CS2020 Quiz 1 Cheat-sheet

1. Detect Sorting Algorithms

Bubble Sort	The **last** few elements have been sorted, the smallest few elements have been shifted forward. They key idea is that all elements can only *bubble* instead of *jump*. The maximum possible times of shifting forward **equals** the number of element that have been sorted.
Insertion Sort	The **first** few elements have been sorted, the rest are unchanged. Notice that the first element *may not be the smallest* among the array, it may only be smallest in the sorted part of the array.
Selection Sort	The **first** few elements have been sorted. Since the time of swapping operations is minimized, only a few elements have exchanged their positions in pair.
Merge Sort	Number the index of all elements from 1 to n. If the outer-most merge has not been executed, the elements from *the first half* will still **stay** in *the first half*. Otherwise, there may be a strong pattern that they have **swapped** in pair two by two.
Quick Sort	The pivots will **separate** the whole array into a few parts. The array will be unsorted *intra-parts*, however, will be partitioned well *inter-parts*.

2. Time Complexity Analysis

Recurrence Tree	Every recurrence relationship can be written in the form of a tree. What we need to is: 1. Expand the tree big enough; 2. Find the sum of each level; 3. Find how many levels there are. In this way, the only last thing is to calculate the sum of first *n* terms for a certain mathematical series.
Master Theorem	If the recurrence relationship is in the form of $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$, we can let $x = n^{\log_b a}$ and compare it with $f(n)$: 1. If $x > f(n)$, then $\mathrm{T(n)} = \Theta(n^{\log_b a})$; 2. If $x = f(n)$, then $\mathrm{T(n)} = \Theta(f(n) \cdot \log n)$; 3. If $x < f(n)$ and $\mathrm{a} \cdot f\left(\frac{n}{b}\right) < f(n)$, then $\mathrm{T(n)} = \Theta(f(n))$; **Notice:** The relationship of > = < here is comparing the order of functions. For example, $x^3 > x^2$ seems normal; however, $n = n \cdot \log n$ since logarithmic part is neglectable compared to polynomial part.

Some mathematical results to use:

Arithmetic sequence	$1 + 2 + \cdots + n = n \cdot (n+1)/2$
Geometric sequence	$\mathrm{a} + \mathrm{a} \cdot \mathrm{q} + \cdots + \mathrm{a} \cdot q^n = a \cdot \frac{1-q^n}{1-q}$
Geometric series	For $\lvert q \rvert < 1$, geometric series converge to $\frac{a}{1-q}$.
Harmonic series	$1 + \frac{1}{2} + \cdots + \frac{1}{n} = \log n$

Take note of the three different kinds of bounds:

Upper bound: $O(n)$	Tight bound: $\Theta(n)$	Lower bound: $1\Omega(n)$

3. Java Language Specification

(1) Access modifier:

a. Class and interface cannot be declared as private;
b. Private methods or fields can only be accessed within its class;
c. However, private fields of the inner class can be access outside the inner class but inside its outer class;
d. Methods in any interface cannot be private;

e. Static variables and methods belong to class instead of any objects instantiated by the class;
f. Local variables cannot become static;
g. Static methods cannot use non-static fields in its class, however, non-static methods can use static fields in its class;
h. Main method must be static.

(2) Interface:

a. Unless abstract class, the class who implements an interface must implement all methods in that interface (meanwhile, the return value type and signature should be compatible);
b. A class can implement multiply interfaces;
c. An interface can inherit multiply interfaces;
d. All methods in an interface must be abstract, no implementation should be written in the interface;
e. No constructor should appear in an interface;
f. All fields in an interface are public static final, no matter being declared explicitly or implicitly;
g. No static method or code block should be used in an interface;
h. All interfaces, methods in an interface, static fields in an interface are abstract implicitly by default.

(3) Inheritance:

a. Java does allow multi-inheritance for interfaces, but not for classes;
b. Protected fields in the superclass can be accessed by its subclass;
c. The constructor of any subclass must begin with calling one of the constructors of its superclass.

4. Algorithm Design

In fact, the strategy for most problem-solving questions in Quiz 1 is "Divide-and-Conquer". From this point, three primitive and classic algorithms are: binary search (peak finding, aggressive cow, and Herbert log), merge sort (counting inversion, merging multiply sorted arrays) and quick sort (quick select, shuffle, kSUM).

In addition, we have learnt the following data structure: array (simple array, array list), linked list (linked list, queue, stack, skip list), and tree (binary tree).

Please remember the following examples:

1) Counting Inversion

Inversion on the left half + inversion on the right half + inversion during merge. The time complexity is O(n * logn).

2) 2SUM Problem

First sort the whole array, it takes the time of O(n * logn). After that, use two pointers to traverse to the middle part, like what we do for the in-place partition for quick sort. The overall time complexity is O(n * logn). If hash table is allowed to use, it can be improved to O(n).

3) 3SUM Problem

First sort the whole array. Then, for each element A[i] in the array, use 2SUM problem to find two elements (from A[i] to A[length - 1]) whose sum is (SUM – A[i]). The overall time complexity is $O(n^2)$.

In similar approach, we may find the time complexity of kSUM problem is $O(n^{k-1})$.

4) 4SUM Problem

We first find all pairs in the array (there are n^2 pairs in total) and store the sum of each pair in an array of length n^2. Then, use 2SUM problem to find two element in the new array whose sum is the final sum we want to find out. The overall time complexity is $O(n^2)$.

In similar approach, we may find the time complexity of kSUM problem is $O(n^{k/2})$ when k is an even number larger than 2.

5) Lecture Hall

Sort them by the start time first. Therefore, we know A[i + 1] >= A[i]. We just need to find the longest continuous segment that A[m] = A[m + 1] = … = A[n]. The answer is n – m + 1.

6) Merge Multiply Sorted Arrays

Merge two of all the arrays each time. The time complexity is O(nk * logk).

7) Herbert Log

The first part use a "jumping" binary search. The difference is normal binary search only recurves on one half, but Herbert log recurves on both halves and skip the whole segment when the start and end is the same. The second part is aggressive cow.

Special Notice: Sometimes, only ***linear search*** can be used. DO NOT always attempt to use binary search.