1. Permutacje

1 Zadanie

1.1 Wypisanie losowo wygenerowanych liczb z zadanego przedziału $[a,\ b]$

Szablon programu należy uzupełnić o definicję funkcji `rand_from_interval(...)`, która korzystając z bibliotecznej funkcji `rand()` i operacji dzielenia modulo zwraca liczbę z domkniętego przedziału $[a,\ b]$.

1. Założenie: liczba elementów zbioru, z którego odbywa się losowanie, nie jest większa od `RAND_MAX`+1.
2. Program wyprowadza m wygenerowanych liczb w kolejności zgodnej z kolejnością ich generowania.
3. Dla powtarzalności wyników, w funkcji `main()` jest wywoływana funkcja `srand(seed)`.

Wejście
1 `seed a b m`

Wyjście
m wylosowanych liczb całkowitych.

Przykład:

Wejście: `1 100 3 30 3`

Wyjście: `11 4 10`

1.2 Losowy wybór permutacji

Szablon programu należy uzupełnić o definicję funkcji `rand_permutation(...)`, która ma losowo wybrać jedną z permutacji n elementów zbioru liczb naturalnych. Elementy tego zbioru – liczby naturalne z przedziału $[0, n-1]$ – mają być wpisane do tablicy `a` w porządku rosnącym. Algorytm wpisywania liczb do tablicy oraz wyboru permutacji jest zapisany w pseudokodzie:

Require: $n \geqslant 0$
  for $i \leftarrow 0$ to $n-1$ **do**
    $a[i] \leftarrow i$
  end for
  for $i \leftarrow 0$ to $n-2$ **do**
    $k \leftarrow \text{random}(i, n-1)$ ▷ losowanie z przedzialu $[i, n-1]$
    $\text{swap}(a[i], a[k])$ ▷ zamiana elementow i i k tablicy a
  end for

Do losowania liczby z przedziału należy wykorzystać funkcję `rand_from_interval(a, b)`

Wejście
2 `seed n`

Wyjście
Wylosowana permutacja n liczb całkowitych.

Przykład:

Wejście: `2 20 10`

Wyjście: `1 0 3 4 6 2 8 9 5 7`

1.3 Sortowanie elementów tablicy metodą bąbelkową

Szablon programu należy uzupełnić o definicję funkcji `bubble_sort(...)`, która ma posortować rosnąco n elementową tablicę `tab` metodą bąbelkową.

Uwaga: Są dwa możliwe kierunki przeglądania elementów sortowanej tablicy - w kierunku rosnących albo malejących **indeksów** tej tablicy. Proszę zastosować ten pierwszy (od małych do dużych).

Uwaga ta nie dotyczy kierunku uporządkowania elementów, lecz kierunku przeglądania tablicy, czyli najpierw porównujemy `t[i]` z `t[i+1]`, później `t[i+1]` z `t[i+2]` itd., a nie `t[i]` z `t[i-1]`, później `t[i-1]` z `t[i-2]` itd.

Funkcja zwraca najmniejszą liczbę przeglądań tablicy (liczbę iteracji zewnętrznej pętli algorytmu sortowania), po której elementy tablicy są właściwie uporządkowane.

Program wywołuje tę funkcję z parametrami: `n` – daną wczytaną oraz tablicą `tab` o elementach wyznaczonych przez funkcję `rand_permutation()`. Dlatego wśród danych dla programu jest parametr `seed`.

Wejście
3 `seed n`

Wyjście
Numer iteracji pętli zewnętrznej, po której tablica była już uporządkowana, np.:

```
dla { 0 1 2 3 7 4 5 6 } wynik = 1,
dla { 1 2 3 7 4 5 6 0 } wynik = 7,
dla { 0 1 2 3 4 5 6 7 } wynik = 0.
```

Przykład:

Wejście: `3 20 10`

Wyjście: `3`