COMP9101 Ass02
Name: Zanning Wang
zID: z5224151

Q3:

In order to try to kill all monsters, we can divide the monsters into two category, the first one is that $a_i <= g_i$, which means by killing those monster, we will gain the strength after killing those monsters, the other category is those $a_i > g_i$. for $a_i <= g_i$. In order to ensure we can kill the next monster, we should accumulate the strength as most as we can, therefore we should start with the smallest a_i, for $a_i > g_i$, after killing those monster, we will definitely lose some strength, we may not be able to kill the next monsters if the strength we recover is too low, therefore we should start with the largest g_i, the sudo-code may like below:

```
1  //array1(ai<=gi),array2(ai>gi)
2  //array1 = [[2,4],[3,6],[4,5]] (the first one is ai, the second is gi)
3
4  [array1, array2] = divide_two(N);
5  sort_1(array1);
6  sort_2(array2);
7  let start = S
8  for (let i=0; i<array1.length;i++) {
9    if (start >= array1[i][0]) {
10     start = start - array1[i][0] + array1[i][1]
11   } else {
12     return "no such ordering"
13   }
14 }
15
16 for (let i=0; i<array1.length; i++) {
17   if (start >= array2[i][0]) {
18     start = start - array2[i][0] + array2[i][1]
19   } else {
20     return "no such ordering"
21   }
22 }
23
24 return "there is such an ordering !"
```

For line 4, we divide all monster into two category, the first category is $a_i <= g_i$, the second is $a_i > g_i$, the output array1 may like that: array1 = [[2,4],[3,6],[4,5]] (the first one is ai, the second is gi), which the time complexity is the number of monster (O(n))

For line5-6, the function sort_1 sort the first value, which is a_i in ascending order, the function sort_2 sort the second value, which is g_i in descending order, which the time complexity is O(NlogN)

From line8-22, we check if the current strength S can beat the next monster with our algorithm, if not return “no such ordering”,the time complexity for those two loop is O(2N)

Therefore, the total time complexity is O(NlogN).