CP 312, Winter 2023
Assignment 2 (5% of the final grade)
(due Friday, February 17, at 23:30)

1. [6 marks] In the 2-Way-2- Sum decision problem, we are given arrays A and B of length n containing (not necessarily distinct) integers, and we must determine whether there are two pairs of indices $i_1, j_1, i_2, j_2 \in \{1, 2, \ldots, n\}$ (not necessary distinct) for which

$$A[i_1] + A[i_2] = B[j_1] + B[j_2].$$

   Design an efficient algorithm that solves the 2-Way-2- Sum problem and has time complexity $O(n^2 \log n)$ in the setting where operations on individual integers take constant time.

   Note, that brute force solution that tries all possible quadruples of indices will have the running time of $\Theta(n^4)$. Less straightforward solution that reduces the problem to 3- Sum which is described in lecture notes, will have the running time of $\Theta(n^3)$. So both of these solutions are unacceptable and will receive 0 marks.

   Your solution must include a description of the algorithm in words, the pseudocode for the algorithm, a justification of its correctness, and an analysis of its time complexity in big-Θ notation.

2. [8 marks] An array of n integers $x_1, x_2, \ldots, x_n$ is known to have the following property: elements follow in ascending order up to a certain index p where $1 < p < n$, then there are several equal entries up to a certain index q, where $p < q < n$ and then follow in descending order:

$$x_1 < x_2 < \ldots < x_{p-1} < x_p = x_{p+1} \ldots = x_q > x_{q+1} > \ldots > x_n.$$

   Give an efficient algorithm to find the indices p and q.
   Note: the worst case running time of your algorithm **must be in** $o(n)$, so simple scanning from left to right will get 0 marks.
   Detailed pseudocode along with the brief description of main idea and justification of correctness is required for full mark in this question.

3. [4 marks] Analyze the following pseudocode and give a tight (Θ) bound on the running time as a function of n. You can assume that all individual instructions are elementary. Show your work.

```
m := 1; s := 1;
while m <= n do
  for j = 1 to 2⌈log m⌉ do
     s := s + 1
  od
  m := 5 * m
od.
```

4. [6 marks] (a) [4 marks] Solve the following recurrence by the recursion-tree method (you may assume that n is a power of 2):

$$T(n) = \begin{cases} 17, & n = 1, \\ 3T(n/2) + 5 \cdot n, & n > 1. \end{cases}$$

(b) [2 marks] Solve part (a) using Master theorem.

5. [6 marks] Consider the following recursive algorithm prototype in pseudocode:

```
int Fiction( A:: array, n:: integer) {
 if (n>1) {
  B <- A[1],...,A[n/3];         // copy 1st "third" of A to B
  C <- A[n/3+1],...,A[2*n/3];   // copy 2nd "third" of A to C
  D <- A[2*n/3+1],...,A[n];     // copy 3rd "third" of A to D
  C <- Perturb(C);
  cond2 <- Fiction(C, n/3);
  cond3 <- Fiction(D, n/3);
  if (cond2) {
    cond1 <- Fiction(B, n/3);
    cond2 <- (cond1 + cond2)/2;
  }
  return cond2;
 }
 else
  return 1;
}
```

B C D

Perturb -> T(n) = O(n)

best case 2 recursive calls

worst case 3 recursive calls

(a) [4 marks] What is the worst case complexity of this algorithm assuming that the algorithm `Perturb` when applied to an array of length n has running time complexity $\Theta(n)$? To justify your answer provide and solve a divide-and-conquer recurrence for this algorithm. You may assume that n is a power of 3.

(b) [2 marks] What is the "best case" complexity of this algorithm under the same assumptions as in (a)?