Problem Set 1
Chieh Lee
You Zhang
CS4800

Part I

1. A is Θ(B)
2. A is O(B)
3. A is Ω(B)
4. A is O(B)
5. A is O(B)

Part II

1.

Use insertion sort to sort the students by GPA **O(n^2)**
Iterate down the sorted list to pair students **O(n)**
Iterate down the list of pairs to calculate the average GPA for each pair of students
O(n)

O(n^2)

2.

The first guess attempt will lead you to the center of the N x N square. So it's a constant
After that, each guess attempt for y coordinate (assuming adversary give up/down until that coordinate is correct) will half the range by changing y_{high} or y_{low}, therefore it's O(Log(n))
There it takes **O(Log(n) + Log(n))** to complete guess
And x coordinate will perform the same, therefore another O(log(n))
In addition, it's doesn't matter if the adversary shuffles the left/right and up/down since the total guess attempts would be the same.

3.

```
For L = N to 1 // L = length; start with long strings, work down O(n)
    For i = 0 to N-L // start of substring in A O(n)
        For j = 0 to N-L // start of substring in B O(n)
```

For k = 0 to L-1 // index into substring **O(n)**
If A[i+k] != B[j+k] break to next j // mismatch of character
Return substrings of length L starting at A[i] and B[j]
Return “no match”

We have go through N length with each substrings for A and B. we also have to access the index when we compare substrings, so we have to go through N basically 4 times.

$O(n^4)$

4.
This function requires accessing all the possible subset of list of integer L. In order to examine all possible reorderings of the data it need **Θ(N!)**

5.
$O(N^2N!)$
The possible matching in this problem is N! . To verify each matching, we need O(N2) time. So

the total running time is **O(N2N!)**.

part III
We create a method that read list from the beginning with two variable value indicate the key where is the end the 1s and 2s. So each time when we read a number from the list, extract the number and add to the sorted list.
If number = 1, add at the beginning, and push the endOfOneKey 1 index further.
If number = 2, add at the next index of endOfOneKey 1 index, and push the endOfOneKey 2 index further.
If number = 3, add at the end of the list. (or add at the next index of endOfOneKey 2 index)

Part IV
Results:
LinearFib.java

40 → 1	42 → 1	44 → 1	46 → 1	48 → 1

ExpFib.java

40 → 512	42 → 1372	44 → 3564	46 → 9360	48 → 24591

InsertionSort.java

1000 → 3	10000 → 17	100000 → 1137	1000000 → 123131

QuickSort.java

1000 → 0	10000 → 4	100000 → 62	1000000 → 142

LoL.java

However generate the list of list take a lot of time, N = 5000 is too long.

100 → 68	1000 → 109	5000 → ???

My2DArray.java

100 → 67	1000 → 121	5000 → 164