HW1: Algorithm Fundamentals. There are 4 parts, total 10 questions, each question is 10 points.

Part 1: Estimation

1. Suppose that you observe the following running times for a program with an input of size N.

N	time
5,000	0.2 seconds
10,000	1.2 seconds
20,000	3.9 seconds
40,000	16.0 seconds
80,000	63.9 seconds
160,000	255.6 seconds
800,000	

doubling

6x increase
3.25x increase
4.1x increase
4x increase
4x increase

Estimate the running time of the program (in seconds) on an input of size N =200,000. Show your work.

1. Seems to be an N^2 relationship in time increase (quadratic)
2. 4x 50,000 = 200,000 (rough estimate)
3. To estimate 50,000 runtime take avg of runtimes for 40K & 80K
   - N: 50,000; Time: $\frac{16 + 63.9}{2} = 39.95$ seconds for N = 50,000
4. Now multiply by 16 (4^2) $39.95 \times 16 = 639.2$ seconds

2. You observe the following running times for a program with an input of size N.

N	time
1,000	0.1 seconds
2,000	0.3 seconds
4,000	2.5 seconds
8,000	19.8 seconds
16,000	160.1 seconds

doubling

3 x increase
8x increase
7.92 x increase
8x increase

Estimate the running time of the program (in seconds) on an input of size N = 80, 000.

1. Time is increasing faster than n^2, going to assume n^3
2. 16,000 x 5 = 80,000
3. assuming ~ n^3 growth, can expect time to increase by $5^3 = 125$
4. $125 \times 160.1 = 20{,}012.5$ seconds

Part 2: Please determine the **running time function**, T(n), for the following codes.

1.

- loop runs as long as $i <= n$
- i doubles each loop
- logarithmic pattern

```
int i=1;
while(i<=n){
    count++;
    i = 2*i;
}
```

$T(n) = O(\log N)$

2.

```
for(int i=0; i<10000;i++){
    for(int j=0; j<i; j++){
        for(int k=0;k<j;k++){
            a[i] = + a[j] + a[k];
        }
    }
}
```

- outer loop runs for fixed amount which means it is constant

$T(n) = O(1)$

3.

```
for(int i=0;i<=2n;i++){
    for(int j=0;j<=3n;j++){
        count++;
    }
}
```

- nested loops dependent on n

$T(n) = O(n^2)$

4.

```
for (int i=0; i<n; i++){
     for(int j=0; j<i; j++){
          a[i] = i + j;
     }
}
```

$T(N) = O(n^2)$

5.

```
for(int i=1;i<=n;i++){
    for(int j=1;j<2*i;j++){
         k = j;
         while(k>=0){
             k = k – 1;
         }
    }
}
```

$T(n) = O(n^2)$

inner loop runs constant "j" times

for every $j < 2 \times i$

Part3: Tilde Notation

1. Using Tilde notation in terms of the parameter *n*, how much time does the following method take?

```
int arraymax(int[] a, int n){
     int max = a[0];
     for(int i=1;i<n-1;i++)
            if(a[i] >max)
                 max = a[i];
      }
      return max;
}
```

every element of except the 1st is checked only one time

$\tilde{O}(n)$

2. Using Tilde notation in terms of the parameter *n*, how much time does the following method take?

```
i=1;
sum = 0;
while (i <= n) {
    j=1;
    while (j <= n) {
        sum = sum + i;
        j = j + 1;
    }
    i = i +1;
}
```

- 2 nested while loops
- inner loop runs for every value of i up to n

← executed 2 times in total so it is n×n or n^2

$\tilde{O}(n^2)$

Part 4: Algorithm Analysis

The code below operates on bacterial genomes of approximately 1 megabyte in size.

```
int N = Integer.parseInt(args[0]);
String[] genomes = new String[N];
for (int i = 0; i < N; i++) {
      In gfile = new In("genomeFile" + i + ".txt");
      genomes[i] = gfile.readString();
}

for (int i = 1; i < N; i++) {
      for (int j = i; j > 0; j--) {
            if (genomes[j-1].length() > genomes[j].length())
                  exch(genomes, j-1, j);
            else break;
      }
}
```

a. What is the mathematical order of growth of the worst-case running time as a function of N? if in reverse order worst case would be $O(n^2)$

b. A table of runtimes for the program above is given below. Approximate the empirical run time in tilde notation as a function of N.

N	Time (s)
1	0.15
2	0.14
4	0.19
8	0.41
16	0.85
32	1.66
64	3.38

$\tilde{O}(n^2)$

c. Explain any discrepancy between your answers to (a) and (b).

discrepancies could come from factors such as Data size, System differences & overhead.