HW3

01. ① Apt-get: used to make it easier with installation utilities and package manager

eg: sudo apt-get install git

② Grep: searches a file for a particular pattern of characters, and displays all lines that contain that pattern

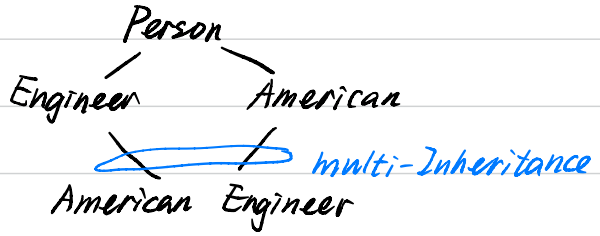
eg: grep "day" re.txt | wc -l

③ '|' in Unix: pipe output of first program to second

eg: grep "day" re.txt | wc -l

④ multi-Inheritance: A class can inherit from multiple classes

eg:



⑤ Access Specifiers: Public means it can be accessed by all classes including same class, derived class and outside class.

Private means it can only be accessed by the same class.

⑥ The public interface represents the abstraction component of the design while typical encapsulation makes class data members private.

⑦ Structured programming are often hard to reuse, extend or maintain while OOP focus on creating objects rather than procedures / functions. Objects have both data and procedures that manipulate that data. OOP is preferred when we want to model a complex platform because it saves development time by reusing code.

⑧ Overloading occurs when two or more methods in one class have the same method name but different parameters;

Overriding means having two methods with the same method name and parameters. One of the methods is in the parent class and the other is in child class.

Q2. When size n is small. Take bubble sort and merge sort as an example. The time complexity of bubble sort is $O(n^2)$ while the time complexity of merge sort is $O(n\log n)$. Usually $O(n^2)$ takes longer time than $O(n\log n)$. However, when n is small, n^2 or $n\log n$ may not be the dominanting one, i.e. constant parameters can not be neglected under this circumstance since constants will affect the total time.

Q3. No, exponential time becomes ugly when size n increases. For example, assuming $k=3$, when n is longer than 100,000, it could take years for the program to run

Q4.
```
CountFrequency (arr[], n) {
    for ( int i=0 ; i< arr.length() ; i++ ) {
        if ( arr[i] == n) {
            while ( a[i] ==n && i< arr.length() ) {
                i++;
                count++;
            }
            return count;
        }
    }
    return 0;
}
```
This is linear search with time complexity $O(n)$.

Q5.
```
Map<Integer, Integer> count = new HashMap<>();
for ( int i : arr[] ) {
    count.put (i, count.getOrDefault ( i, 0) +1);
}
int max = 0;
```

```
Queue<Integer> queue = new LinkedList<>();
for ( int key : count.keySet()) {
    if (count.get(key) = max)   queue.push(key);
    else if (count.get(key) > max) {
        queue.clear();
        queue.push(key);
        max = count.get(key);
    }
}
print queue;
```

There are two for loop, each has O(n) time complexity, so the total time complexity is 2O(n) = O(n)

Q6. Code is in seperate file.

Result:

```
usc-securewireless-student-new123:HW3 insane$ g++ -o sameside HW3-Q6.cpp
usc-securewireless-student-new123:HW3 insane$ ./sameside
Please enter three parameter of line ax+by-c=0 (e.g: 1 2 3): 1 2 3
Please enter the first integer points (e.g: 3 4): 6 7
Please enter the second integer points (e.g: 3 4): 1 9
(6, 7) and (1, 9) lie on the same side of the given line.
usc-securewireless-student-new123:HW3 insane$ ./sameside
Please enter three parameter of line ax+by-c=0 (e.g: 1 2 3): 1 2 3
Please enter the first integer points (e.g: 3 4): 6 -7
Please enter the second integer points (e.g: 3 4): 1 9
(6, -7) and (1, 9) lie on the different side of the given line.
usc-securewireless-student-new123:HW3 insane$
```

Q7. Code is in seperate file.

Partial result:

```
860: 68
861: 11
862: 83
863: 24
864: 64
865: 25
866: 95
867: 65
868: 19
869: 46
870: 47
871: 49
872: 10
873: 30
874: 2
875: 74
876: 68
877: 69
878: 26
879: 66
880: 39
881: 48
882: 78
883: 21
884: 45
885: 77
886: 89
887: 89
Maximum Value: 100
```

Q8.

```
int i = 0, j = 1;
while ( j < arr.length()) {
    if ( arr[j] - arr[i] == 2)  return arr[i]+1;
}
return -1;   //-1 stands that no missing integer.
```

Extra Credit:

```
SameSidePlane ( x1, y1, z1, x2, y2, z2 ):
    if ( x1 · x2 > 0 ) {
        if( y1 · y2 > 0 ) {
            if (z1 > 0 && z2 < 0)  return false;
            else if ( z1 < 0 && z2 > 0)  return false;
            else return true;
        }
        return false;
    }
    return false;
```