Problem 1: Sorting

(a) Insertion Sort

[314, 512, 004, 999, 023, 042, 613, 109, 001, 123, 666]
[004, 314, 512, 999, 023, 042, 613, 109, 001, 123, 666]
[004, 023, 314, 512, 999, 042, 613, 109, 001, 123, 666]
[004, 023, 042, 314, 512, 999, 613, 109, 001, 123, 666]
[004, 023, 042, 314, 512, 613, 999, 109, 001, 123, 666]
[004, 023, 042, 109, 314, 512, 613, 999, 001, 123, 666]
[001, 004, 023, 042, 109, 314, 512, 613, 999, 123, 666]
[001, 004, 023, 042, 109, 123, 314, 512, 613, 999, 666]
[001, 004, 023, 042, 109, 123, 314, 512, 613, 666, 999]

(b) Quick Sorting

[314, 512, 004, 999, 023, 042, 613, 109, 001, 123, 666]
Left = 0, List[left] = 314; Right = 10, List[right] = 666;
List[Center] = 042; Swap List[Left], List[Center];
Pivot 314 [004 023 042 109 001 123][314] [512 999 613 666]
Pivot 042 [004 023 001] [042] [109 123] [314] [512 999 613 666]
Pivot 004 [001] [004] [023] [042] [109 123] [314] [512 999 613 666]
Pivot 109 [001] [004] [023] [042] [109] [123] [314] [512 999 613 666]
Pivot 666 [001] [004] [023] [042] [109] [123] [314] [512 613] [666] [999]
Pivot 512 [001] [004] [023] [042] [109] [123] [314] [512] [613] [666] [999]
[001 004 023 042 109 123 314 512 613 666 999]

(c) 0
1 001
2 512 042
3 023 613 123
4 314 004
5
6 666
7
8
9 999 109

001 512 042 023 613 123 314 004 666 999 109

0 001 004 109
1 512 613 314
2 023 123
3
4 042

5
6 666
7
8
9 999

001 004 109 512 613 314 023 123 042 666 999

0 001 004 023 042
1 109 123
2
3 314
4
5 512
6 613 666
7
8
9 999

Result: 001 004 023 042 109 123 314 512 613 666 999

Problem 2: Sorting Stability

(a) If we add index(location) for each number in the key it will be helpful to make sorting become stable. (For example, add 0 for first number in their key, add 1 for second number in their key …) After we finish sorting, just look whether is there are any number are same, but the order of those number are different with the index information order for those number which is append in the key. For example, we have 060 which index is 4 in their key and 060 which index is 5 in their key. But the 060 which have index is 5 in their key but is before 060 which have index is 4 in their key. So we could see it is unstable, we just need swap them to make it become stable.

(b) For example, we have number which is 4 4 3 2 1.
We using 3 as pivot. Firstly, i is point to the first 4, which is bigger than pivot. We have j is point to the 1 which is small than pivot. We need swap first 4 and 1. Now the list looks like this: 1 4 3 2 4; Now the i is point to the Second 4 which is bigger than pivot, j is point to the 2 which is smaller than pivot. So we need swap them. Now the list looks like: 1 2 3 4 4. But you can see the second 4 now is before first 4, which proved that the quick sort is not stable.

Problem 3: Parallel MergeSort

For our situation, it will be looks like $T(N) = T\left(\frac{N}{2}\right) + N$

And T (1) = 1;

Recursively: $T(N/2) = T\left(\frac{N}{4}\right) + N/2$

$$T(N/4) = T\left(\frac{N}{8}\right) + N/4$$

$$T(2) = T(1) + 2$$

If we sum up. The equation will be $T(N) = T(1) + \left(2 + 4 + \cdots + \frac{N}{2} + N\right)$

$$T(N) = 1 + 2 + 4 + \cdots + \frac{N}{2} + N \approx 2N$$

$$O(2N) = O(N)$$

So it will be O(N).

Problem 7: Extra Credit

Firstly, we need have three heap. First one will be max heap called it as X, the second can be min heap we called it as Y, the last one is min heap we called it as Z. The size of second heap will be always be 1. If N is 1. We just put the first element in the first heap, and then we add new element into the heap, we pop the first element in the first heap the compare with the second element, if the second element is bigger, we put the second element into the third heap. The smallest will be median. If we need add the third element, it compares with the number in the third heap first, if it is bigger than the number in the third heap, it will be adding in the third heap. The number in the third heap will be pop into the second heap and it will be median. If this element is smaller than the number in the third heap and also bigger than the element in first heap, the second heap will add this number and it is median. If this third number is smaller than the number in the first heap. We need pop out the first element in the first heap and pop into the second heap. And we need add this third number into the first heap. The number in the second heap will be median.

Now if N>3, when we adding a number, it firstly compare with the number the third heap pop out, if the new number is bigger than the min number in the third heap , we firstly need pop the number in second heap into the first heap. And then pop the min number in third heap into the second heap, and add the new number into the third heap. The number in the second heap will also be median. If the new number is smaller than the min number in the third heap and also bigger than or equal to the number in the second heap. We only need add the new new number into the third heap, the second heap still hold the median. If the new number is smaller than number in the second heap and bigger than the number pop by first heap which is max number in the first heap, we need firstly pop the number in the second heap add into the third heap, and then add this new number into the second heap. The second heap will still hold the median number. If the new number is smaller than the number pop by the first heap. Firstly, we need pop the number in the second heap and add into the third heap. And then we need pop the number in the first heap and add into the second

heap. And then we need add the new number into the first heap. The second heap will still hold the median number.