 3 keys (i.e., 4 children)?

A. The tree is restructured from the root down to evenly redistribute all keys

B. The node's subtree is rebalanced by rotating keys with siblings

C. The node is split into two nodes, and the middle key is promoted to parent

D. The insertion is deferred until the next rebalancing phase

IV. Everything Else

- Graphs: Nodes with edge links
- BinaryTries: prefix tree for binary strings
- HashTables: Key-value store via hash formula
- MeldableHeap: heap w/ random merge path
- PriorityQueues: Remove item with top rank

17. Recall

$$h(x) = ((z \cdot x) \bmod 2^w) \text{ div } 2^{w-d}$$

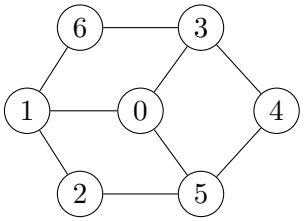
Suppose $w = 5, d = 3, z = 9$. What is $h(6)$?

A. $h(6) = 2$
B. $h(6) = 3$
C. $h(6) = 4$
D. $h(6) = 5$
E. $h(6) = 6$
F. $h(6) = 7$

18. Put the following steps in correct order for melding two heaps $h1$ and $h2$.

1. If one heap is empty, return the other
2. Compare root values of $h1$ and $h2$
3. Swap $h1$ and $h2$ if $h2$ has a smaller root
4. Recursively meld $h1$'s right child with $h2$

19. Perform a BFS on this graph, starting from 0. Indicate the order of vertices traversed (i.e. 0 1 2 3 4 5)



20. If x is not in a `BinaryTrie`, what does `find(x)` do?
    A. Returns null
    B. Returns a random key
    C. Finds the smallest key greater than x
    D. Finds the root value

21. In a chained hash table, what happens when two keys hash to the same index?
    A. One key is discarded
    B. A binary search tree is created
    C. Both are stored in a linked list at that slot
    D. A hash collision error is thrown

V. Sorting Algorithms and Runtime Efficiency

No description

22. Place the following runtime complexities in order of relative growth, starting with the slowest.
    - $O(\log n)$
    - $O(n)$
    - $O(n \log n)$
    - $O(n^2)$

23. How does QuickSort work?
    A. Splits array in half and merges sorted parts
    B. Repeatedly selects minimum and moves it to front
    C. Picks a pivot, partitions array, recursively sorts parts
    D. Builds a heap and extracts elements one by one

24. What is the space complexity of MergeSort (recall that it is not in-place)?
    A. $O(1)$
    B. $O(\log n)$
    C. $O(n)$
    D. $O(n \log n)$

25. What is the time complexity of HeapSort in the worst case?
    A. $O(\log n)$

B. $O(n)$

C. $O(n \log n)$

D. $O(n^2)$

26. How much more review do you think you'll have to do for the exam?
    none at all / a lot more