C949 Study Guide

Data Types 36%

Abstract Data Types and Data Dictionaries

Terms to know

- **Data structures**
  - array
  - linked list
  - binary search tree
  - hash table
  - heap

- **Abstract data types** (see table 3.4.1) Know how they function, what makes them distinct from each other (you'll need to choose between at least two similar choices), the underlying data structure, and the basic Python commands and syntax, e.g., push, pop, {} , [], etc. Creating organizational tables like table 3.6.1 may help.
  - list (and Array in Java)
  - tuple
  - stack
  - queue
  - deque
  - bag
  - set
  - priority queue
  - dictionary (map). The pre-assessment has 5 problems (7%) about dictionaries.

- **Data types** Know what they are and the common syntax used.
  Java data types
  Python data types; quiz Python Data Types
  - boolean
  - byte
  - int
  - char
  - float
  - double
  - String

- **Intro programming concepts**

    - Assignment vs. comparison
    - Garbage collection
    - reference count
    - memory allocation
    - linked allocation
    - sequential allocation
    - pointer
    - binary search. Know how to conduct a binary search on an array/list (see problem 19 and 21 on the pre-assessment).
    - null
    - object constructor

Data Structures 30%

Design of Data Structures

Design properties. Know the principal design properties of the listed data structures, e.g., how things are compared, inserted, deleted, which have indexing, which has hierarchies, etc.

- **Linear Data Structures**
    - Arrays
    - Linked list.
        - linked list
        - Doubly linked list
    - Queue
    - Stack

- **Hierarchical Data Structures**
    - Tree (binary)
    - Heap
        - child and parent
        - heapList
        - right/left child
        - min-heap
        - max-heap
- Graph data sturctures
    - weight and direction
    - vertices/nodes and edges/node pairs
    - directed/undirected graph
- Indexing of data structures

Implementation of Data Structures

Data structure functions. From the pre-assessment, it appears this section will focus mostly on the basic functions of stacks, queues, and dictionaries with a focus on .pop() and .push()

- Know what .pop() does for:
  - stacks
  - queues
  - priority queues
- Be familiar with tree traversal
  - inorder traversal
  - preorder traversal
- Know the major dictionary functions (see table 8.2.1)
- Hash tables
  - hashing
  - chaining
  - hash key
  - modular arithmetic (see pre-assessment problem 36)

Algorithms 34%

Introduction to Algorithms

- You will need to be able to read pseudocode (which may be actual Java or C code). For these problems, practice carefully writing out the steps on your whiteboard. One misstep will give you the incorrect choice! This is the hardest section. First focus on Data Types and Data Structures.

Example:

48 What is displayed when n = 2 in this pseudocode?

```
for(int i = 2; i <= n; i++){
for(j = 0; j <= n;){
   display j;
   j = j + n/2; the division is integer division, decimal part neglected
}
}
```

- 1, 0, 2
- 1, 2, 0
- 0, 1, 2
- 0, 2, 1

1st for loop	2nd for loop	Display
i=2: **i=2<=2 → i=3**	**j=0<=n=2**	display **0**; **j=1**
	j=1<=n=2	display **1**; **j=2**

	j=2<=n=2	display **2**; **j=3**
	j=3!<=n=2 → EXIT loop	
i=2: i=3!<=2→ EXIT loop		

☑ **0, 1, 2**

- Big-$\mathcal{O}$
  - Introductory video
  - Know the Big-$\mathcal{O}$ time-complexity for the major sorting algorithms:

Sorting algorithm	Avg runtime complexity	Fast?$\mathcal{O}$
Selection sort	O(N^2)	No
Insertion sort	O(N^2)	No
Shell sort	O(N^1.5)	No
Quicksort	O(N logN)	Yes
Merge sort	O(N logN)	Yes
Heap sort	O(N logN)	Yes
Radix sort	O(N)	Yes

- You will need to determine Big-$\mathcal{O}$ time-complexity from pseudocode. A nutshell summary for the assessment,
  - If the number of steps stays the same no matter how large n is, then it's constant time: $\mathcal{O}(1)$.
  - If you go through a n long list (linearly; say a loop), then you'll take *at most* n steps: $\mathcal{O}(n)$.
  - If you go through a n long list and then do n things each time (e.g.., a loop inside a loop), then you'll take *at most* n^2 steps: $\mathcal{O}(n^2)$.
  - But if you did the first and then the second thing, $n + n^2$, we call this $\mathcal{O}(n^2)$.

This last step is what seems odd to people. But as n gets big, $n + n^2$ will b very close to n^2. So we only worry about the significant part. For the same reason, the notation is always simplified to ignore differences that will be insignificant as n gets large.

- ❏ For 1,200 steps, we write $\mathcal{O}(1)$ *not* $\mathcal{O}(1200)$
- ❏ We write $\mathcal{O}(n)$ *not* $\mathcal{O}(2n)$, $\mathcal{O}(3n)$,...
- ❏ We write $\mathcal{O}(n^2)$ *not* $\mathcal{O}(2n^2)$, $\mathcal{O}(3n^2)$,...
- ❏ We write $\mathcal{O}(n^2)$ *not* $\mathcal{O}(2n^2 + n)$,...
- ❏ We write $\mathcal{O}(\log n)$ *not* $\mathcal{O}(\log_{10} n)$ or $\mathcal{O}(\log_2 n)$,...

From the pre-assessment, it appears that any non-sensical choice, e.g., $\mathcal{O}(-1)$, can be thrown out. See problems 8-12: Big-O practice and Big-O practice solutions.

- You will need to identify the major sorting algorithms from code.
  - **Bubble**: Look for something that swaps so the result can “bubble” to the top.
  - **Bucket**: Look for something that distributes the vales into “buckets” where they are individually sorted.
  - **Merge:** Look for something that continually splits a list in half.
  - **Quicksort:** Look for the keywords “pivot” and/or “split”.