CS2020 Coding Quiz Cheat-sheet

1. Problem-solving Strategy

Divide and conquer	*Pre-condition*: 1) A large problem can be divided into several small problems; 2) Several small problems can be solved separately and all the solutions can be combined together. *Features*: 1) A smaller problem is easier to solve than a larger one; 2) All sub-problems are independent; 3) Use recursive programming much.
Greedy algorithm	*Pre-condition*: A local optimal solution is the overall optimal solution (known as optimal sub-structure). *Features*: 1) Have the optimal choice on each step; 2) Traverse over the collection iteratively.
Dynamic programming	*Pre-condition*: Previous sub-problems can provide useful information for later ones. *Features*: 1) Some of the sub-problems are dependent; 2) Useful previous sub-problems should be stored, but this may result in very bad space complexity (equal to memorization algorithm in the worst scenario); 3) Rely on recursive programming.
Randomized algorithm	*Pre-condition*: Be able to determine good enough cases and find them. *Feature*: Effectively avoid worst cases.
Brute-force	*Pre-condition*: 1) Sufficient hardware resources; 2) Be able to go through all cases. *Feature*: Easy to think and implement.
Mathematical expression	*Pre-condition*: Obtain a close-form formula for the problem however large the scale is. *Feature*: The time and space complexity is O(1).

2. Classical Algorithm Design

1) *Divide and conquer*: binary search, peak finding(1D – linear search, binary search; 2D – linear search on each column, linear search on binary column, increasing path on binary column, border & cross), aggressive cow (exponentially increasing range), Herbert log (skip unnecessary segments), quick sort, quick select, median list (compare two medians and select appropriate halves), shuffle, kSUM (first sort then find pairing), counting inversions (#left + #right + #merge), multiple merging (only merge two each time).

2) *Greedy algorithm*: lecture hall (create and sort a collection of starting and ending moments, maintain a queue of lecture halls being used, enqueue at starting points & dequeue at ending points, keep record of the maximum size of the queue so far), single-sell profit (keep record of the minimum value and maximum profit so far), activity scheduling (sort by ending time, traverse through the array, add one if the last activity has finished).

3. Useful Java System APIs

1) `java.util.Arrays` package

`boolean Arrays.equals(T[] arr1, U[] arr2)` returns true if two arrays contain the same elements in the same order.

`void Arrays.sort(T[] arr)` sorts an array of comparable items in its ascending numerical order.

`T[] Arrays.copyOf(T[] origin, int length)` copies and returns elements in `origin` from `0` to `length-1` to a new array.

2) `java.lang.Integer` class

`String Integer.toString(int x, int radix)` returns a string representing the integer `x` in base `radix`.

`int Integer.parseInt(String s, int radix)` returns the integer that string `s` represents and converts from base `radix` to decimal.

3) `java.lang.Character` class

`boolean Character.isAlphabetic(char c)` returns whether a certain character represents a letter (by ASCII code).

`boolean Character.isDigit(char c)` returns whether a certain character represents a digit 0~9 (by ASCII code).

Notice the following *ASCII code point* for characters: new line – 10, white space – 32, 0 - 48, A – 65, a – 97.

4) Miscellaneous

`boolean m.equals(Object n)` returns true if and only if `m` and `n` points to the same object.

`boolean str1.equals(str2)` returns true if two strings consists of the same characters in the same order.

`char str.charAt(int x)` returns the character of index `x` in the string `str`.

`boolean arrList.contains(x)` returns true if an array list contains at least one element `y` that `y.equals(x)` is true.

`void Collections.sort(List<T> list)` sorts a list of comparable items in its ascending order.

`void System.arrayCopy(T[] src, int srcPos, T[] dest, int destPos, int length)` copies from the source array to destination array at a certain position for a determined length.

4. Important code implementation

1) Binary search

```
public int searchFirstIterative(int[] arr, int target) {
    int start = 0, end = arr.length - 1, mid = -1;
    while (start < end) {
        mid = (start + end) / 2;
        if (arr[mid] < target) {
            start = mid + 1;
        } else {
            end = mid;
        }
    }
    if (arr[start] == target) {
        return start;
    } else {
        return -1;
    }
}
```

2) Count inversions

```
private int count(ArrayList<T> arr, int start, int end) {
    if (start == end) {
        return 0;
    } else {
        int mid = (start + end) / 2;
        return count(arr, start, mid) + count(arr, mid + 1, end) + merge(arr, start, mid, end);
    }
}

private int merge(ArrayList<T> arr, int start, int mid, int end) {
    int indexA = start;
    int indexB = mid + 1;
    int count = 0;
    int size = end - start + 1;
    ArrayList<T> temp = new ArrayList<T>();

    for (int i = 0; i < size; i++) {
        if (indexA == mid + 1) {
            temp.add(arr.get(indexB++));
        } else if (indexB == end + 1) {
            temp.add(arr.get(indexA++));
        } else if (arr.get(indexA).compareTo(arr.get(indexB)) > 0) {
            count += (mid - indexA + 1);
            temp.add(arr.get(indexB++));
        } else {
            temp.add(arr.get(indexA++));
        }
    }
    for (int i = 0; i < size; i++) {
        arr.set(i + start, temp.get(i));
    }
    return count;
}
```

All the best!
