**The Ultimate Data Structures Study Guide: Exam 1 Edition**

**Contents:**

* **Native Types** 
  + In Java the native types: Boolean, Byte, Char, Short, Int, Long, Float and Double
* **Reference Types**
  + A reference type is a variable that holds a memory address, an example is String aRef = “my word”
* **Abstraction**
  + Abstraction is a mold of a real or synthetic object.
  + In Java these can be classes, objects or methods.
* **Encapsulation**
  + Encapsulation is hiding detail of what is inside
  + Examples of it in Java: method, class, access, modifiers. Public, private, protected, and package protected.
* **Inheritance** 
  + Inheritance is a mechanism for a code to use between at least two classes in which the subclass (child) inherits behaviors from the parent class (super) by the extend of a key word.
* **Polymorphism** 
  + Polymorphism is when a method is called but the actual implementation/call to execute is specified of run time
  + Polymorphisms have multiple implementation of a class via abstract class or via interface
* **Array**
  + Arrays are collections of objects and variables that are stored continuously in memory.
  + Arrays are finite and their size cannot be changed
  + In Java, all arrays independently of their type ae objects
  + Example: INT a[] = new int []
  + String newArray[] = {“mage”, “knight”, “rogue”, “priest”};
* **Comp Complexity** 
  + Comp complexity is to measure the efficiency of an algorithm based on input size
  + Types:
    - Space (memory)
    - Time
      * Evaluating time complexity: n towards infinity, input size is n
      * Asymptote behavior: T(n) -> count number of instruction, O(.) find the fastest growing rate in the T(n)
      * To evaluate the T(n) count the number of fundamental operations
        + A statement that takes O(1)
      * For loops: number of iterations times the cost of a single loop.
      * Find the behavior of the T(n) as n -> infinity

T(n) = n^2 + 4n + 10

T(1) = 1^2 + 4(1) + 10 = 15

T(10) = 10^2 + 4(10) + 10 = 150

T(100) = 100^2 + 4(100) + 10 = 1000,000,000

T(n) behaves similarly to O(n^2)

* + - * For example: for (int c = o; c(n, c++)

{

f(c) -> f(c) = O(n^2)

}

T(n) = n \* n^2 = n^3

O(n^3)

* + - Power
* **Recursion**
  + Recursion is a function that calls itself.
  + Needed for recursion:
    - At least 1 base case
    - At least 1 recursive call. Always has the same method with a smaller version of the problem due to memory constraints
  + Recursive call is a call to the same function to solve a smaller version of the problem
  + There cannot be infinite recursions as you’d need infinite memory
  + When we use recursion in programming the recursion steps must always be smaller versions of the same problem
  + Base case ->[if n = 0 or n = 1]

f(n) > 1 = f(n-1) + f(n-2)] <- recursive step

* + Cost of operation = number of iterations \* [cost of a single iteration] of a loop
  + Outmost loop = n \* [cost of the complete inner loop]
  + Cost of complete inner loop = number of iterations
* **Searching and Sorting**
  + Searching:
    - Sequential (Linear) O(n)
      * Linear is comparing one by one starting from the beginning
    - Binary Search O(log(n))
      * Binary search works on an array as long as it consists of numbers greater than the last.
      * Array must be in order.
  + Sorting:
    - Sequential Sorting
      * Bubble sort
        + At every iteration if elements are out of order consecutively, we switch their positions
      * Insertion Sort
        + Portioning the array into two sections: a sorted section and unsorted. At every iteration we select the first element of the unsorted section and insert it into the sorted one in order
      * Selection sort
        + At every iteration the algorithm finds the greatest element and puts it at the end of an array.
    - Logarithmic Sort:
      * Quick sort
        + Sorts by choosing at each iteration an element called the pivot and finding the correct place of the pivot in the sorted away (the partition method) then apply quick sort to the left of the pivot and to the right of the pivot. After partition method is called, all elements smaller than the pivot are left to it (the pivot), and all greater to the right.
      * Merge sort
        + Divide and conquest
        + Divide array until you have arrays of size 1, and then merge in order
    - External sort:
      * Radix sort
        + Makes use of buckets to place elements to be sorted according to the order between the digits. After placing them in the buckets, elements are placed together again.
* **Sorting using Java API**
  + Arrays.sort with comparator
  + Arrays.sort() in Java:
    - public static void **sort**(int[] arr, int from\_Index, int to\_Index)

**arr** - the array to be sorted

**from\_Index** - the index of the first element, inclusive, to be sorted

**to\_Index** - the index of the last element, exclusive, to be sorted

This method doesn't return any value.

* + - Descending order example - Arrays.sort(arr, Collections.reverseOrder())
    - Ascending order example- Arrays.sort(arr);

 System.out.printf("Modified arr[] : \n%s\n\n",

                         Arrays.toString(arr));

* + - Example of a basic sort:
      * import java.util.Arrays;
      * Int[] arr = { 7, 3, 5, 1};
      * Arrays.sort(arr);
      * System.out.print(Arrays.toString(arr));