Ochirgarid Chinzorig, Assignment 4

1. (A). The expected number of tosses required to get a 2 is **10**. By the theorem of geometric distribution "The expected number of trials to obtain a success is 1/p" here p=1/10 because we have a fair die with 10 sides.
   (B). Again, by the theorem the expected number of tosses to get a total of three 2's is **3/p=30**.
2. As we shown in the lecture (Lecture-4, page 32) the worst case of the Comparison-based algorithm for the 4 distinct integer is 5. Since there is not any Comparison-based algorithm with only 4 comparisons. The quick sort will use only 5 comparisons to sort the array with only 4 distinct integers in the worst case.
3. (A). Yes, it will work. Since he arranges the array in a random order the sorted order of the array is in included in the random order.
   (B). The best case for the GoofySort will be when the given array is already sorted.
   (C). The best case is O(N), since he needs to iterate through the array at least once for every element.
   (D). In the worst-case we could get infinite. The algorithm could be never ending because of the random is not guaranteed to hit the target order.
   (E). The average-case running time will be **O(N*N!).** Because there is a total of N! arrangements for an array with N elements. And the Expected number of tries to get sorted order is 1/p, and p=1/n! . On each step we use check the array is sorted by O(N) and randomly arrange the array by O(N). Hence on each step we use O(N) for average of N! steps that results the average-case running time will be O(N*N!).
   (F). No, the algorithm is not inversion-bound, nor comparison-bound.
4. A = [5, 1, 4, 3, 6, 2, 7, 1, 3], 3n/4 = 6.75

	🟩 5	🟥 1	🟩 4	🟩 3	🟥 6	🟩 2	🟥 7	🟥 1	🟩 3
L	6	0	5	3	7	2	8	0	3
E	1	2	1	2	1	1	1	2	2
R	2	7	3	4	1	6	0	7	4

   (A) Above, I created a table to mark the good pivots with green color and bad pivots with red color.
   (B) The number of good pivot values is 5 > 7 /2.

5. The sideways algorithm pseudo-code

```
Algorithm SidewaySort(arr)
Input: array of integers arr
Output: Sideway sorted array
    sort(arr)                //using any of the fastest sort
    result <- new integer array of size arr.length
    for i<-0 to i < arr.length do
        result[i] = arr[i]
        result[i + 1] = arr[arr.length - (i / 2) - 1]
        {i = i + 2}
    return result
```

   (A) The algorithm has 2 parts first part is the sorting phase, second part is O(N) assigning the values into the result array phase. Depending on the sorting algorithm our asymptotic running time will be:

(i) The sorting algorithm is Inversion-bound algorithm the running rime will be $O(N^2)+O(N) = O(N^2)$

(ii) The sorting algorithm is comparison-based algorithm the running rime will be $O(N \log N)+O(N) = O(N \log N)$

(iii) The sorting algorithm is bucket algorithm the running rime will be $O(M+N)+O(N) = O(M+N)$

(B) The algorithm I devised solely depends on the running time of the sorting algorithm. Since the sorting could not be improved depending on the type (Comparison-based, inversion-bound, bucket-sort) they used, the algorithm I devised is the fastest running time of the sideway sorting.