Exam Imperative Programming

Friday December 8th 2017

- You can earn 90 points. You will get 10 points for free. So, you can obtain 100 points in total, and your exam grade is calculated by dividing your score by 10.
- This exam consists of 5 problems. The first two problems are multiple choice questions, and must be answered on the (separate) answer sheet. The problems 3, 4, and 5 are made using a computer.
- The problems 3, 4, and 5 are assessed by the Themis judging system. For each problem, there are 10 test cases. Each test case is worth 10% of the points.
- Note that manual checking is performed after the exam. For example, if a recursive solution is requested then a correct iterative solution will be rejected after manual checking, even though Themis accepted it. Also, precomputed answers will be rejected.
- This is an open book exam! You are allowed to use the pdf of the reader (which is available in Themis), pdfs of the lecture slides (also available in Themis), and the prescribed ANSI C book (hard copy). Any other documents are not allowed. You are allowed to use previous submissions that you made to Themis.
- Do not forget to hand in the answer sheet for the multiple choice part. You are allowed to take the exam text home!

Problem 1: Assignments (20 points)
For each of the following annotations determine which choice fits on the empty line (.....). The variables `x`, `y` and `z` are of type `int`. Note that `A` and `B` (uppercase letters!) are specification-constants (so not program variables).

```
1.1  /* x + 2*y == A, x + y == 2*A */
     .....
     /* y == A */

    (a) y = y/2;
    (b) y = -y;
    (c) y = (y-x)/2;

1.2 /* 2*x + y == A + B, x + y == 2*B */
    .....
    /* x == A, y == B */

    (a) x = (x - y)/2; y = (x + y)/2;
    (b) x = (3*x + y)/2; y = (x + y)/3;
    (c) y = (x + y)/2; x = (x - y)/2;

1.3 /* x + A == B, y + B == 2*A */
    .....
    /* x == 2*A - B, y == 2*B - 3*A */

    (a) y = x + y; x = x + y;
    (b) x = x + y; y = x + y;
    (c) y = x - y; x = x - y;
```

```
1.4 /* x == A + 2*B, y == A + B */
     y = x - y; x = x - y;
    .....

    (a) /* x == A + B, y == B */
    (b) /* x == A, y == B */
    (c) /* x == B, y == A */

1.5 /* x == A, y == 2*B */
     z = x; x = y; y = z;
     .....

    (a) /* x == 2*A, y == A */
    (b) /* x == 2*B, y == B */
    (c) /* x == 2*B, y == A */

1.6 /* x + z == A, y + z = A - B */
     x = x - y; z = y + z; y = x + z;
     .....

    (a) /* x == B, y == A */
    (b) /* x == -A, y == B */
    (c) /* x == B - A, y == A - B */
```

Problem 2: Time complexity (20 points)
In this problem the specification constant N is a positive integer (i.e. N>0). Determine for each of the following program fragments the sharpest upper limit for the number of calculation steps that the fragment performs in terms of N. For a fragment that needs N steps, the correct answer is therefore $O(N)$ and not $O(N^2)$ as $O(N)$ is the sharpest upper limit.

1.
```
int s=0;
for (int i=N; i > 0; i/=2) {
  for (int j=i; j > 0; j--) {
    s += i*i;
  }
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

2.
```
int s=0;
for (int i=N; i > 0; i--) {
  for (int j=i; j > 0; j/=2) {
    s += i*i;
  }
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

3.
```
int s=0;
for (int i=N; i > 0; i--) {
  for (int j=i; j > 0; j-=2) {
    s += i*i;
  }
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

4.
```
int s=0;
for (int i=0; i < N/i; i++) {
    s += i;
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

5.
```
int s=0, i=N;
while (i >= 0) {
  int d = 1 + i%5;
  s += d;
  i = (i%d == 0 ? i-1 : i/d);
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

6.
```
int j=0, s=0;
for (i=1; i < N; i*=2) {
  while (j < i) {
    s += (i+j);
    j++;
  }
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

Problem 3: Reordering (15 points)
Two integer arrays are called *equivalent* when they contain the same numbers with the same number of occurences. So e.g. `[0,1,1,2]` and `[2,1,0,1]` are equivalent, but `[0,1,1,2]` and `[0,0,1,2]` are not.

The input of this problem consists of two equally sized arrays filled with integers. Your program should output `YES` if the arrays are equivalent, otherwise it should print `NO`.

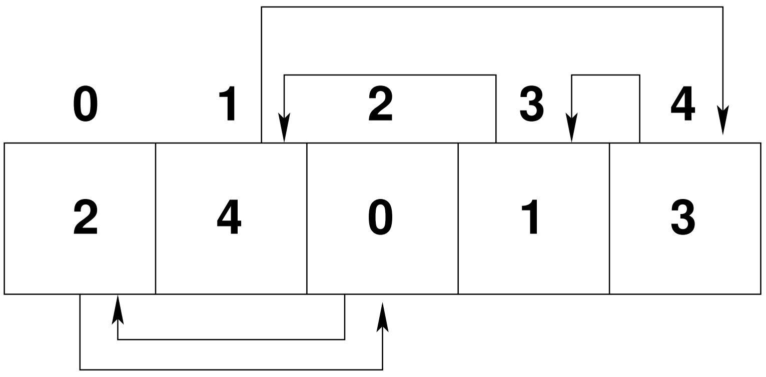
The first line of the input contains an integer n (where $0 < n \leq 100.000 = 10^5$), the size of the arrays. The next two lines both contain an array of n integers.

Example 1:
input:
```
6
0 1 2 3 4 5
5 4 3 0 1 2
```
output:
```
YES
```

Example 2:
input:
```
4
0 1 1 2
0 0 1 2
```
output:
```
NO
```

Example 3:
input:
```
5
0 0 0 0 1
1 0 0 0 0
```
output:
```
YES
```

Problem 4: Repeated permutations (15 points)
Consider the series `p=[2,4,0,1,3]`, which is a permutation (reordering) of the numbers $0, 1, 2, 3, 4$. The array `p` defines a permutation operator on an array with the same length as follows: for each location `i`, the data item stored at index `i` is moved to the location with index `p[i]`. So, for this concrete example, the data item at location 0 is moved to location 2, the data item at location 1 is moved to location 4, the data item at location 2 is moved to location 0, etc. This permutation is depicted in the following figure.



From the figure, it is clear that this permutation contains two cycles: the cycle $0 \rightarrow 2 \rightarrow 0$ has length 2 (steps), and the cycle $1 \rightarrow 4 \rightarrow 3 \rightarrow 1$ has length 3 (steps). Hence, if we apply this permutation twice to some input, then the values at the indexes 0 and 2 do not change, while the values at the indexes 1, 3, and 4 do change. Similarly, if we apply the permutation three times to some input, then the values of the indexes 0 and 2 get swapped, while the other values do not change. In conclusion, if we apply the permutation 6 times, then the result is the original input.

The input of this problem consists of two lines. The first line contains a positive integer `n` (where $1 \leq n \leq 100$), the size of the permutation. The second line consists of a reordering of the numbers $0..n$. The output of your program should be the smallest integer $m > 0$ such that applying the permutation m times to some input yields the original input.

Example 1:
input:
```
5
2 4 0 1 3
```
output:
```
6
```

Example 2:
input:
```
10
0 1 2 3 4 5 6 7 8 9
```
output:
```
1
```

Example 3:
input:
```
10
1 2 3 4 5 6 7 8 9 0
```
output:
```
10
```

Problem 5: Sums and products (20 points)
In this problem we consider, given a positive integer n, arithmetical expressions of the form

$$n = 1 \circ 2 \circ 3 \circ 4 \circ 5 \circ 6 \circ 7 \circ 8 \circ 9,$$

where $\circ$ can be plus (i.e. the addition operator $+$) or times (i.e. the multiplication operator $\times$).
For example, for $n = 45$, the following two expressions are the only valid ones of that form:

$$\begin{aligned} 45 &= 1+2+3+4+5+6+7+8+9 \\ 45 &= 1\times2\times3+4+5+6+7+8+9 \end{aligned}$$

Another example is the number $n = 121$, which can only be written as $1\times2+3\times4\times5+6\times7+8+9$. Note that multiplication has higher priority than addition, so $1\times2+3\times4\times5+6\times7+8+9 = (1\times2)+(3\times4\times5)+(6\times7)+8+9$.
Write a program that accepts on its input a positive integer `n` (where $0 < \mathtt{n} \leq 362880 = 9!$) and outputs the number of expressions of the above form that evaluate to n. You are not allowed to precompute the list of possible answers.

The following incomplete code fragment is available in Themis (file `prodsum.c`). Download it and complete the code. You are asked to implement the body of the function `prodSum` that returns the number of expressions that evaluate to n. The function should call a *recursive* helper function (with suitably chosen parameters/arguments) that solves the problem. You are not allowed to make changes in the `main` function.

```
#include <stdio.h>
#include <stdlib.h>

int prodSum(int n) {
  /* Implement the body of this function.
   * Moreover, this function should call a recursive helper
   * function that solves the problem.
   */
}

int main(int argc, char *argv[]) {
  int n;
  scanf("%d", &n);
  printf("%d\n", prodSum(n));
  return 0;
}
```

Example 1:
input:
```
45
```
output:
```
2
```

Example 2:
input:
```
121
```
output:
```
1
```

Example 3:
input:
```
42
```
output:
```
0
```