Part-1 Write down complexity in terms of Big-O for methodA, methodB and methodC.

Note: There's no need to run the code. Run by your eyes and analyze by your brain.

You should write down the complexity in terms of Big-O in a word or PDF document.

Explain why you conclude your answer.

methodA

```
public static void methodA(int n) {
      int i=0;
      int j=0;
         int total = 0;
         while(i<n) {
             while (j < n) {
                 total++;
                 j++;;
             }
             i++;
         }
     }
```

Method A's complexity is $O(N^2)$. This is because, after the first initialization of i (which occurs n times), j has increased from 0 to n. As a result, the total number of increments is doubled (n * n), denoted as $O(N^2)$. In other words, for each iteration of the outer loop, the inner loop is run n times, for a total of n * n operations.

Interation	i	j	total
1	0	0	1
2	0	1	2
3	0	2	3
4	0	3	3
5	1	3	3
6	2	3	3
7	3	3	3

methodB

```
    public static void methodB(int n) {
      int i=0;
      int j=0;
      int k=0;
         int total = 0;
         while(i<n) {
             while (j < n) {
                    while (k < n) {
                 total++;
                 k++;;
             }
             k=0;
             j++;
         }
             j=0;
             i++;
    }
```

MethodB in this example has a complexity of $O(N^3)$. The reason for this is that for each iteration of the outer loop, the middle loop runs n times, and the inner loop runs n times, for a total of n * n * n operations. The total variable is incremented n^3 times, resulting in an $O(N^3)$ time complexity for this approach.

Interaction	i	j	k	total
1	0	0	0	1
2	0	0	1	2
3	0	0	2	2
4	0	1	0	3
5	0	1	1	4
6	0	1	2	4
7	0	2	0	4
8	1	0	0	5
9	1	0	1	6
10	1	0	2	6
11	1	1	0	7
12	1	1	1	8
13	1	1	2	8
14	1	2	0	8
15	2	0	0	8

methodC

```
    public static void methodC(int n) {
      int i=0;
      int j=0;
      int k=0;
         int total = 0;
         j=n;
         while(k<n) {
             while ((j=j/2) > 0) {
                    for (i=0;i<100*n;i++) {
                 total++;

                 }
                    k++;
             }
         }
    }
}
```

The level of complexity for the methodC in this instance is $O(N^2 log(n))$

The reason is the outer while loop runs n times because k is incremented once per iteration of the inner while loop and continues until k<n.

The inner while loop executes log(n) times since j becomes half in each iteration (j=j/2) and continues until j>0.

The innermost for loop is executed 100n times because the i ranges from 0 to 100n.

So the overall time complexity is the product of the number of times each loop is executed, which is n * log(n) * 100n. Thus, the temporal complexity of this technique is O(n2*log(n).

Part-2

The Selection Sort algorithm has a worst-case time complexity of O(n^2), where n represents the array's number of items. This is because, for each element in the array, the algorithm searches the rest of the array for the smallest element. This produces quadratic time complexity.