Question A

A) There is no need to implement Strassen's algorithm as I managed to pass it in my first time using the O(n^3) algorithm (which is slower than Strassen's algorithm)
B) I tried using Strassen's algorithm for the first few times and I didn't manage to get it. Then, I went to the discord forum and found out that many people were implementing Freivald's algorithm. I also read the PA_2 PDF and visited the wiki page to learn more about Freivald's algorithm and therefore I went ahead to use Freivald's algorithm to complete A2. I think that we won't be able to pass the test cases under the time limit given when using Strassen's algorithm as it is much slower than Freivald's algorithm.
C) I didn't get to try Duan, Wu, Zhou as I just went ahead to use Freivald's algorithm which manage to pass the test cases under the time limit given. Freivald's algorithm is a randomised algorithm while I assume Duan, Wu, Zhou algorithm to be deterministic. And I managed to pass with Freivald's algorithm after submitting multiple attempts.

A) Freivald's algorithm is a Monte Carlo algorithm.
B) For my code, Freivald's algorithm runs faster when A X B not equal to C as I run k iterations of Freivald's algorithm and I exit the loop immediately when I encounter a ABr[I] != Cr[I] (this is most likely/usually smaller than k). For the case where A X B = C, my code will run finish k iterations therefore, it tends to be a little slower for positive cases as there are more operations to be completed. Therefore for the test A X B = C, my code runs O(kN^2) whereas for the negative case, my code runs O(aN^2) where a < k.
   The test case for A X B = C is always correct. We would always expect to get AC. For the test case A X B != C, it will sometimes return AC even though it is not true.
C) At first, I chose 5 as my K but I realised that might be too small for it too be correct most of the time. Therefore, I choose 6 and 7 subsequently and I passed the test case when I used 7.
D) In theoretical expectation, we expect the error rate to be of 1/(2^7) which is 0.0078125. We can just use negative binomial distribution to calculate the number of test cases before we get our first success. The expected number of trials needed to see r successes is r/p. Therefore, win our case we want to see 1 success scenario, there is 1/ ((1- (0.5^7)))^77 which 77 is the upper bound of the total test case given. The answer we will get is 1.82 which we will round up to get 2. Therefore, the 2nd time we run the test case, we will most probably be successful.

Question B

A) I don't think we will be able to use any comparison-based sorts at all as we needed to solve the problem in linear time, therefore only randomised quick select seem wise. We are given a time-limit of 2.5s. The rough number of operations for all comparison-based sorts that runs in O(nlogn). The upper limit of TC given is 30 and the upper limit of N given is 10^6. We were told to ensure total number of operations is not more than 2.8 x 10^8 basic operations (modulo is kinda expensive, minimize this other than when necessary). Given the upper limit of TC and upper limit of N, we are definitely not able to do this under 2.8 x 10^8 basic operations.
B) Only the middle element (which is the median of U) will be in the correct position. Therefore, we will always be able to get the median of the array. The other indexes are most likely not in sorted order as it don't matter if they are sorted or not as they do not contribute to us finding the median. However, one guarantee that we can have is the the left hand side of the median will always be smaller than the median and the right hand side of the median will always be bigger than the median.
C) No, there are more operations involved in radix sort as we have to actually make sure all the elements are in the correct position in the array. However, quick select is able to look pass that which allow us to use lesser operations to find the median. For PA1-B1/counting sort it will definitely not work at its runtime is O(n+k) since the k given is very large, it would definitely take a lot of operations to create and iterate through these large arrays.
D) I used the expected linear time QuickSelect that was discussed in tutorial. Quickselect has an expected average run time of O(n).

Efg-2
I used the nth_element() function provided in the algorithm library and from there I partition by the middle and find the newest middle element which is the median.
1. Yes, it passed the time limit of B.
2. It is a Las Vegas algorithm.

3. We can just start over and run the test case again as Las Vegas algorithm is a probabilistic algorithm and we might be able to pass the next time round provided that we implemented it correctly and efficiently.