Základní třídící algoritmy

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava

Prezentace ke dni 12. února 2020





Osnova přednášky

Základní třídící algoritmy

- Asociativní třídicí algoritmy
- Třídění výběrem – SelectSort
- Třídění vkládáním – InsertSort
- Bublinové třídění – BubbleSort

Asociativní třídicí algoritmy

- jsou nejvšeobecnější skupinou třídících algoritmů,
- nepředpokládají o prvcích tříděné množiny nic víc než že jsou vybrané z uspořádaného univerza,
- jinak řečeno mezi prvky je definována relace „menší nebo rovno“,
- jsou založeny na operacích **porovnání** a **přesun** (výměna) dvou prvků,
- tyto dvě operace významně ovlivňují složitost asociativních třídicích algoritmů.

Složitost asociativních třídicích algoritmů

- Složitost asociativních třídících algoritmů budeme měřit počtem porovnání a přesunů prvků.
- Porovnání a přesuny prvků nemusí být triviální operace.
- Porovnání se týká jen klíče, zatímco přesun se týká celého záznamu.
- Markantní rozdíl mezi cenou porovnání a cenou přesunu se projeví zvláště tehdy, když je délka záznamu podstatně větší než délka klíče.

Složitost asociativních třídicích algoritmů

- Efektivnost některých asociativních třídicích algoritmů závisí od toho, v jakém pořadí jsou uspořádané prvky ve vstupní posloupnosti.
- Je-li vstupní posloupnost již setříděná, neměl by třídící algoritmus, po odhalení tohoto faktu, vykonávat žádnou činnost.
- Složitost třídícího algoritmu by měla vzrůstat s mírou nesetříděnosti posloupnosti.
- Třídící algoritmus by měl být **přirozený**.

Míra nesetříděnosti posloupnosti

- Budeme ji definovat jako počet inverzí v permutaci.
- Uvažujme permutaci $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$
- Řekneme, že dvojice různých prvků (i, j) představuje **inverzi permutace** π, jestliže

$$i \leq j \wedge \pi_i > \pi_j.$$

- Inverzi permutace můžeme volně interpretovat tak, že „na menším indexu je větší prvek a naopak na větším indexu je menší prvek“.
- Ještě jinak řečeno, tyto dva prvky jsou přehozeny, za předpokladu, že chceme posloupnost třídit vzestupně.

Míra nesetříděnosti posloupnosti

Příklad

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 3 & 2 \end{pmatrix}$$

Inverze:

- $1 < 2 \wedge 4 > 1$
- $1 < 3 \wedge 4 > 3$
- $1 < 4 \wedge 4 > 2$
- $3 < 4 \wedge 3 > 2$

Míra nesetříděnosti posloupnosti

- Počet inverzí setříděné posloupnosti je 0.
- Maximální počet inverzí, rovný $\frac{1}{2}n(n-1)$, obsahuje posloupnost setříděná v opačném pořadí.
- Průměrný počet inverzí v posloupnosti je přibližně $\frac{1}{4}n(n-1)$, za předpokladu, že každá posloupnosti je stejně pravděpodobná.
- Uvažujme všechny permutace n prvků $\{1, 2, \ldots, n\}$.
- Permutace n prvků vytvoříme přidáním čísla n k $(n-1)!$ permutacím $n-1$ prvků $\{1, 2, \ldots, n-1\}$.
- Číslo n můžeme přidat:
  - na poslední místo – pak se počet inverzí nezvýší,
  - na předposlední místo – pak přibude jedna inverze, a tak dále až
  - na první místo – pak přibude $n-1$ inverzí.

Míra nesetříděnosti posloupnosti

Pro průměrný počet inverzí I_n tedy bude platit:

$$\begin{aligned} n!I_n &= (n-1)!I_{n-1} + 0\cdot(n-1)! + \\ &\quad (n-1)!I_{n-1} + 1\cdot(n-1)! + \\ &\quad (n-1)!I_{n-1} + 2\cdot(n-1)! + \\ &\quad (n-1)!I_{n-1} + (n-1)\cdot(n-1)! \end{aligned}$$

Tedy

$$\begin{aligned} n!I_n &= n(n-1)!I_{n-1} + (0+1+2+\cdots+(n-1))\cdot(n-1)! \\ &= n!I_{n-1} + \frac{1}{2}n(n-1)(n-1)! \\ &= n!I_{n-1} + \frac{1}{2}(n-1)n! \end{aligned}$$

Míra nesetříděnosti posloupnosti

Odkud

$$
\begin{aligned}
I_n &= I_{n-1} + \frac{1}{2}(n-1) \\
&\vdots \\
&= \frac{1}{2}(n-1) + \frac{1}{2}(n-2) + \frac{1}{2}(n-3) + \frac{1}{2}\cdot 1 \\
&= \frac{1}{4}n(n-1)
\end{aligned}
$$

Asociativní třídicí algoritmy

Princip

Na algoritmy asociativního třídění se tedy můžeme dívat jako na algoritmy odstraňující postupně inverze ze vstupní posloupnosti pomocí vzájemných výměn vhodných prvků, do té doby, dokud posloupnost neobsahuje žádné inverze.

Vizualizace permutací

Permutaci

$$\pi = \begin{pmatrix} 1 & 2 & \dots & n \\ \pi_1 & \pi_2 & \dots & \pi_n \end{pmatrix}$$

můžeme znázornit jako posloupnost n svislých pruhů, kde

- pořadí pruhu odpovídá indexu i a
- výška pruhu odpovídá hodnotě π_i.

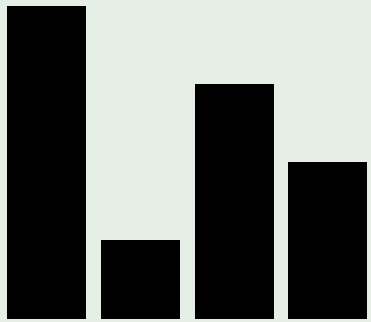
Šířka všech pruhů je stejná a ve vizualizaci nehraje roli.

Příklad

Permutaci

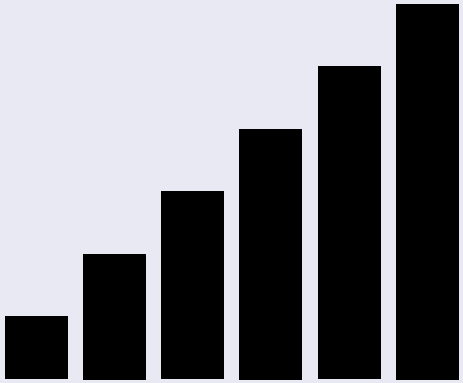
$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 3 & 2 \end{pmatrix}$$

odpovídá zobrazení

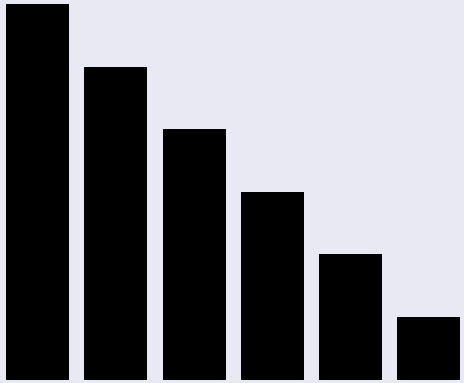


Vizualizace permutací – ukázky

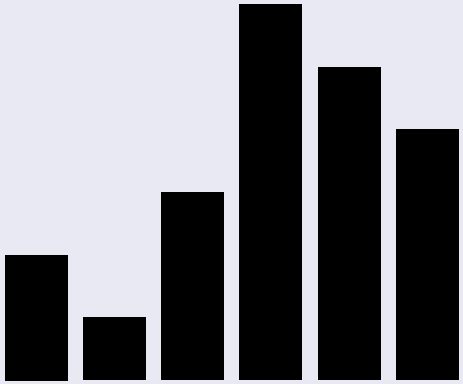
Identická



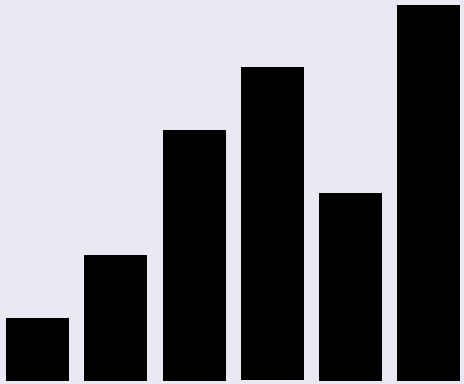
Opačně setříděná



Náhodná



Téměř setříděná



Obecná funkce pro výměnu dvou prvků pole

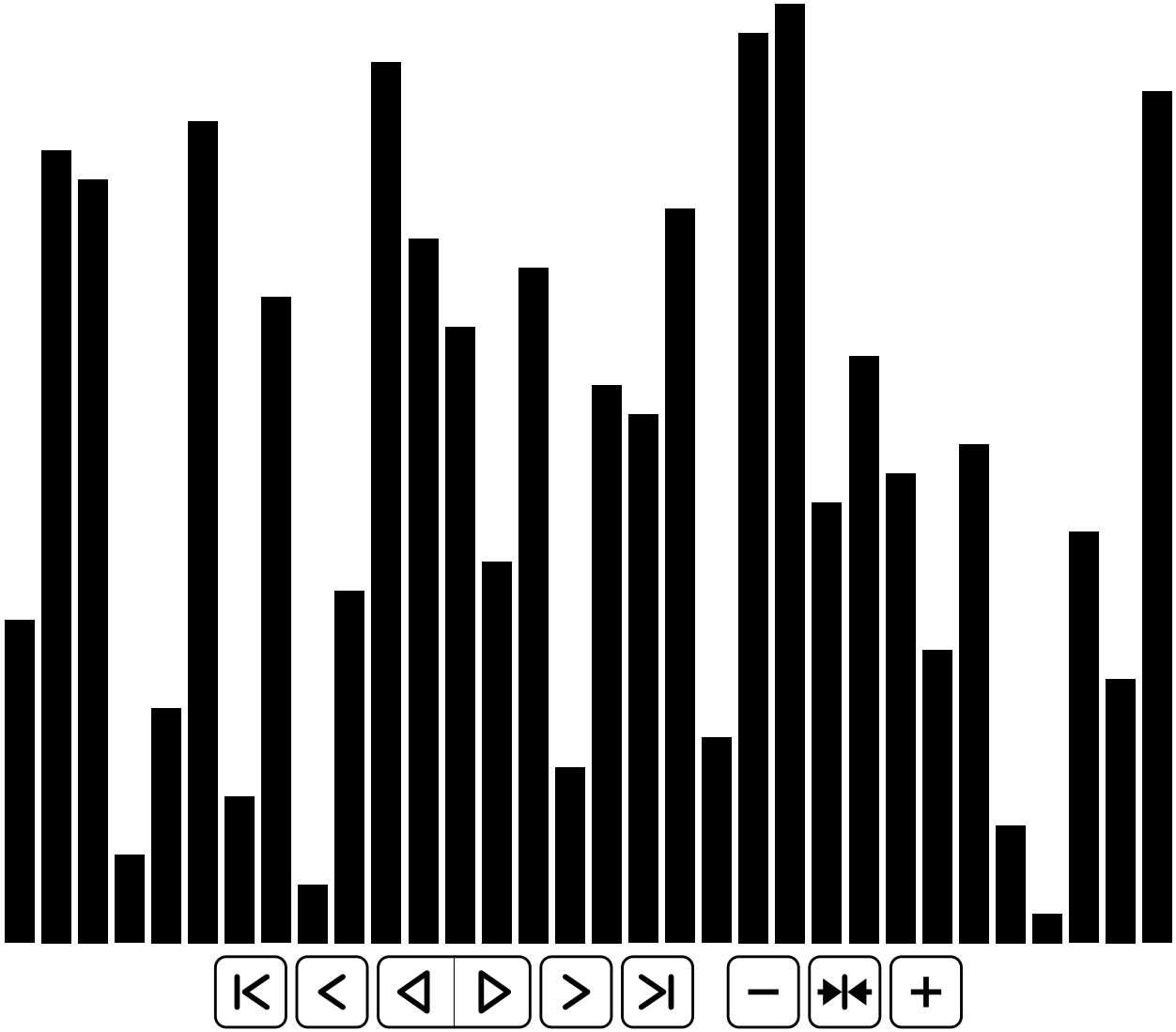
```
1 void Exchange(int& x, int& y)
2 {
3     int aux = x;
4     x = y;
5     y = aux;
6 }
```

Třídění výběrem – SelectSort

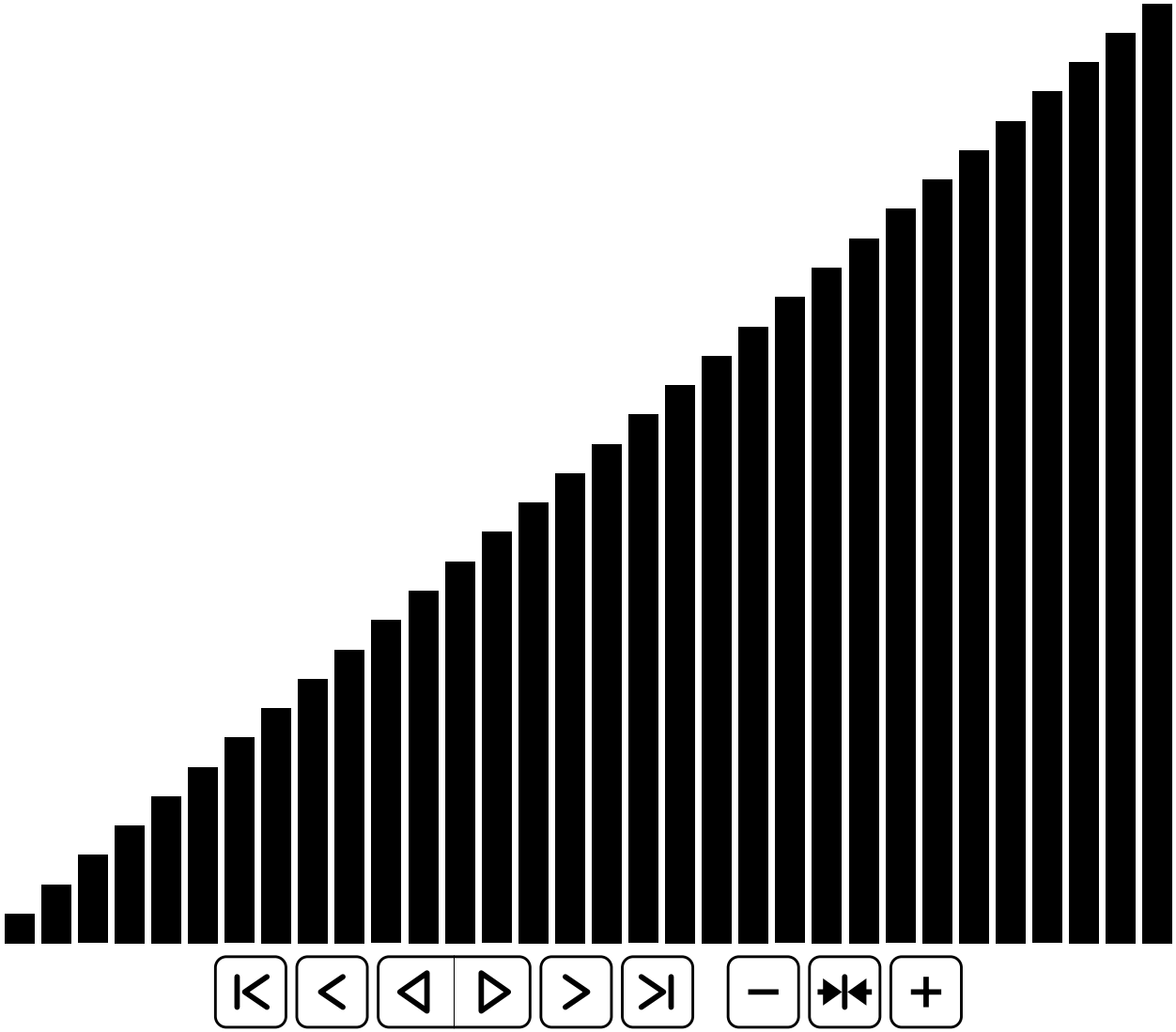
```
1  void SelectSort(int a[], const int n)
2  {
3     for (int i = 0; i < n; i++)
4     {
5        int min = i;
6        for (int j = i + 1; j < n; j++)
7        {
8           if (a[j] < a[min])
9           {
10             min = j;
11          }
12       }
13       Exchange(a[min], a[i]);
14    }
15 }
```

Nahrazení výměny prvků funkcí `Exchange`.

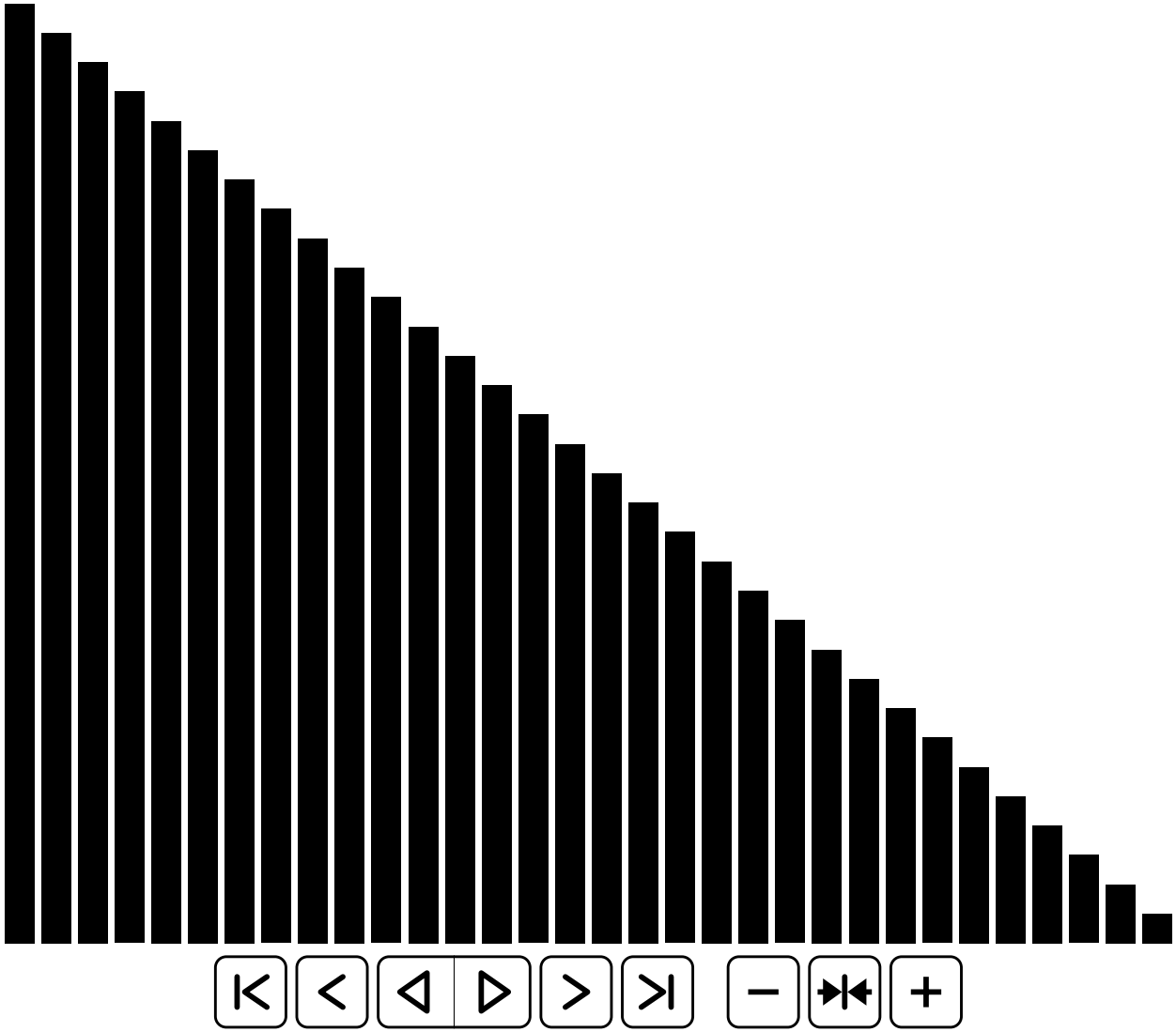
Třídění výběrem – náhodná permutace



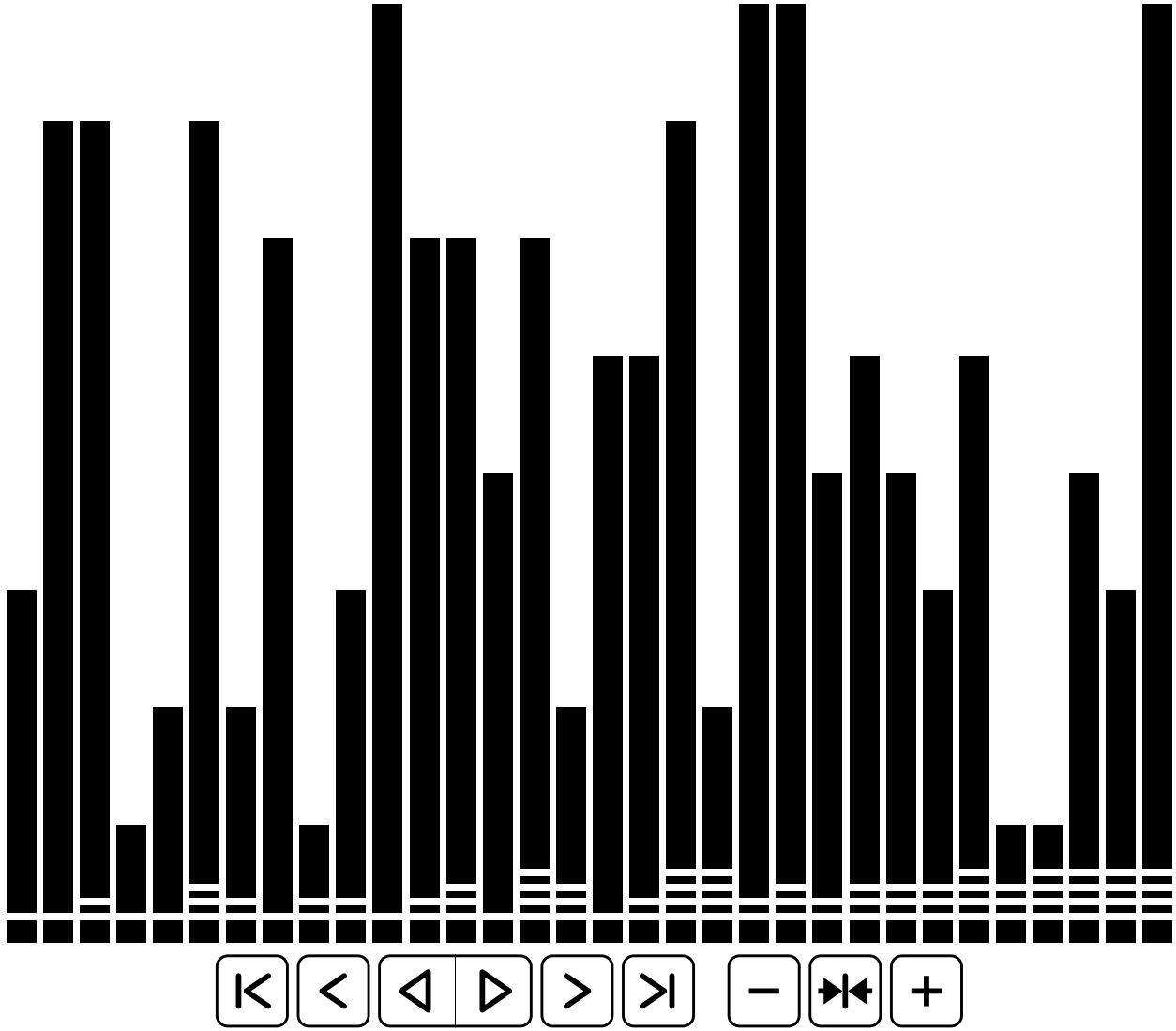
Třídění výběrem – identická permutace



Třídění výběrem – opačná permutace



Třídění výběrem – permutace pouze několika hodnot



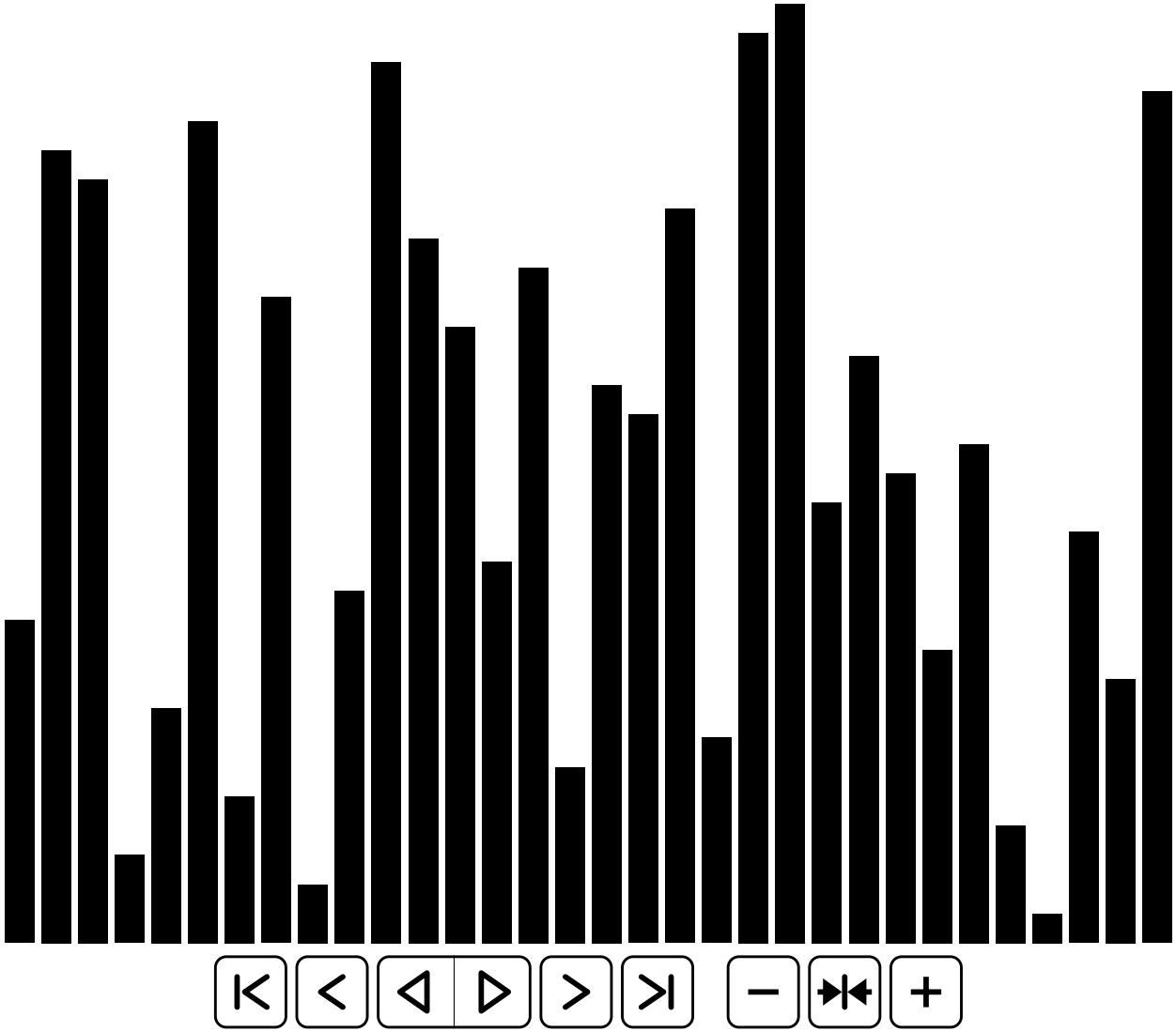
Třídění vkládáním – InsertSort

```
1  void InsertSort(int a[], const int n)
2  {
3     for (int i = 1; i < n; i++)
4     {
5        int v = a[i];
6        int j = i;
7        while (j > 0)
8        {
9           if (a[j - 1] > v)
10          {
11             a[j] = a[j - 1];
12             j -= 1;
13          }
14          else
15          {
16             break;
17          }
18       }
19       a[j] = v;
```

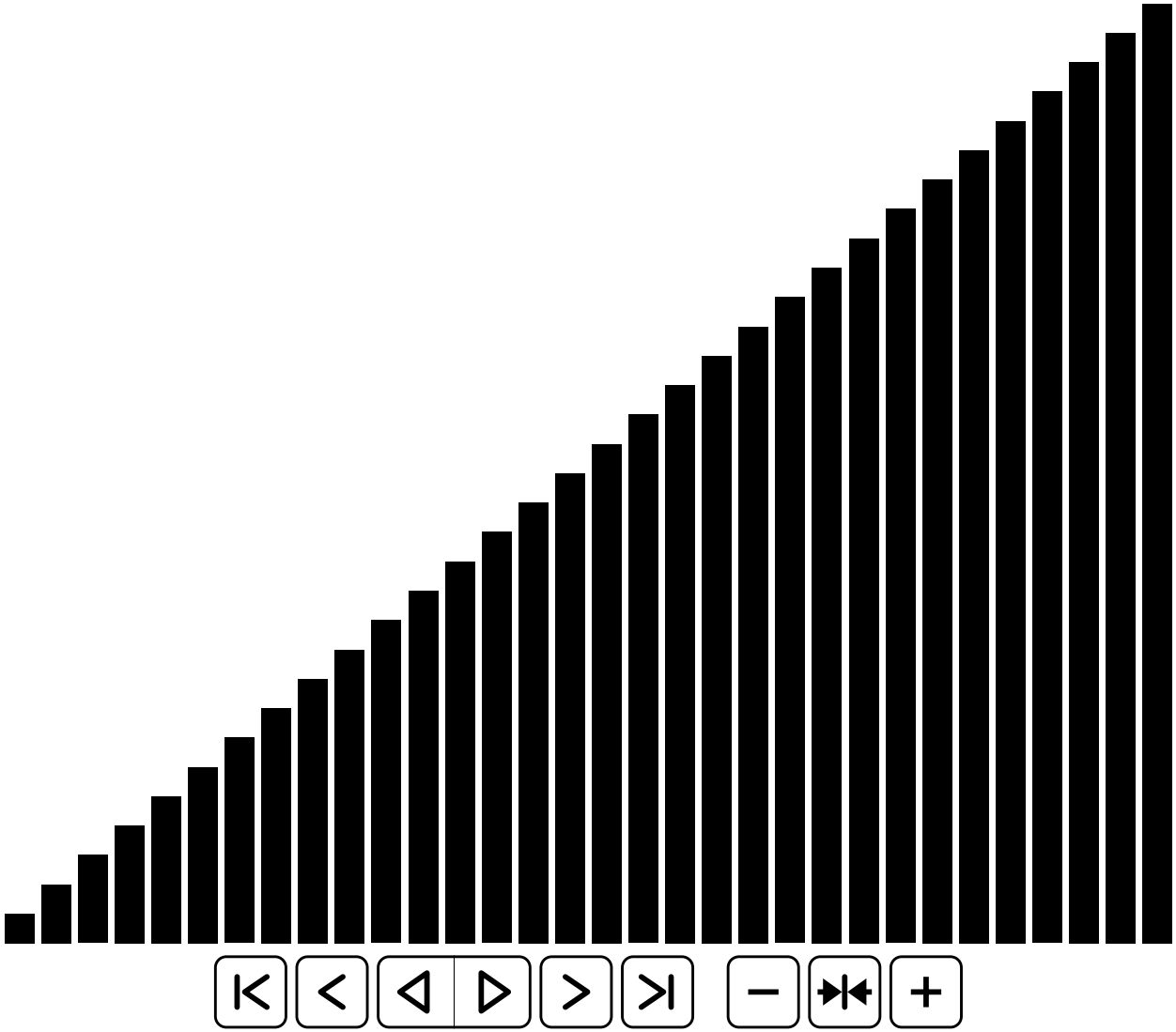
Třídění vkládáním – InsertSort (pokrač.)

```
20      }
21  }
```

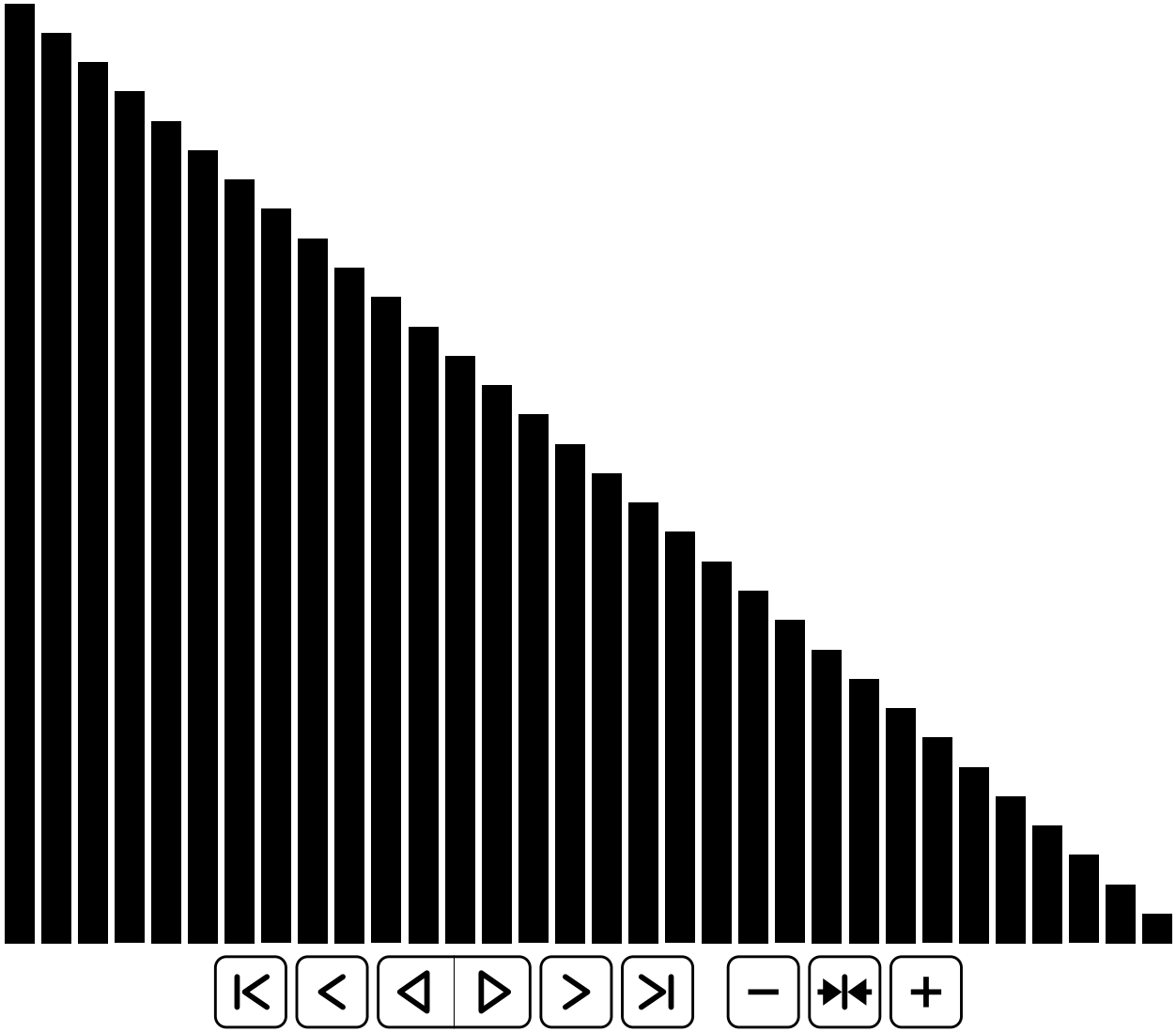
Třídění vkládáním – náhodná permutace



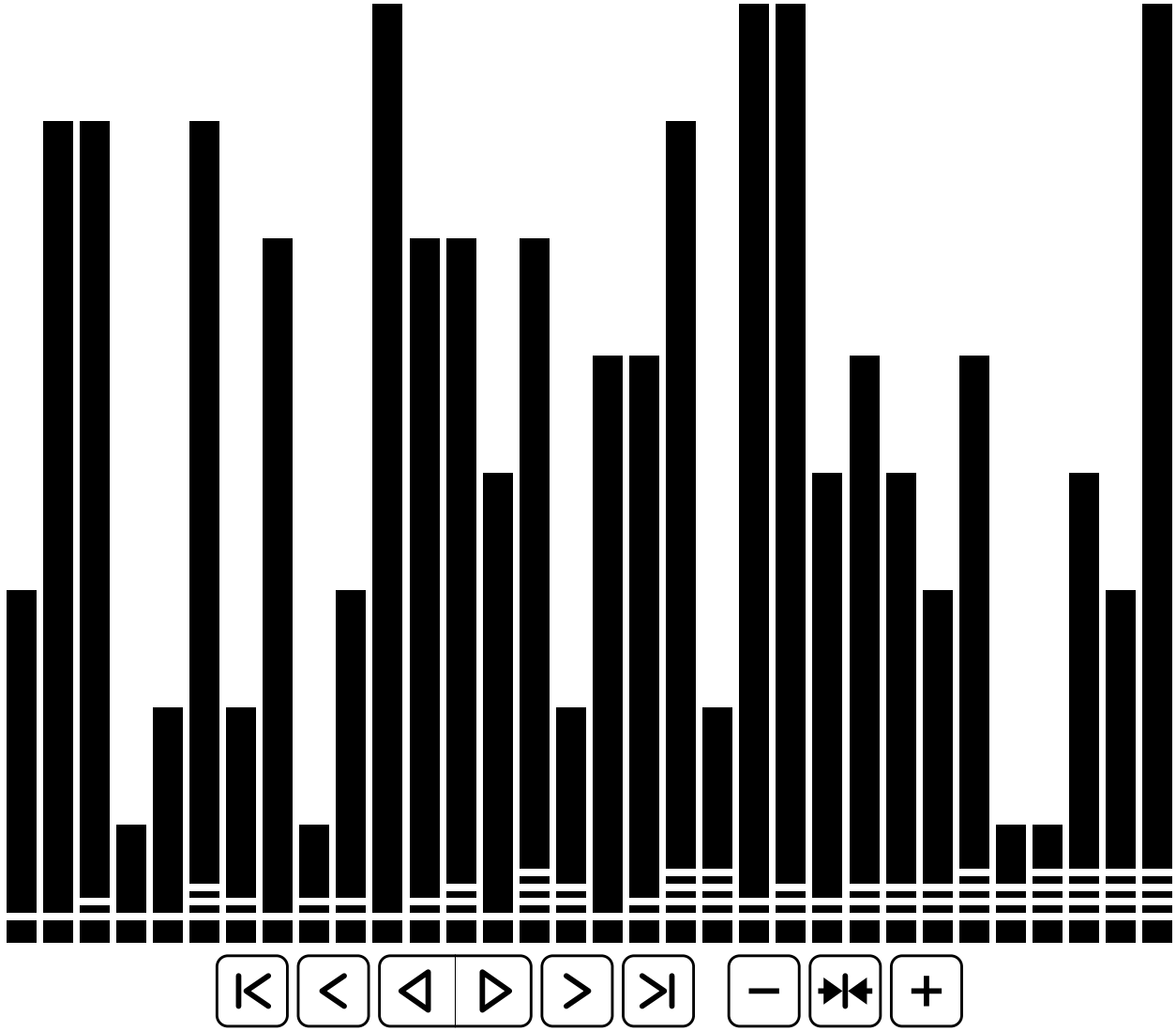
Třídění vkládáním – identická permutace



Třídění vkládáním – opačná permutace



Třídění vkládáním – permutace pouze několika hodnot



Bublinové třídění – BubbleSort

- Vezmu dvojici sousedních prvků a_i a a_{i+1}.
- Pokud tvoří inverzi, vyměním je.
- Pokračuji další dvojicí prvků a_i a a_{i+1}, ale pro $i + 1$ atd.
- Výměnou dvojice prvků odstraním jednu inverzi.
- Proč bublinové třídění – největší prvek „se rozjede“, „probublá“ směrem ke konci pole.

Bublinové třídění – BubbleSort

```
1  void BubbleSort0(int a[], const int n)
2  {
3     for (int i = 0; i < n; i++)
4     {
5        for (int j = 0; j < n - 1; j++)
6        {
7           if (a[j] > a[j + 1])
8           {
9              Exchange(a[j], a[j + 1]);
10          }
11       }
12    }
13 }
```

Tak takhle ne!

Tato implementace bublinového třídění je nejhorší možná! Takto by se bublinové třídění nemělo nikdy implementovat.

Bublinové třídění – BubbleSort

```
1  void BubbleSort1(int a[], const int n)
2  {
3     for (int i = n - 1; i > 0; i--)
4     {
5        for (int j = 0; j < i; j++)
6        {
7           if (a[j] > a[j + 1])
8           {
9              Exchange(a[j], a[j + 1]);
10          }
11       }
12    }
13 }
```

- Po každém průchodu se dostane jeden prvek určitě na své místo – probublá nahoru.
- Není nutné jej proto dále zpracovávat.
- Tříděnou posloupnost je možné o tento prvek zkrátit.

Bublinové třídění – BubbleSort

- Jiná možnost ukončení algoritmu – nedošlo k výměně prvků.
- Zavedeme si logickou proměnnou `AnyExchange`, kam si budeme ukládat informaci o případné výměně.
- Pokud algoritmus obdrží již setříděnou posloupnost – po jednom průchodu pozná, že je posloupnost již setříděná.

Bublinové třídění – BubbleSort

```
1  void BubbleSort2(int a[], const int n)
2  {
3     bool AnyExchange;
4     do
5     {
6        AnyExchange = false;
7        for (int i = 0; i < n - 1; i++)
8        {
9           if (a[i] > a[i + 1])
10          {
11             Exchange(a[i], a[i + 1]);
12             AnyExchange = true;
13          }
14       }
15    } while (AnyExchange);
16 }
```

Bublinové třídění – BubbleSort

- Kombinovaná možnost ukončení algoritmu:
  - nedošlo k výměně prvků a
  - zkracování posloupnosti o jeden prvek zprava.
- Zavedeme si logickou proměnnou `AnyExchange`, kam si budeme ukládat informaci o případné výměně.
- Proměnná `Right` bude označovat pravý okraj tříděné posloupnosti.

Bublinové třídění

```
1  void BubbleSort3(int a[], const int n)
2  {
3     bool AnyExchange;
4     int Right = n - 1;
5     do
6     {
7        AnyExchange = false;
8        for (int i = 0; i < Right; i++)
9        {
10          if (a[i] > a[i + 1])
11          {
12             Exchange(a[i], a[i + 1]);
13             AnyExchange = true;
14          }
15       }
16       Right -= 1;
17    } while (AnyExchange);
18 }
```

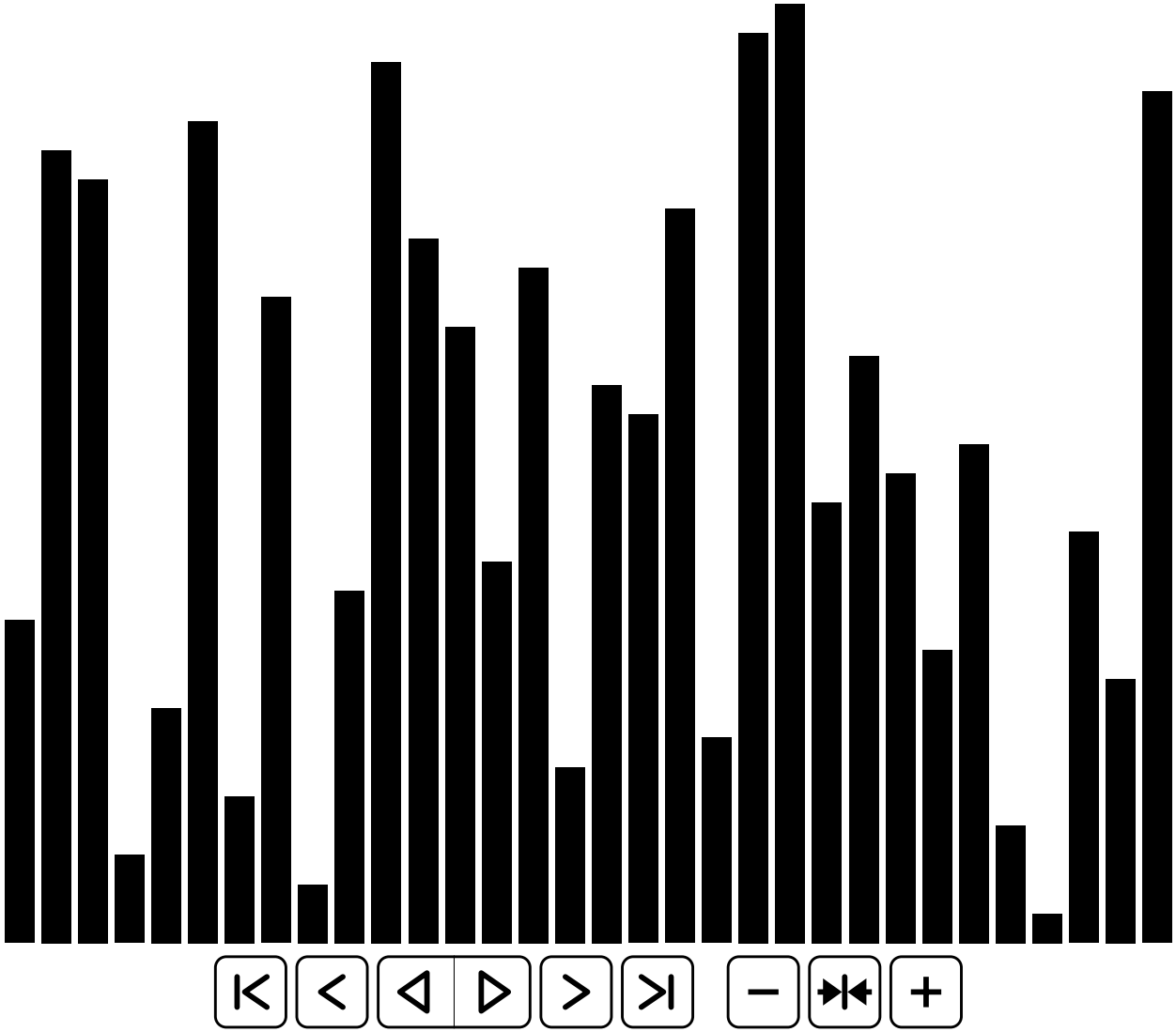
Bublinové třídění – BubbleSort

- Kombinovaná možnost ukončení algoritmu:
  - nedošlo k výměně prvků a
  - zkracování posloupnosti na index poslední výměny.
- Zavedeme si logickou proměnnou `AnyExchange`, kam si budeme ukládat informaci o případné výměně.
- Proměnná `LastExchangeIndex` bude označovat pravý okraj tříděné posloupnosti, index poslední výměny prvků.

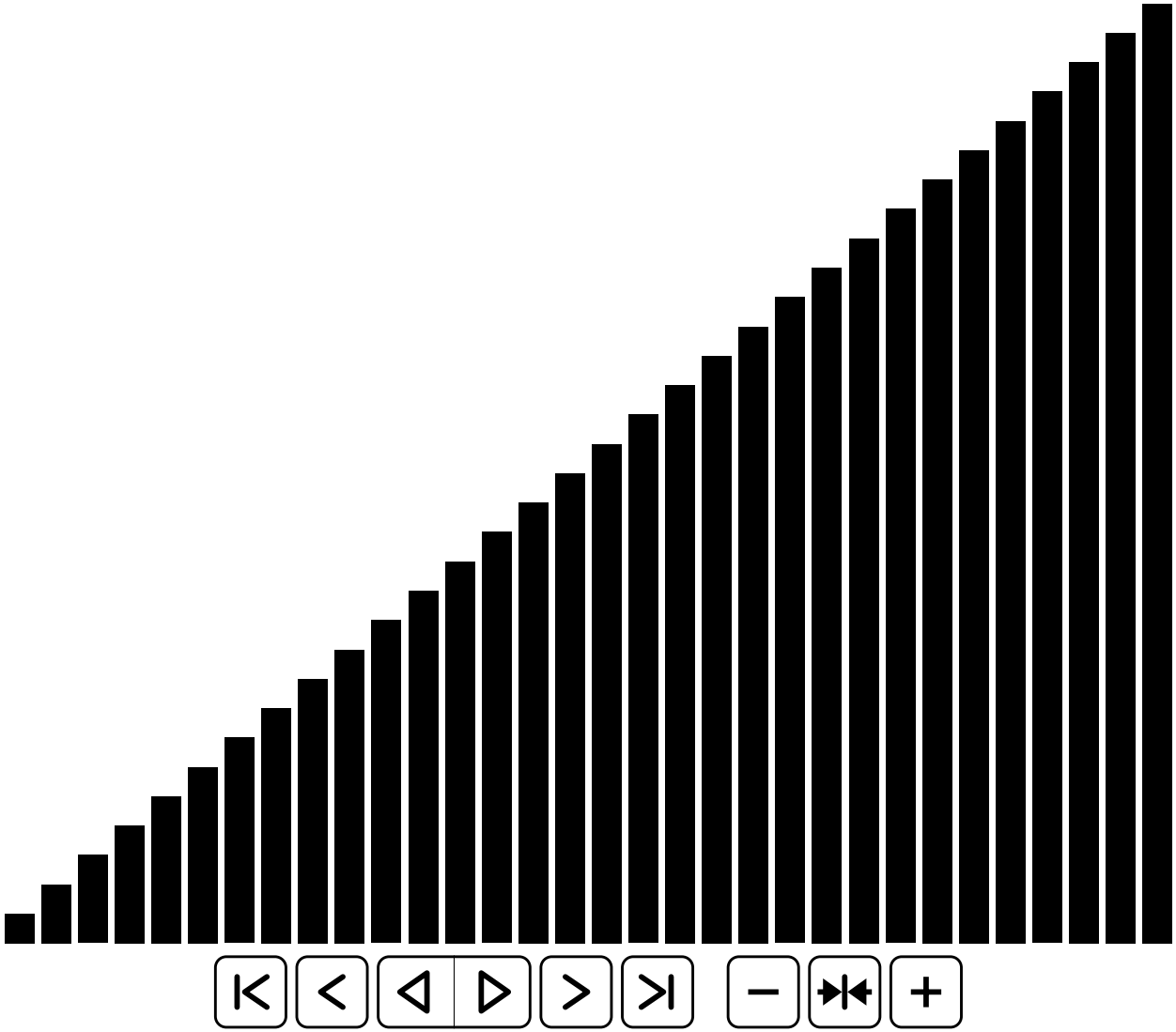
Bublinové třídění

```
1  void BubbleSort4(int a[], const int n)
2  {
3     int Right = n - 1;
4     int LastExchangeIndex;
5     do
6     {
7        LastExchangeIndex = 0;
8        for (int i = 0; i < Right; i++)
9        {
10          if (a[i] > a[i + 1])
11          {
12             Exchange(a[i], a[i + 1]);
13             LastExchangeIndex = i + 1;
14          }
15       }
16       Right = LastExchangeIndex;
17    } while (LastExchangeIndex > 0);
18 }
```

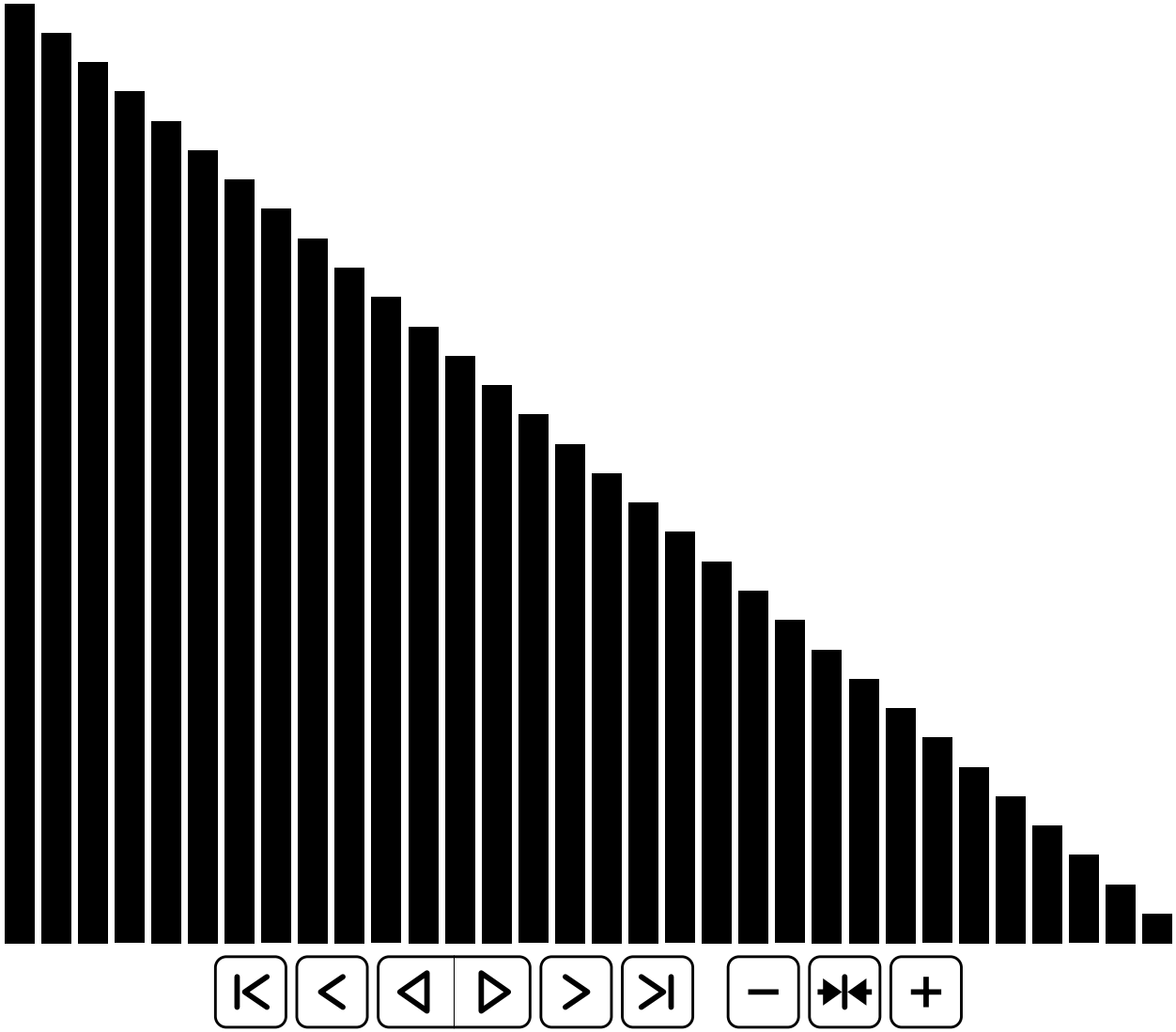
Bublinové třídění – náhodná permutace



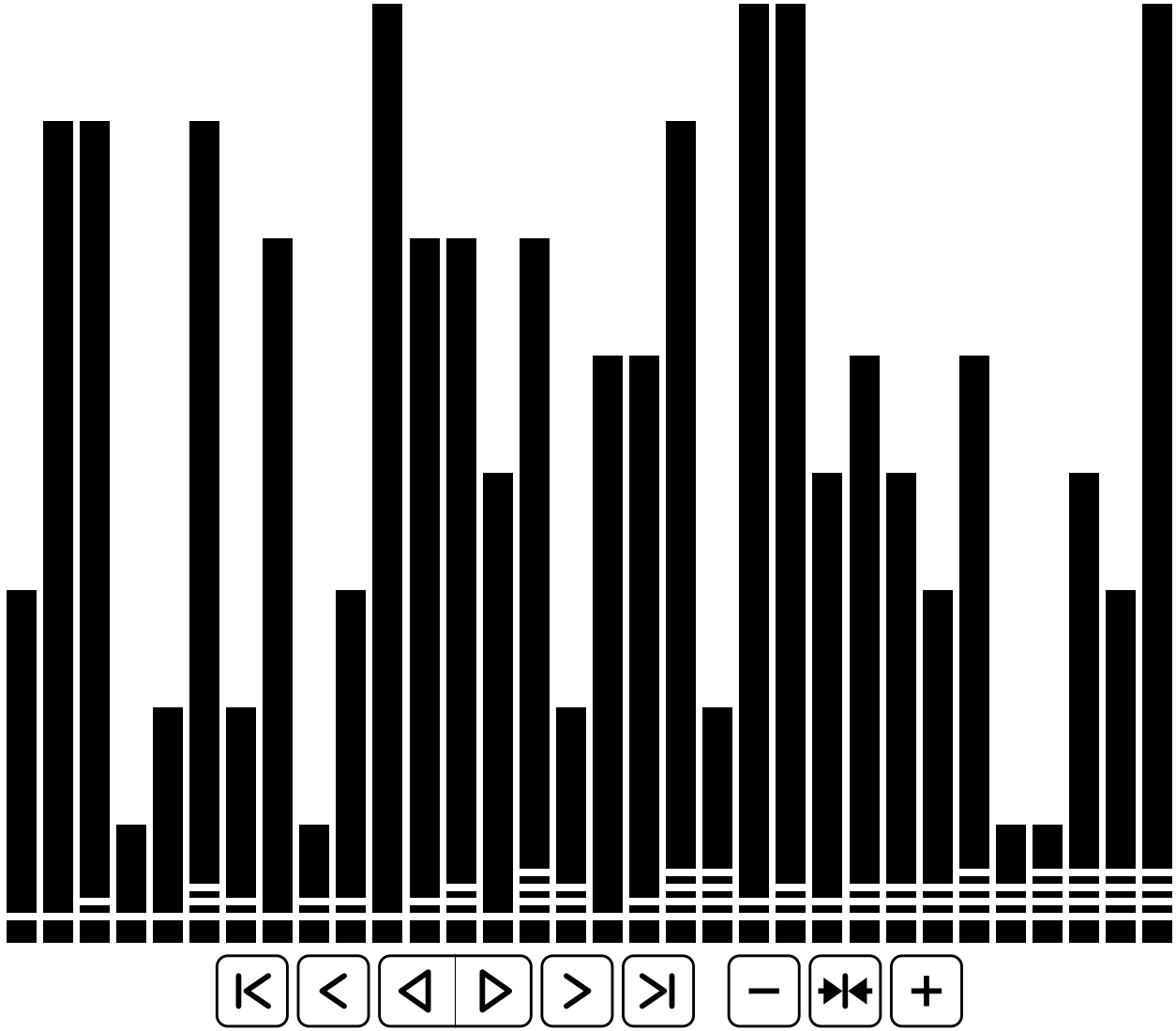
Bublinové třídění – identická permutace



Bublinové třídění – opačná permutace



Bublinové třídění – permutace pouze několika hodnot



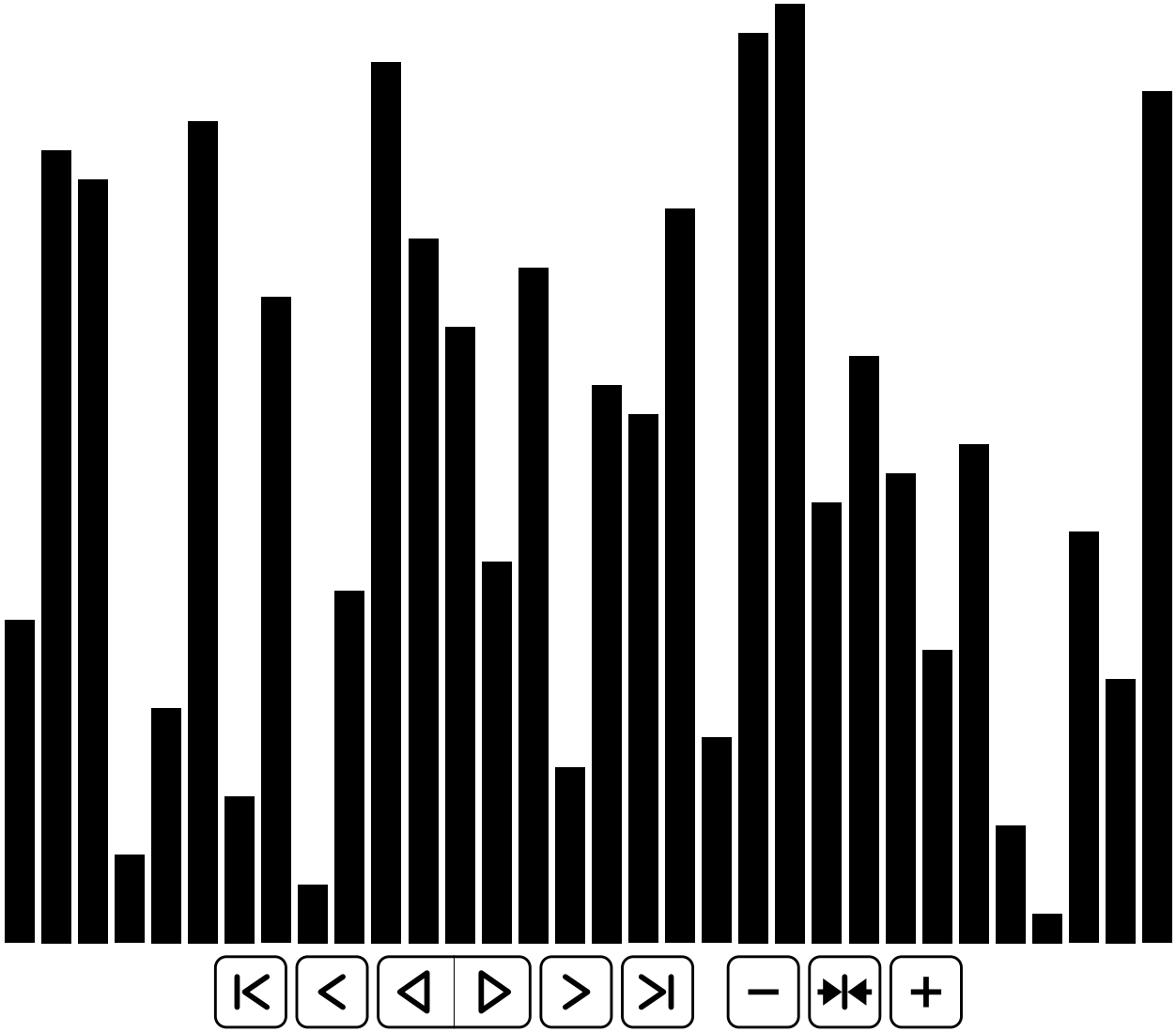
Třídění přetřásáním

```
1  void ShakerSort(int a[], const int n)
2  {
3     int ExchangeIndex = 0;
4     int Left = 0;
5     int Right = n - 1;
6     do
7     {
8        for (int i = Left; i < Right; i++)
9        {
10          if (a[i] > a[i + 1])
11          {
12             Exchange(a[i], a[i + 1]);
13             ExchangeIndex = i;
14          }
15       }
16       Right = ExchangeIndex;
17       for (int i = Right; i > Left; i--)
18       {
19          if (a[i - 1] > a[i])
```

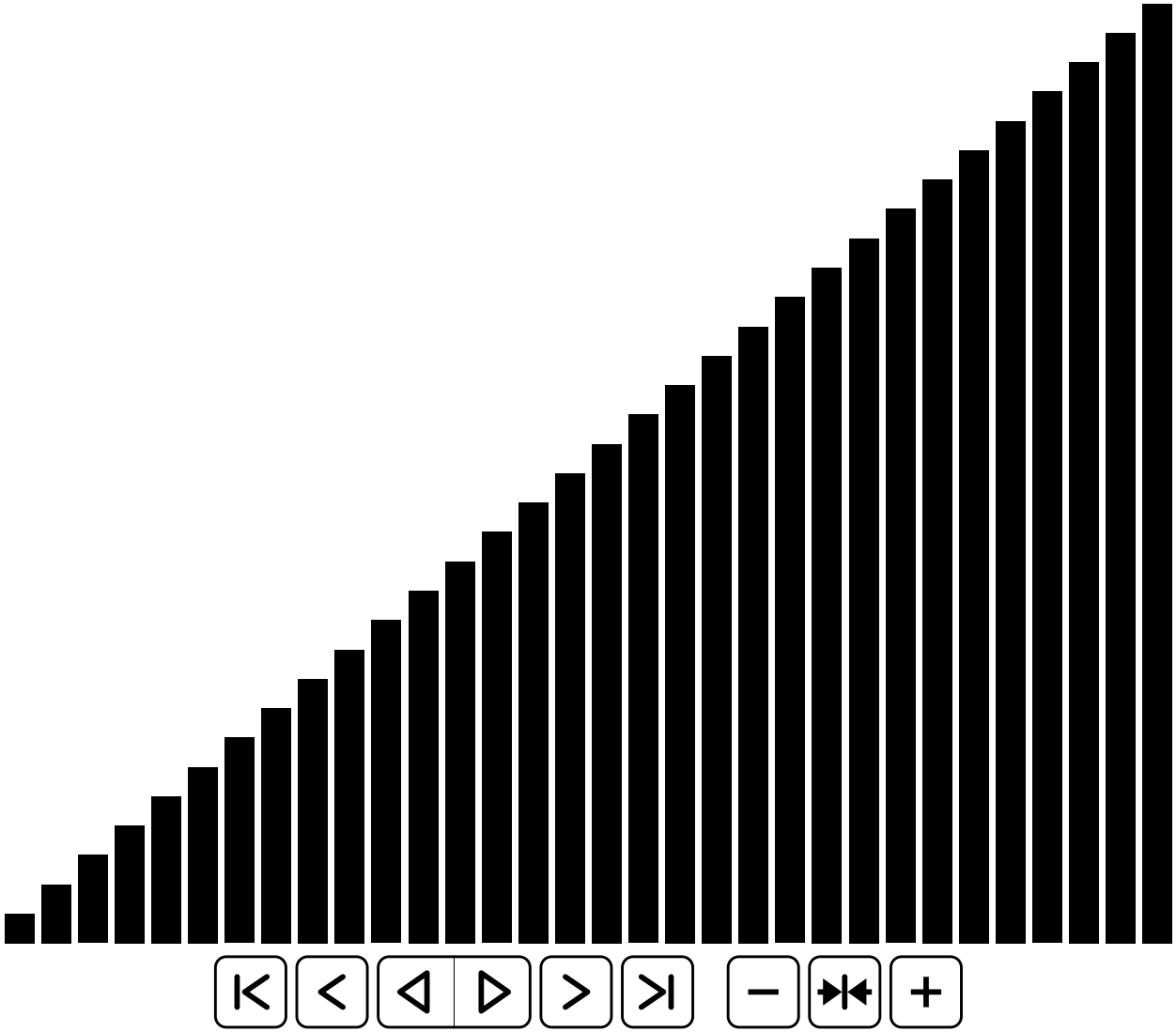
Třídění přetřásáním (pokrač.)

```
20          {
21              Exchange(a[i - 1], a[i]);
22              ExchangeIndex = i;
23          }
24      }
25      Left = ExchangeIndex;
26    } while (Left < Right);
27 }
```

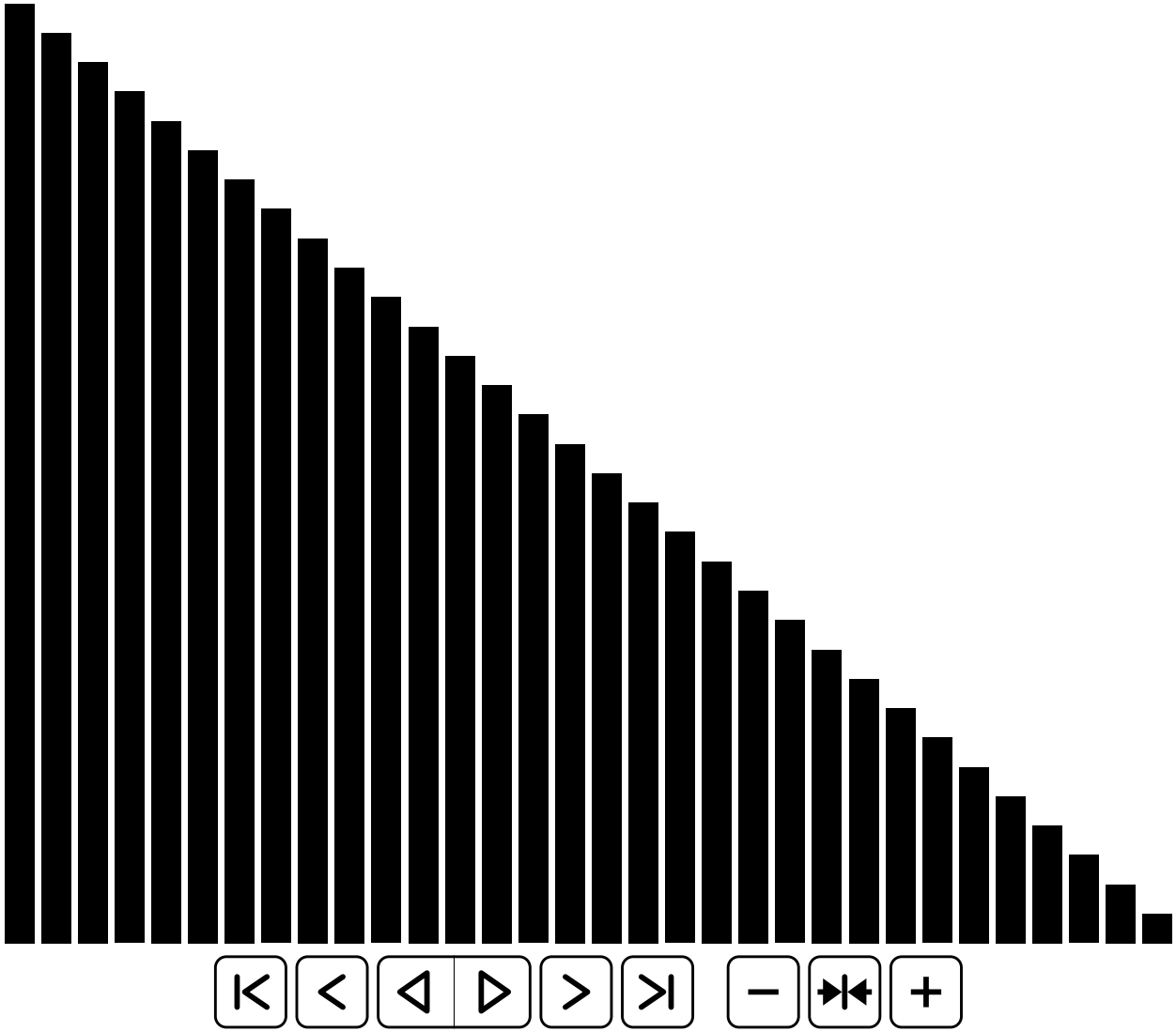
Třídění přetřásáním – náhodná permutace



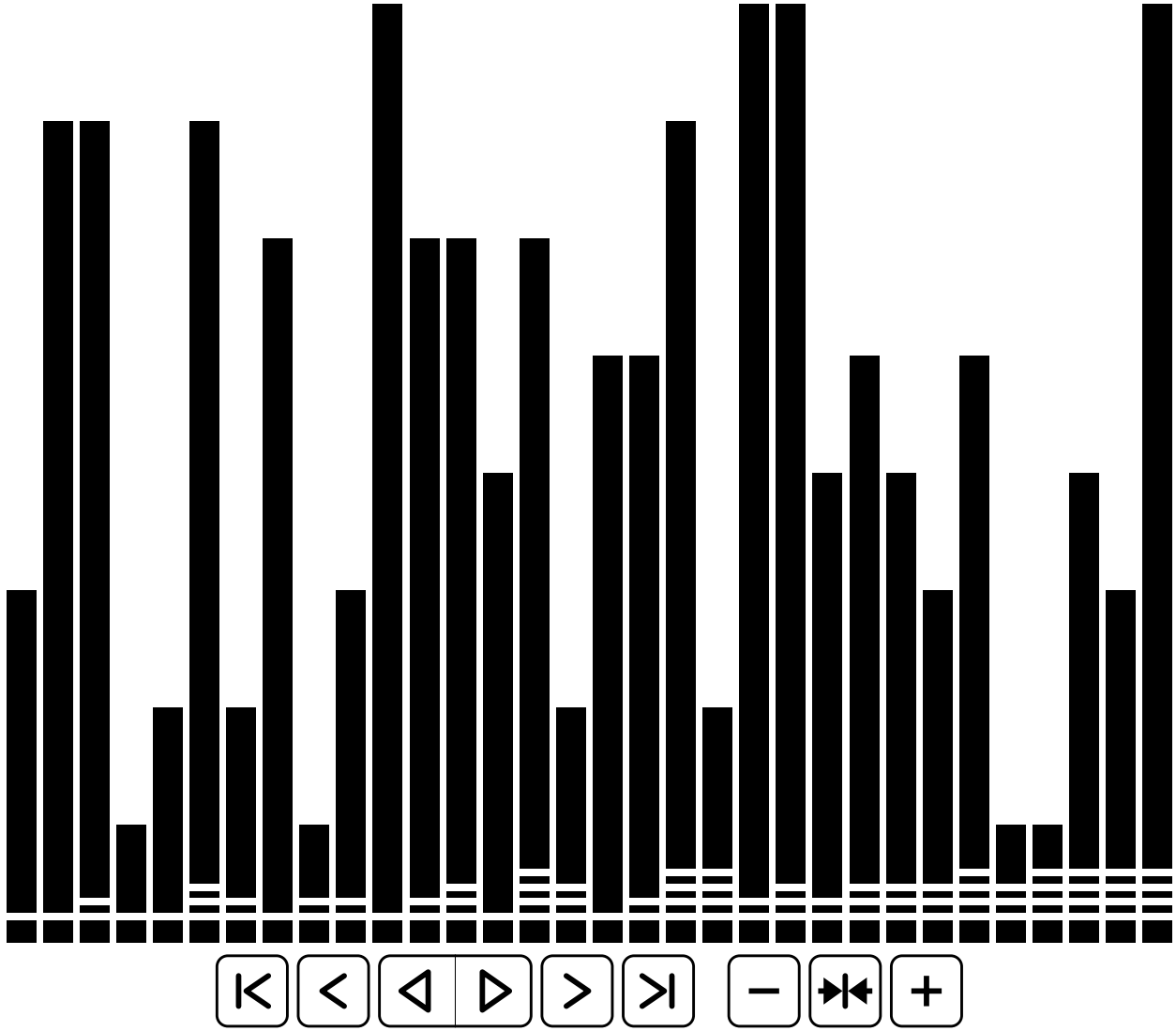
Třídění přetřásáním – identická permutace



Třídění přetřásáním – opačná permutace



Třídění přetřásáním – permutace pouze několika hodnot



Děkuji za pozornost