Zadanie 1

Napisz funkcję

```
void minMaxRep(int a[], size_t size, int& mn, size_t& in,
                                     int& mx, size_t& ix);
```

która pobiera tablicę **int**ów a, jej wymiar size oraz, przez referencje, cztery zmienne, do których wpisany ma być wynik działania funkcji: mn, in, mx i ix. Funkcja znajduje wartości najmniejszego i największego elementu tablicy i wpisuje je do zmiennych mn i mx, a do in i ix wpisuje, odpowiednio, liczbę wystąpień tej najmniejszej i największej wartości w całej tablicy.
Na przykład program

download *MinMaxRepF.cpp*

```
#include <iostream>

void minMaxRep(int a[], size_t size,
               int& mn, size_t& in, int&mx, size_t& ix) {
    // ...
}

int main() {
    using std::cout;
    int a[]{2,3,4,2,7,4,7,2};
    size_t size = sizeof(a)/sizeof(*a);
    int    mn, mx;
    size_t in, ix;
    minMaxRep(a,size,mn,in,mx,ix);
    cout << "Array: [ ";
    for (size_t i = 0; i < size; ++i)
        cout << a[i] << " ";
    cout << "]\n";
    cout << "Min = " << mn << " " << in << " times\n";
    cout << "Max = " << mx << " " << ix << " times\n";
}
```

powinien wydrukować

```
Array: [ 2 3 4 2 7 4 7 2 ]
Min = 2 3 times
Max = 7 2 times
```

Uwaga: nie wolno stosować żadnych dodatkowych tablic ani kolekcji. Funkcja może przebiec w pętli po elementach tablicy tylko raz. Liczby w tablicy są dowolnie duże, zarówno ujemne jak i dodatnie.

Zadanie 2

Napisz *rekurencyjną* funkcję **binSearchRec**

```
int binSearchRec(const int a[], int what,
                 int from, int to);
```

która pobiera

- *Posortowaną* tablicę **int**ów a;
- liczbę całkowitą what;
- zakres indeksów: od from do to.

Zadaniem funkcji jest zwrócenie indeksu, pod którym w tablicy a występuje element o wartości what, biorąc pod uwagę tylko elementy od tego o indeksie from *włącznie* do tego o indeksie to *wyłącznie*. Jeśli element o wartości what w tablicy nie występuje, funkcja zwraca -1.
Należy użyć wyszukiwania binarnego (które ma złożoność obliczeniową $\log n$, a nie liniową).

Następujący program

download *RecBinSearch.cpp*

```
#include <iomanip>
#include <iostream>

int binSearchRec(const int a[], int what,
                                int from, int to) {
    // ...
}
int main() {
    int a[]{1, 4, 5, 7, 9, 10};
    size_t sz = std::size(a);
    for (int i = a[0]; i <= a[sz-1]; ++i)
        std::cout << "what=" << std::setw(2) << i << " ind="
                  << std::setw(2)
                  << binSearchRec(a, i, 0, sz) << std::endl;
    std::cout << "**************\n";
    int b[]{-1,1,3,4,6};
    sz = std::size(b);
    for (int i = b[0]; i <= b[sz-1]; ++i)
        std::cout << "what=" << std::setw(2) << i << " ind="
                  << std::setw(2)
                  << binSearchRec(b, i, 0, sz) << std::endl;
}
```

powinien wydrukować

```
what= 1 ind= 0
what= 2 ind=-1
what= 3 ind=-1
```

```
what= 4 ind= 1
what= 5 ind= 2
what= 6 ind=-1
what= 7 ind= 3
what= 8 ind=-1
what= 9 ind= 4
what=10 ind= 5
**************
what=-1 ind= 0
what= 0 ind=-1
what= 1 ind= 1
what= 2 ind=-1
what= 3 ind= 2
what= 4 ind= 3
what= 5 ind=-1
what= 6 ind= 4
```

Zadanie 3

Napisz funkcję

```
void egyptian(int n, int d);
```

która wypisuje (lub, jeśli wolisz, zwraca **string**) ułamek $\frac{n}{d}$ w postaci „egipskiej", tzn. sumy liczby całkowitej i pewnej (jak najmniejszej) liczby ułamków o liczniku 1.

Na przykład poniższy program

download *Egyptian.cpp*

```cpp
#include <iostream>
#include <utility>

void egyptian(int n, int d) {
    // ...
}

int main() {
    std::pair<int,int> fracs[]{
        {3,4}, {7,9}, {0, 8}, {7, 0}, {123, 43}
    };
    for (auto [n, d] : fracs) {
        std::cout << n << '/' << d << " -> ";
        egyptian(n, d);
        std::cout << std::endl;
    }
}
```

powinien wypisać

```
3/4 -> 1/2 + 1/4
```

```
7/9 -> 1/2 + 1/4 + 1/36
0/8 -> 0
7/0 -> Denominator is 0!!!
123/43 -> 2 + 1/2 + 1/3 + 1/37 + 1/9546
```

Zadanie 4

Stwórz zestaw funkcji operujących na „zwykłych” (C-podobnych) tablicach oraz równoważne, ale operujące na obiektach **std::vector**. Implementując wersje dla **vector** możesz używać wersji „zwykłej” (dla C-tablic) pamiętając, że dla każdego wektora, powiedzmy vec, można uzyskać zwykłą tablicę, której jest „opakowaniem”, poprzez wywołanie `vec.data()`.

Funkcje do zaimplementowania są następujące (podana jest wersja dla C-tablic, ale trzeba też napisać odpowiedniki operujące na wektorach):

- `bool allDiff(const int arr[], size_t sz)` — pobiera tablicę i sprawdza, czy wszystkie jej elementy są różne;
- `int numDiff(const int arr[], size_t sz)` — pobiera tablicę, zwraca liczbę elementów różnych (na przykład 2 dla tablicy 1, 5, 1, 1, 5);
- `int fillWithPrimes(int arr[], size_t sz)` — pobiera tablicę o wymiarze sz, wypełnia ją kolejnymi liczbami pierwszymi i zwraca ostatnią z nich (może się przydać osobna mała funkcja sprawdzająca, czy dana liczba jest pierwsza);
- `int fillGaps(int arr[], size_t sz)` — pobiera tablicę i ją modyfikuje następująco: wszystkie elementy równe najmniejszemu nie są zmieniane; jeśli najmniejszą wartością jest, powiedzmy, 2, a nie ma trójek ani czwórek, to wszystkie piątki zamieniane są na trójki, itd. Na przykład tablica 2, 5, 7, 5 byłaby zastąpiona przez 2, 3, 4, 3;
- `size_t blockRem(int arr[], size_t sz, size_t from, size_t to)` — pobiera tablicę i dwa indeksy, from i to; następnie „usuwa” elementy od tego z indeksem from (włącznie) do elementu z indeksem to (ale *wyłącznie*). Oczywiście, nie jest możliwe prawdziwe usunięcie jakichkolwiek elementów tablicy; funkcja po prostu przesuwa elementu na prawo od usuwanego bloku nadpisując elementy z bloku. Funkcja zwraca nowy „logiczny” wymiar tablicy. Na przykład, jeśli from jest 2 a to jest 4, to tablica zawierająca 1, 2, 3, 4, 5, 6 będzie teraz zawierać na początku 1, 2, 5, 6, a zwrócone zotanie 4. Jeśli to jest większe od wymiaru tablicy, „usuwane” są wszystkie elementy od tego o indeksie from do końca.

Następujący program

download *ArrVecs.cpp*

```cpp
#include <iostream>
#include <vector>

bool isPrime(int n);
bool allDiff(const int arr[], size_t sz);
bool allDiff(const std::vector<int>& vec);
int numDiff(const int arr[], size_t sz);
```

```
int numDiff(const std::vector<int>& vec);
int fillWithPrimes(int arr[], size_t sz);
int fillWithPrimes(std::vector<int>& vec);
int fillGaps(int arr[], size_t sz);
int fillGaps(std::vector<int>& vec);
size_t blockRem(int arr[], size_t sz,
                size_t from, size_t to);
size_t blockRem(std::vector<int>& vec,
                size_t from, size_t to);

int main() {
    using std::cout; using std::vector;

    int a[]{3, 2, 3, 2, 5};
    size_t sza = sizeof(a)/sizeof(*a);
    vector<int> b(a, a+sza);
    cout << "allDiff: " << std::boolalpha
         << "a - " << allDiff(a, sza) << ", "
         << "b - " << allDiff(b) << '\n';
    cout << "numDiff: "
         << "a - " << numDiff(a, sza) << ", "
         << "b - " << numDiff(b) << '\n';

    int c[15];
    size_t szc = sizeof(c)/sizeof(*c);
    vector<int> d(szc,0);
    auto lastc = fillWithPrimes(c, szc);
    cout << "Primes: ";
    for (auto x : c) cout << x << " ";
    cout << "\n  Last: " << lastc << "\n";
    auto lastd = fillWithPrimes(d);
    cout << "Primes: ";
    for (auto x : d) cout << x << " ";
    cout << "\n  Last: " << lastd << "\n";

    int e[]{-3, 3, 5, -2, 8, 5, 8, -2};
    size_t sze = sizeof(e)/sizeof(*e);
    vector<int> f(e, e+sze);
    cout << "Filling gaps: ";
    for (auto x : e) cout << x << " ";
    auto laste = fillGaps(e, sze);
    cout << "\n     becomes: ";
    for (auto x : e) cout << x << " ";
    cout << "\n   max value: " << laste << "\n";
    cout << "Filling gaps: ";
```

```
    for (auto x : f) cout << x << " ";
    auto lastf = fillGaps(f);
    cout << "\n     becomes: ";
    for (auto x : f) cout << x << " ";
    cout << "\n   max value: " << lastf << "\n";

    int g[]{1, 2, 3, 4, 5, 6, 7};
    size_t szg = sizeof(g)/sizeof(*g);
    vector<int> h(g, g+szg);
    cout << "Original arr: ";
    for (auto x : g) cout << x << " ";
    auto newDimg = blockRem(g, szg, 2, 5);
    cout << "\nAfter 'removing': ";
    for (size_t i = 0; i < newDimg; ++i)
        cout << g[i] << " ";
    cout << "\n";
    cout << "Original vec: ";
    for (auto x : h) cout << x << " ";
    auto newDimh = blockRem(h, 2, 5);
    cout << "\nAfter   removing: ";
      // vector has been resized by the function
    for (auto x : h) cout << x << " ";
    cout << "\n";
}
```

powinien wydrukować

```
allDiff: a - false, b - false
numDiff: a - 3, b - 3
Primes: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
Last: 47
Primes: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
Last: 47
Filling gaps: -3 3 5 -2 8 5 8 -2
    becomes: -3 -1 0 -2 1 0 1 -2
max value: 1
Filling gaps: -3 3 5 -2 8 5 8 -2
    becomes: -3 -1 0 -2 1 0 1 -2
max value: 1
Original arr: 1 2 3 4 5 6 7
After 'removing': 1 2 6 7
Original vec: 1 2 3 4 5 6 7
After   removing: 1 2 6 7
```

Uwaga: nie twórz w swoich funkcjach żadnych pomocniczych tablic czy kolekcji.
