上海科技大学
ShanghaiTech University

Algorithms and Data Structures
Midterm Review
Part (1)

**Adapted from CS101 Midterm Review (Made by TA Keyi Yuan), 2018 Fall, ShanghaiTech University*

**Made by TA Yao Shen*

Claim

1. Topics that reviewed in this discussion may not be covered in the Exam
2. Topics that not reviewed in this discussion may be covered in the Exam

Checklist – Linked List

- Knowledge
  - The implementation of Node and Linked List Class
  - The operations and complexity of Linked List functions
    - Find, Insert before/after, Erase …
  - Optimizations of Linked List functions
    - Insert before, Erase
  - Comparison between Linked List and Array
    - Array: Fast accessing speed, suitable for stable data …
    - Linked List: Suitable for data with frequent modifications …

Checklist – Stack and Queue

- Knowledge
  - Stack: Last-in-first-out (LIFO)
    - Operations: Push (at top) and Pop (from top)
    - Implementations: singly linked list, one-ended array
  - Queue: First-in-first-out (FIFO)
    - Operations: Push (at back) and Pop (from top)
    - Implementations : (1) singly linked list (with tail pointer) and two-ended array
      (2) doubly linked list and circular array
- Applications:
  - Stack: Reverse-Polish Notation, Function calls, Parsing XHTML …
  - Queue: …

Checklist – Algorithm Analysis

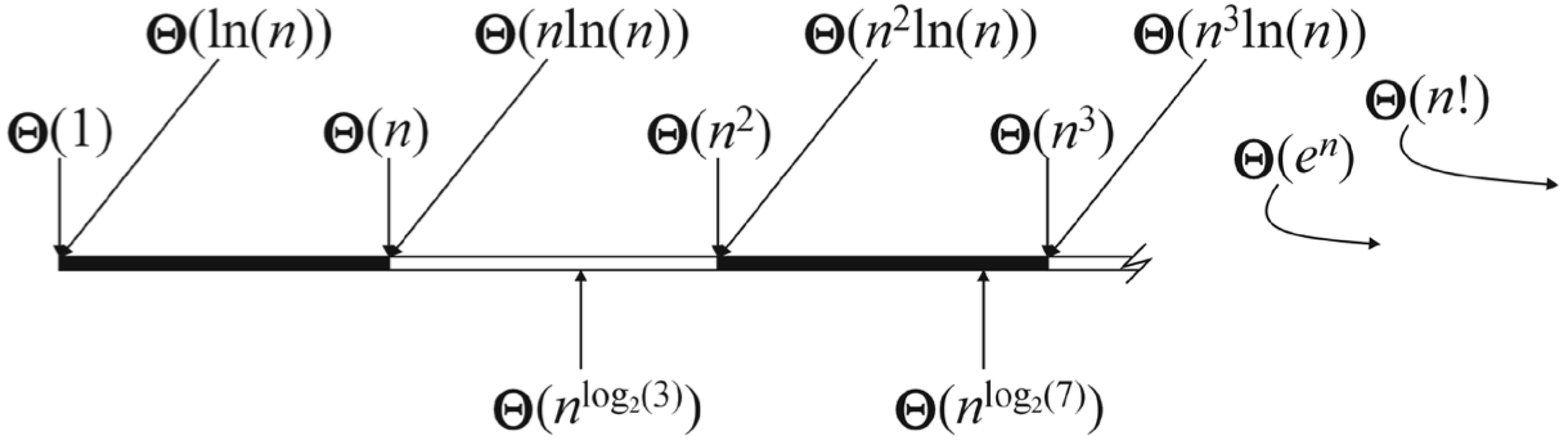
- Knowledge
  - Understand the definition of Landau Symbols : O, Θ, Ω
  - Understand how to use the notation to represent time complexity
    - E.g. $f(n) = O(n^3)$
    - Please write your most accurate answer in your exam!
      - E.g. Insertion sort time complexity: write $O(n^2)$ instead of O(n!)
  - Solve the time complexity of recursive functions with substitution method
  - E.g. $T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$

    Guess $T(n) = O(n \log n)$: $\exists c,\ s.t.,\ T(n) \le cn \log n$

    Substitute: $T(n) \le 2\left(c\left\lfloor \frac{n}{2} \right\rfloor \log\left\lfloor \frac{n}{2} \right\rfloor\right) + n \le cn \log \frac{n}{2} + n = cn \log n + (1-c)n$

Checklist – Algorithm Analysis

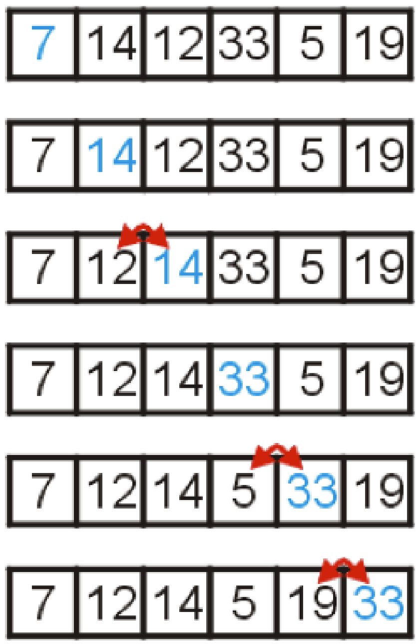
- Knowledge



Checklist – Bubble Sort

- Knowledge
  - Understand algorithm of bubble sort
    - Starting with the first item, assume that it is the largest
    - Compare it with the second item:
      - If the first is larger, swap the two
      - Otherwise, assume that the second item is the largest
    - Continue up the array, either swapping or redefining the largest item

    (After one pass, the largest item must be the last in the list)
    - Start at the front again:
      - Repeat n – 1 times, after which, all entries will be in place



Checklist – Bubble Sort

- Knowledge
  - Time complexity

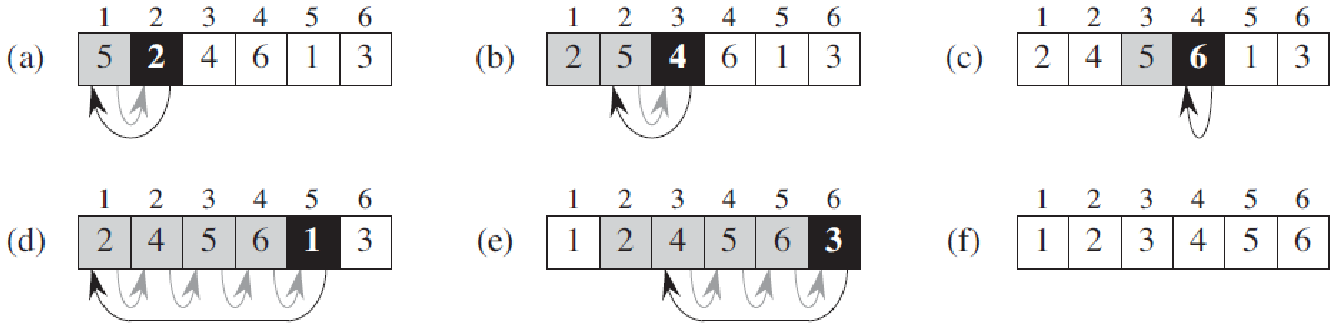
$$\sum_{k=1}^{n-1}(n-k)=n(n-1)-\frac{n(n-1)}{2}=\frac{n(n-1)}{2}=\Theta(n^2)$$

Checklist – Bubble Sort

- Knowledge
  - Optimizations:
    - Flagged Bubble Sort:
      - halting if the list is sorted
    - Range-limiting Bubble Sort:
      - limiting the range on which we must bubble
    - Alternating Bubble Sort:
      - alternating between bubbling up and sinking down

Checklist – Insertion Sort

- Knowledge
  - Inversion:
    - *a pair of entries which are reversed(a.k.a: (a_j, a_k) if j < k but $a_j > a_k$ for ascending order).*
  - Algorithm of Insertion Sort



Checklist – Insertion Sort

- Knowledge
  - Time complexity: $O(n^2)$
    - Insertion sort which does $n + d$ comparisons where d is the number of inversions.
    - If the inversion count is O(n), then the time complexity of insertion sort is O(n).
    - In worst case, there can be n*(n-1)/2 inversions. (totally reversed). So the worst case time complexity of insertion sort is $O(n^2)$.
  - Bubble sort cannot be any better than insertion sort

Checklist – Tree

- Tree Structure
  - Root, parent, children.
  - Degree: the number of its children.
  - Leaf: nodes with degree zero.
  - Path: a sequence of nodes: (a0, a1, ..., an)
  - Height: the maximum depth of any node. The height of a tree with one node is 0. (You should care about that the height is the maximum nodes on path minus 1.)
  - Ancestor, descendant.

Checklist – Tree Traversal

- Tree Traversal
  - Without specification, traverse from left to right !
  - BFS
    - Θ(n) run time and o(n) memory, maximum nodes at a given depth.
    - Using a queue.
  - DFS
    - Θ(n) run time and Θ(h) memory, h is the height of tree.
    - Using recursion algorithm, based on a stack. (Inverse order)
  - Pre-ordering / Post-ordering / In-ordering
    - Given in-order and pre-order, how to get post-order? First, the first element in pre-order is the root, then divide post-order into 2 parts by the root.
    - For more information, you can search on the Internet.
    - Recall pre-order: root, left, right; post-order: left, right, root.; in-order: left, right, root

Checklist – Binary Tree

- Binary Tree
  - Each node has at most two children.
  - Full binary tree
  - Complete binary tree: A complete binary tree filled at each depth from left to right.
  - Recursive Definition:
    - The left sub-tree is a complete tree of height h – 1 and the right subtree is a perfect tree of height h – 2, or
    - The left sub-tree is perfect tree with height h – 1 and the right sub-tree is complete tree with height h – 1
  - There is no relationship between full and complete binary tree.
  - The worst case: Θ(n); the best case: Θ(ln(n)).
  - Perfect binary tree: height h with $2^{(h+1)}$-1 nodes.
  - Array storage may not be the best solution.

Checklist – Binary Heap

- Binary Heap
  - Min-Heap: The key associated with the root is less than or equal to the keys associated with the sub-trees, and the sub-trees (if any) are also min-heaps. (Recursive definition)

    There is not relationship between children!
  - Max-Heap: …
  - We will do better in complete binary tree. (Avoid unbalanced binary tree.)
  - We may store a complete tree using an array. E.g. Ignore the index = 0. And then, for index I, its parent is $i/2$, and its children is $2i$ and $2i + 1$.
  - Operations: Top, Pop, Push.
  - For push, insert at the back, and compare the value of its parent.
  - For pop, delete the index = 1, and then copy the last entry to the top. Then compare the value of its both children.
  - Access: Θ(1), pop and push: O(ln(n))
  - Space complexity: O(n)

Checklist – Binary Heap

- Binary Heap
  - Build heap: Floyd's Method
    - No percolation for the leaf nodes (n/2 nodes)
    - At most n/4 nodes percolate down 1 level
      At most n/8 nodes percolate down 2 levels
      At most n/16 nodes percolate down 3 levels
      …

$$1\frac{n}{4} + 2\frac{n}{8} + 3\frac{n}{16} + \cdots = \sum_{i=1}^{\log n} i\,\frac{n}{2^{i+1}} = \frac{n}{2}\sum_{i=1}^{\log n}\frac{i}{2^i} = n = \Theta(n)$$

Checklist – Heap Sort

- Binary Heap
  - Place the objects into a heap (Floyd’s Method)
    - $O(n)$
  - Repeatedly popping the top object until the heap is empty
    - $O(n \ln(n))$
  - Time complexity: $O(n \log n)$