Problem 1. In NP1 we found an efficient algorithm that finds the smallest and largest values in a list of size n, where n is a power of 2. In this problem, we will generalize the algorithm to work for all n.

(a) n even:

i. Give an algorithm that finds the smallest and largest values in a list of size n, where n is even. Minimize the number of comparisons.

ii. Analyze the exact number of comparisons.

(b) n odd:

i. Give an algorithm that finds the smallest and largest values in a list of size n, where n is odd. Minimize the number of comparisons.

ii. Analyze the exact number of comparisons.

(c) Give a single formula for the exact number of comparisons independent of whether n is even or odd. In other words the formula should not have cases.

Problem 2. Consider the following selection-sort-like algorithm: In the first iteration, find the smallest and largest values and put them at the beginning and the end of the list, respectively. In the second iteration, find the second smallest and second largest values and put them next to the beginning and next to the end of the list, respectively. In the third iteration, find the third smallest and third largest values and put them into their proper locations. Etc.

(a) Write the pseudo code for this version of selection sort. Assume that you have a method (as developed in Problem 1)

`MinMaxIndices(A,p,r)`

that returns the indices of the smallest and largest values in `A[p,...,r]`. Make your algorithm efficient with respect to number of comparisons.

(b) Analyze the exact number of comparisons for n even.

(c) Analyze the exact number of comparisons for n odd.

Problem 3. Give an $O(n \log n)$ algorithm to count the number of transpositions in a list of size n. It is sufficient to give very high level pseudo code or to explain the algorithm clearly in English. HINT: Use Mergesort.

Problem 4. Given two sorted arrays `A` and `B`, each of size n, we know that a worst case for merging the two lists into one sorted list `C` occurs when the two lists are interleaved. Give two simple functions

$$f : \{0, 1, 2, \ldots, n-1\} \to \{0, 1, 2, \ldots, 2n-1\} \quad \text{and} \quad g : \{0, 1, 2, \ldots, n-1\} \to \{0, 1, 2, \ldots, 2n-1\}$$

so that $f(i)$ gives the interleaved position of the ith value of `A` in the sorted list `C` and $g(i)$ gives the interleaved position of the ith value of `B` in the sorted list `C`. Assume that `A[0]<B[0]`.

For example, if `A` = (10, 30, 50, 70) and `B` = (20, 40, 60, 80) the two lists would be interleaved to give `C` = (10, 20, 30, 40, 50, 60, 70, 80).

Do not justify.

Problem 5. (**Bonus problem, 10 points. You must do this problem yourself, without help.**) Given a list of distinct values, we define the *rank* of a value to be the number of values that are smaller than it (so the smallest value has rank 0, the second smallest value has rank 1, etc.). Give a simple function

$$f : \{0, 1, 2, \ldots, n-1\} \to \{0, 1, 2, \ldots, n-1\}$$

where n is a power of 2, so that if the value with rank i is in position $f(i)$, and the list is sorted using Mergesort, every merge will interleave the values (in the two lists being merged). We also require that first value comes from the left list (which implies that the second value comes from the right list, etc.).

For example, a list of eight elements should start in the order

(10, 50, 30, 70, 20, 60, 40, 80)

When (10, 50) is merged with (30, 70), the two list will be interleaved giving (10, 30, 50, 70). When (20, 60) is merged with (40, 80), the two list will be interleaved giving (20, 40, 60, 80). Finally, when (10, 30, 50, 70) is merged with (20, 40, 60, 80) the two list will be interleaved giving the final sorted order.

Briefly explain how you got the answer. You do not need to justify.

Problem 6. **(Challenge Problem – will not be graded.)** Do Problem 5 for general n.