1. Sito Eratostenesa.

Sito Eratostenesa jest algorytmem, który szybko znajduje wszystkie **liczby pierwsze** z przedziału [2..n]. Inaczej mówiąc, przesiewa z tego zbioru liczby, w taki sposób, że zostają tylko pierwsze.

Algorytm opracował grecki matematyk, filozof, atronom - Eratostenes, który żył w latach ok. 276 r. p.n.e - ok. 194 r. p.n.e.

Strategia przesiewania liczb jest bardzo prosta. Tworzymy tablicę n elementową i wszystkie jej wartości ustawiamy na 0. Następnie rozpoczynamy od pierwszej liczby pierwszej (czyli 2), kierując się do komórki o tym indeksie. Komórkę 2 zostawiamy niezmienioną, natomiast już każdą wielokrotność dwójki ustawiamy na 1, co będzie oznaczać, że nie jest już ona pierwszą (bo przecież dzieli się przez 2). Pokażę to na przykładzie liczb z przedziału [2..25]

2, 3, /4, 5, /6, 7, /8, 9, /10, 11, /12, 13, /14, 15,

/16, 17, /18, 19, /20, 21, /22, 23, /24, 25

Teraz przechodzimy do następnej komórki w tablicy, w której wartość jest równa z 0. Będzie to numer 3 - ta komórka nie została oznaczona jako wielokrotność liczby 2, a więc jest ona pierwsza. Teraz każdą wielokrotność tej liczby wykreślamy ustawiając wartości tych komórek na 1:

2, 3, /4, 5, /6, 7, /8, /9, /10, 11, /12, 13, /14, /15,

/16, 17, /18, 19, /20, /21, /22, 23, /24, 25

W tym kroku nowo wykreślone liczby to: 9, 15 oraz 21.

W kolejnym kroku szukamy następnej komórki, której wartość jest równa 0. Jest nią liczba 5. Powtarzamy czynność wykreślając wielokrotności tej liczby (nowo wykreślona liczba to 25). Dalej nie musimy już wykreślać, ponieważ doszliśmy do liczby $\leq\sqrt{n}$. Uzasadnienie znajduje się w tym miejscu.

W ostateczności otrzymujemy:

2, 3, /4, 5, /6, 7, /8, /9, /10, 11, /12, 13, /14, /15,

/16, 17, /18, 19, /20, /21, /22, 23, /24, /25,

a liczby, które nie zostały wykreślone to:

2, 3, 5, 7, 11, 13, 17, 19, 23

a więc same liczby pierwsze.

Zaletą takiego rozwiązania jest na pewno szybkość, natomiast poważną wadą jest ograniczenie pamięci. Musimy zadeklarować tablicę na n+1

elementów. Można to nieco obejść stosując wykreślanie bitowe, co zaprezentuję w drugim rozwiązaniu.

Prześledźmy przykład, który wypisze wszystkie liczby pierwsze z przedziału [2..n]

```
//algorytm.edu.pl
#include<iostream>
using namespace std;

void sito(bool *tab, unsigned int n)
{
          for (int i=2; i*i<=n; i++)  //przeszukujemy kolejnych kandydatów na pierwsze
    {                                 //wystarczy sprawdzić do pierwiastka z n
                                      // i<=sqrt(n) - podnosząc do kwadratu mamy
                                      // i*i <= n
```

```
        if(!tab[i])                    //jesli liczba jest pierwsza(ma wartosc 0)
                  for (int j = i*i ; j<=n; j+=i) //to wykreslamy jej wielokrotnosci
           tab[j] = 1;                          //ustawiając wartosć na 1
    }
}

int main()
{
          int n;
          bool *tab;

          cout<<"Podaj zakres górny przedziału: ";
          cin>>n;

          tab = new bool [n+1];

          for(int i=2; i<=n; i++) //zerowanie tablicy
                    tab[i] = 0;

          sito(tab, n); //przesianie liczb

          cout<<"Kolejne liczby pierwsze z przedziału [2.."<<n<<"]: ";

          for(int i=2;i<=n;i++)
                    if(!tab[i])
                              cout<<i<<" ";

          delete []tab;

    return 0;
}
```

Drugie rozwiązanie oparte jest na odwoływaniu się do pojedynczych bitów liczby 32 bitowej. Zaletą jest to, że możemy zwiększyć maksymalny zakres tablicy w stosunku do poprzedniego rozwiązania ośmiokrotnie. Niestety program będzie wykonywał się wolniej z konieczności wykonywania obliczeń na bitach takich jak przesunięcie bitowe, koniunkcję i alternatywę bitową.

```
//algorytm.edu.pl
#include<iostream>
using namespace std;

void sito(unsigned int *tab, unsigned int n)
{
          for (int i=2; i*i<=n; i++)
          {
          if(!((tab[i>>5]>>(i&63))&1))
          for (int j = i*i ; j<=n; j+=i)
                    tab[j>>5] |= (1<<(j&63)); //ustawienie odpowiedniego bitu na 1
    }
}

int main()
{
          unsigned int n, *tab;

          cout<<"Podaj zakres górny przedziału: ";
          cin>>n;

          tab = new unsigned int [n/32 + 2]; //typ unsigned int składa się z 32 bitów

          for(int i=0; i<= n/32 + 1; i++) //zerowanie tablicy
                    tab[i] = 0;

          sito(tab, n); //przesianie liczb

          cout<<"Kolejne liczby pierwsze z przedziału [2.."<<n<<"]: ";

          for(int i=2;i<=n;i++)
          if(!((tab[i>>5]>>(i&63))&1))
                    cout<<i<<" ";

          delete [] tab;

    return 0;
}
```

2. Schemat Hornera

Schemat Hornera jest algorytmem służącym do bardzo szybkiego obliczania wartości wielomianu. Redukuje on ilość mnożeń do minimum. Przeanalizujmy następujący wielomian:

$$W(x)=3x^3+3x^2-2x+11$$

Do wyznaczenia wartości wielomianu tradycyjnym sposobem należy wykonać 6 mnożeń:

$$W(x)=3\cdot x\cdot x\cdot x+3\cdot x\cdot x-2\cdot x+11$$

Schematem Hornera tych mnożeń należy wykonać tylko 3:

$$W(x)=3\cdot x3+3x2-2x+11=x\cdot(3\cdot x2+3x-2)+11=x\cdot(x\cdot(3\cdot x+3)-2)+11$$

a więc mamy:

$$(1)\quad W(x)=x\cdot(x\cdot(3\cdot x+3)-2)+11$$

Dla wielomianu n-tego stopnia zwykle należy wykonać następującą ilość mnożeń (wliczając czynniki):

$$n+(n-1)+(n-2)+...+2+1=n\cdot(n+1)2$$

(suma ciągu arytmetycznego), natomiast dla omawianej metody zaledwie n mnożeń.

Jak widać na przykładzie (1) najpierw zaczniemy wykonywać działanie znajdujące się wewnątrz nawiasu:

$$3\cdot x+3$$

Następnie przemnażamy przez wartość argumentu x, potem odejmujemy 2 i znów przemnażamy przez x. Czynności powtarzamy do momentu obliczenia wartości całego wielomianu.

Danymi wejściowymi będą współczynniki wielomianu oraz jego stopień. Następnie podamy argument, dla jakiego chcemy wyznaczyć wartość wielomianu. Zauważmy, że wielomian, którego stopień jest równy n ma n+1czynników.

Rozwiązanie rekurencyjne w C++:

```
/*******www.algorytm.edu.pl***************/
#include<iostream>
using namespace std;

int horner(int wsp[],int st, int x)
{
          if(st==0)
                    return wsp[0];

          return x*horner(wsp,st-1,x)+wsp[st];
}

int main()
{
          int *wspolczynniki;
          int stopien, argument;

          cout<<"Podaj stopień wielomianu: ";
          cin>>stopien;

          wspolczynniki = new int [stopien+1];

          //wczytanie współczynników
          for(int i=0;i<=stopien;i++)
          {
                    cout<<"Podaj współczynnik stojący przy potędze "<<stopien-i<<": ";
                    cin>>wspolczynniki[i];
          }

          cout<<"Podaj argument: ";
```

```
	cin>>argument;

	cout<<"W( "<<argument<<" ) = "<<horner(wspolczynniki,stopien,argument)<<endl;

	delete [] wspolczynniki;

	return 0;
}
```

Rozwiązanie iteracyjne:

```
int horner(int wsp[],int st, int x)
{
	int wynik = wsp[0];

	for(int i=1;i<=st;i++)
		wynik = wynik*x + wsp[i];

	return wynik;
}
```