Pokročilé 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

Pokročilé třídící algoritmy

- QuickSort – Třídění rozdělováním
- Třídění haldou – HeapSort

QuickSort – Třídění rozdělováním

- C. A. R. Hoare, 1962.
- využití metody „rozděl a panuj“ (divide et impera),
- rekurzivní algoritmus,
- výměny na velkou vzdálenost,
- v každém kroku rozdělení tříděného úseku pole na dvě disjunktní části,
- všechny prvky v levé části jsou menší než všechny prvky v pravé částí,
- rozdělení pomocí prvku pole zvaného **pivot**,
- levá a pravá část pole je tříděna pomocí rekurze stejným způsobem.

QuickSort, výběr pivota

- pivotem může být libovolný prvek ze tříděného úseku pole,
- tříděný úsek pole se prochází:
  1. zleva a hledá se prvek větší než pivot,
  2. zprava a hledá se prvek menší než pivot,
  3. takto nesprávně zařazené prvky se vzájemně prohodí.
- až se indexy kterými pole zleva a zprava „potkají“ je tříděný úsek pole rozdělen na dvě části
- algoritmus funguje vždy, **ale**
- výběr pivota silně ovlivňuje složitost algoritmu!

QuickSort, výběr pivota

- ideální případ – pivot rozdělí tříděný úsek pole na dvě přibližně stejné části – **medián**,
- nejhorší případ – pivot rozdělí tříděný úsek pole na část s jedním prvkem a zbytkem – **minimum** či **maximum**,
- další možnosti:
  - prvek ve středu tříděného úseku,
  - medián ze tří prvků tříděného úseku,
  - první prvek,
  - libovolný prvek.

Složitost algoritmu QuickSort

- průměrná časová složitost $O(n \log_2 n)$
- nejhorší časová složitost $O(n^2)$!

QucikSort – Třídění rozdělováním

```
1  void QuickSortInternal(int a[], const int l, const int r)
2  {
3     int i = l;
4     int j = r;
5     int pivot = a[(l + r) / 2];
6     do
7     {
8        while (a[i] < pivot)
9           i += 1;
10       while (pivot < a[j])
11          j -= 1;
12       if (i <= j)
13       {
14          Exchange(a[i], a[j]);
15          i += 1;
16          j -= 1;
17       }
18    } while (i <= j);
19    if (l < j)
```

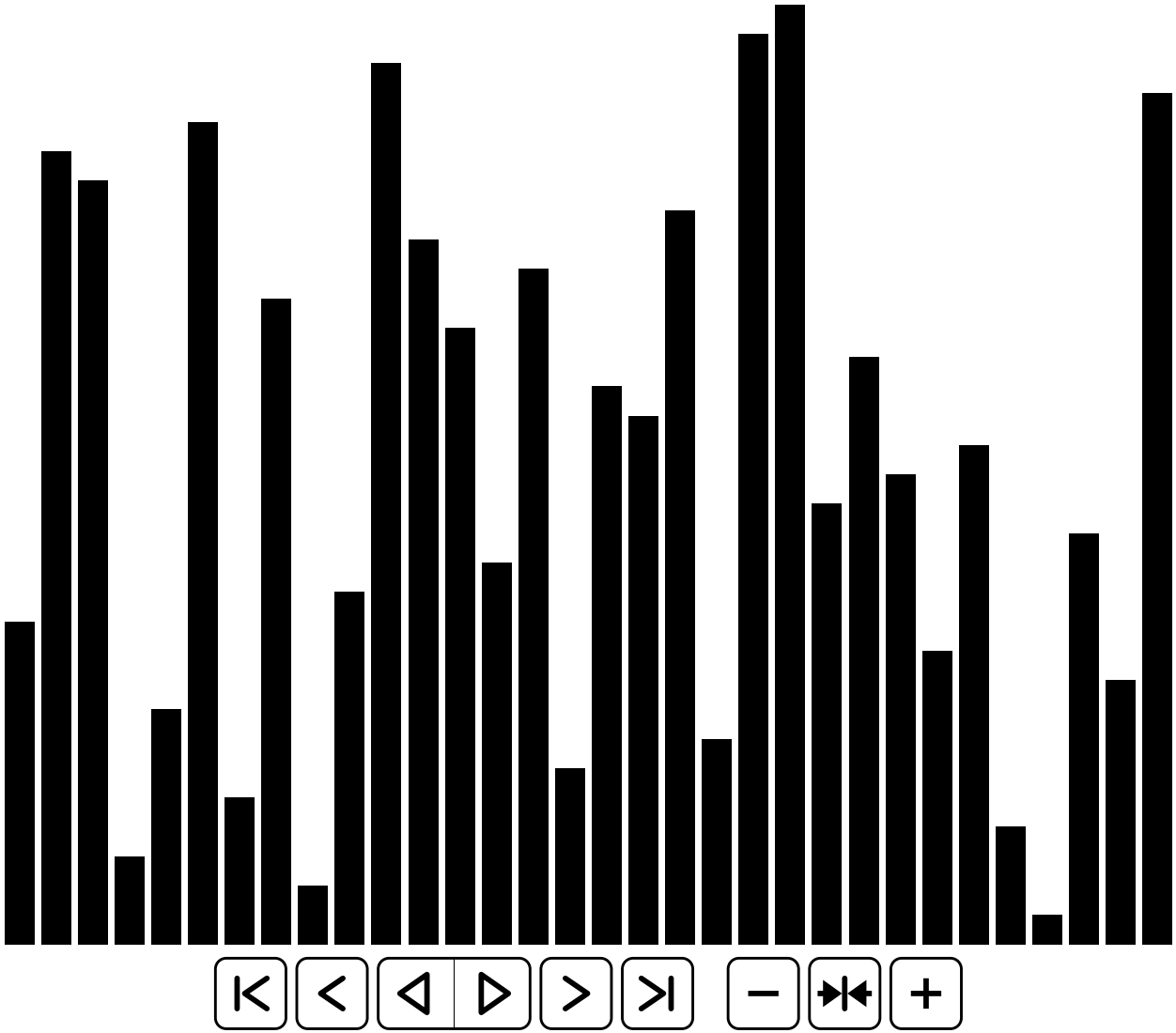
QucikSort – Třídění rozdělováním (pokrač.)

```
20        QuickSortInternal(a, l, j);
21    if (i < r)
22        QuickSortInternal(a, i, r);
23 }
```

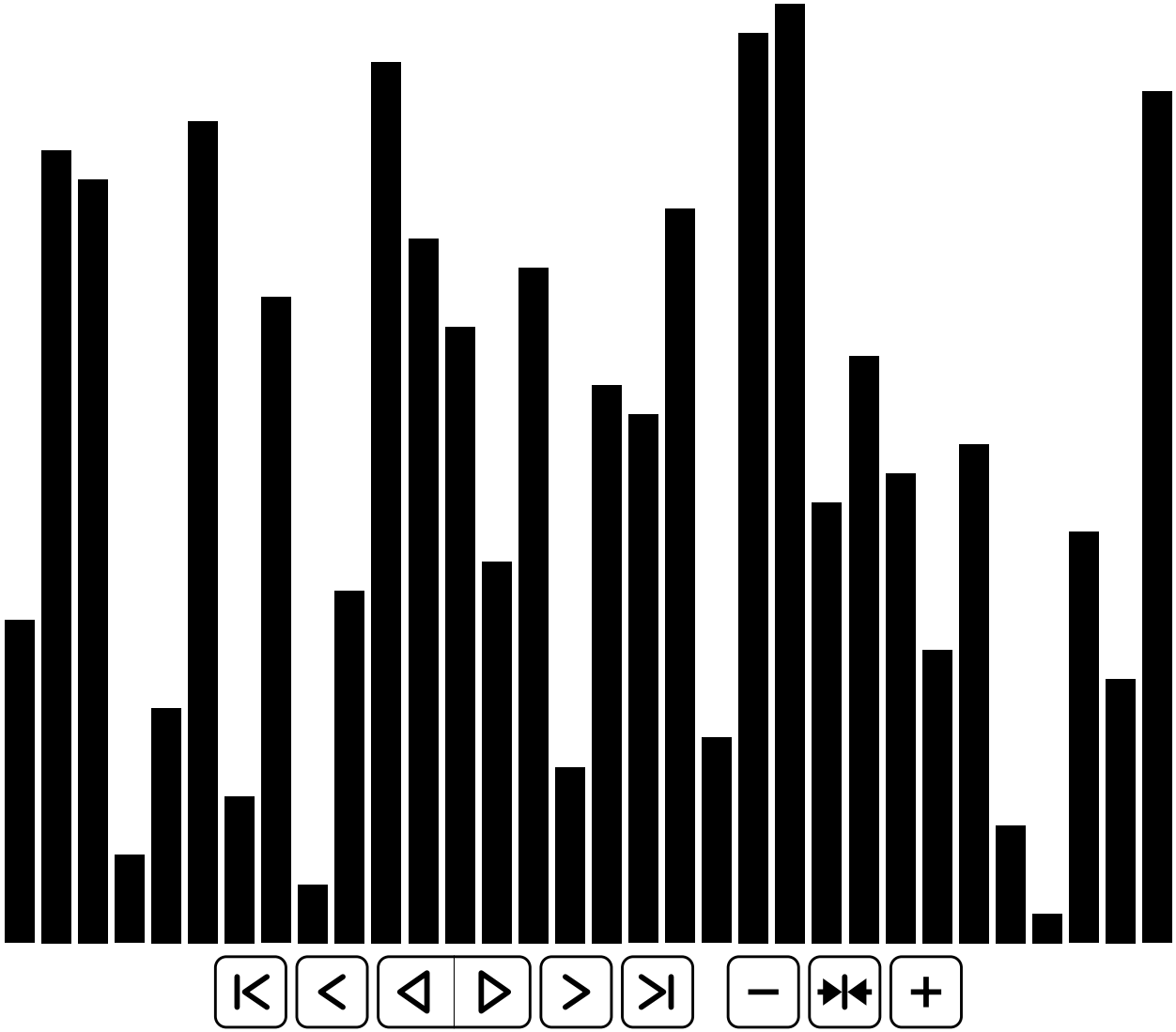
QucikSort – Třídění rozdělováním

```
1 void QuickSort(int a[], const int n)
2 {
3    QuickSortInternal(a, 0, n - 1);
4 }
```

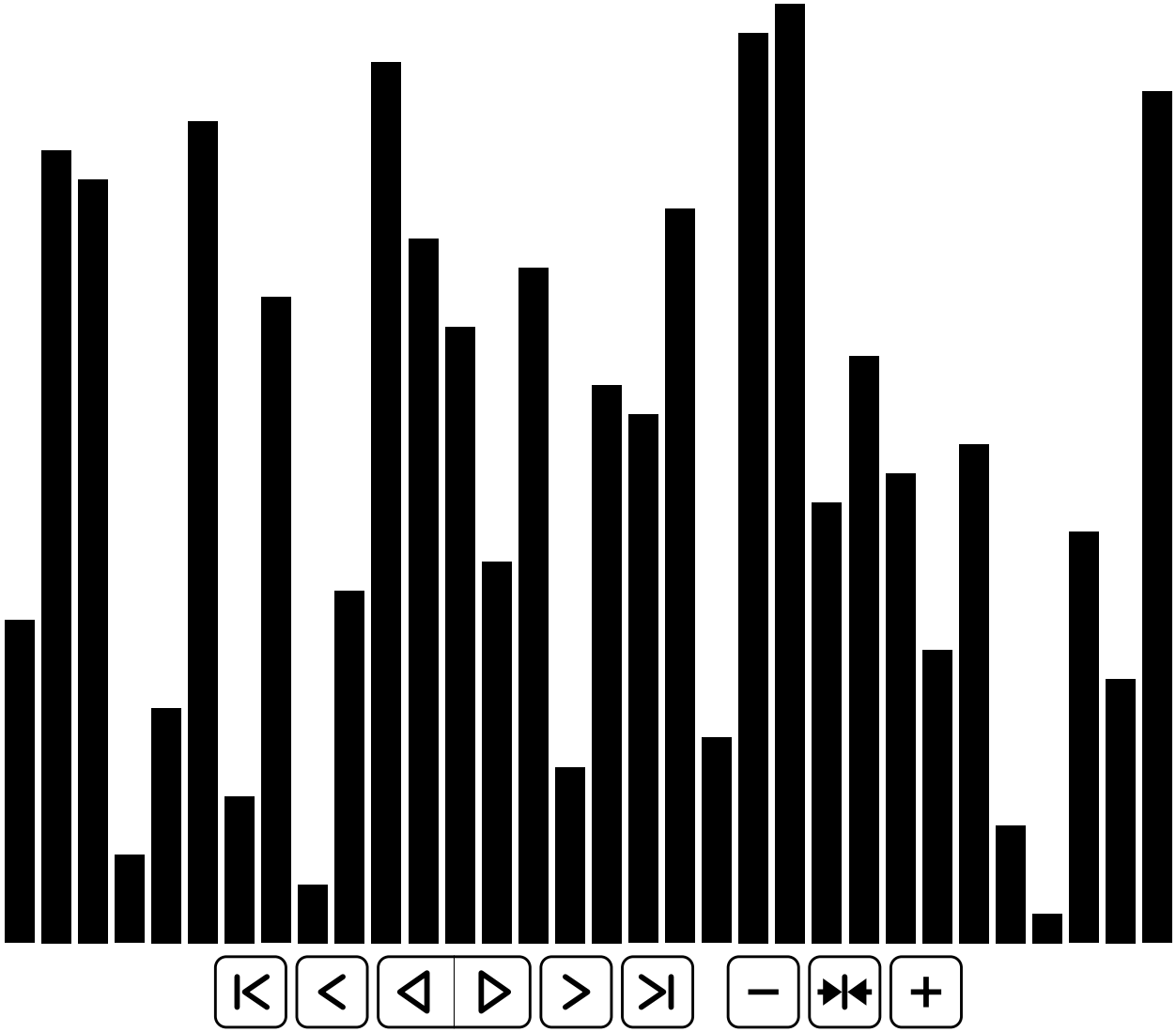
QuickSort – náhodná permutace, pivot prostřední prvek úseku



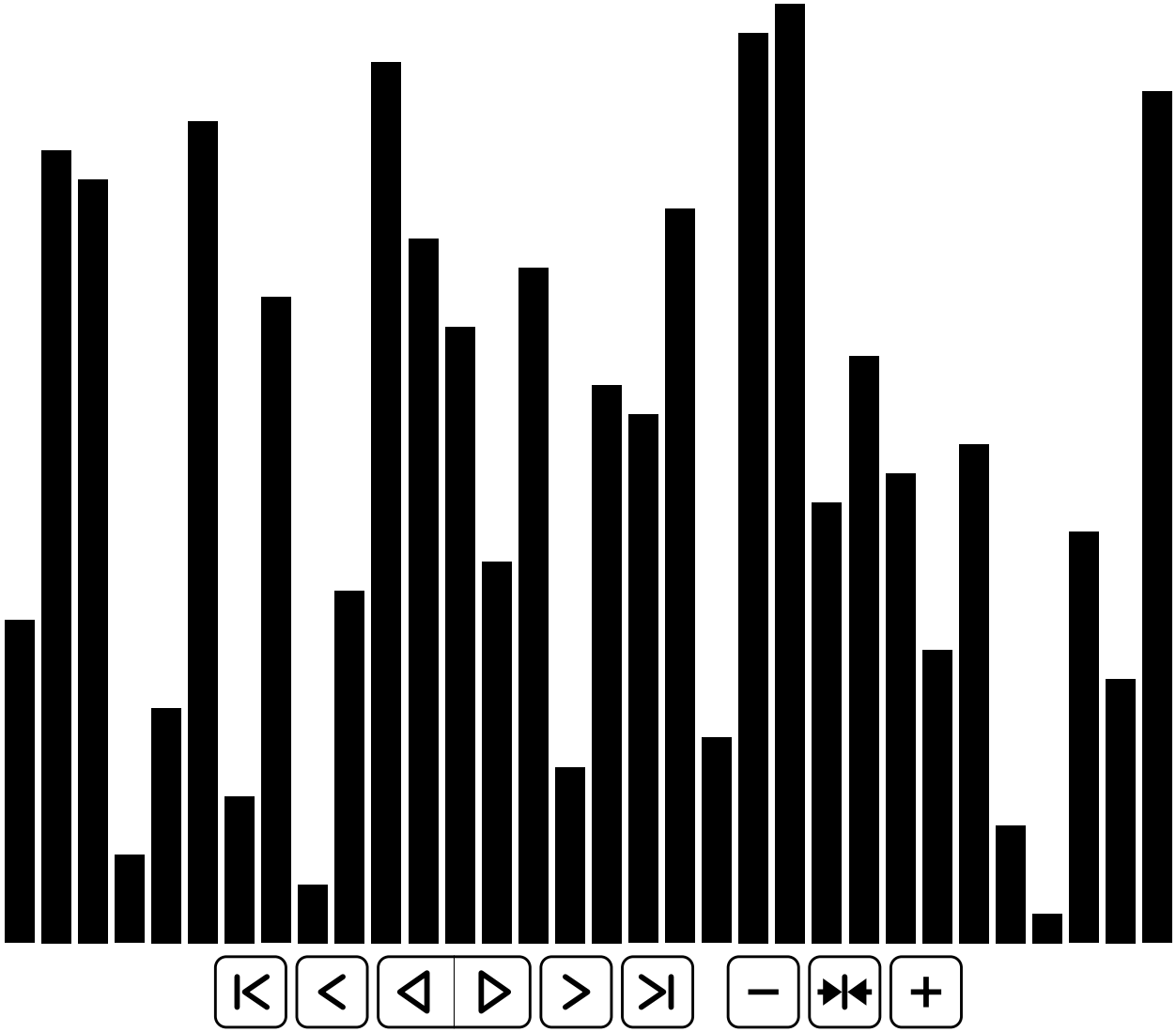
QuickSort – náhodná permutace, pivot medián úseku



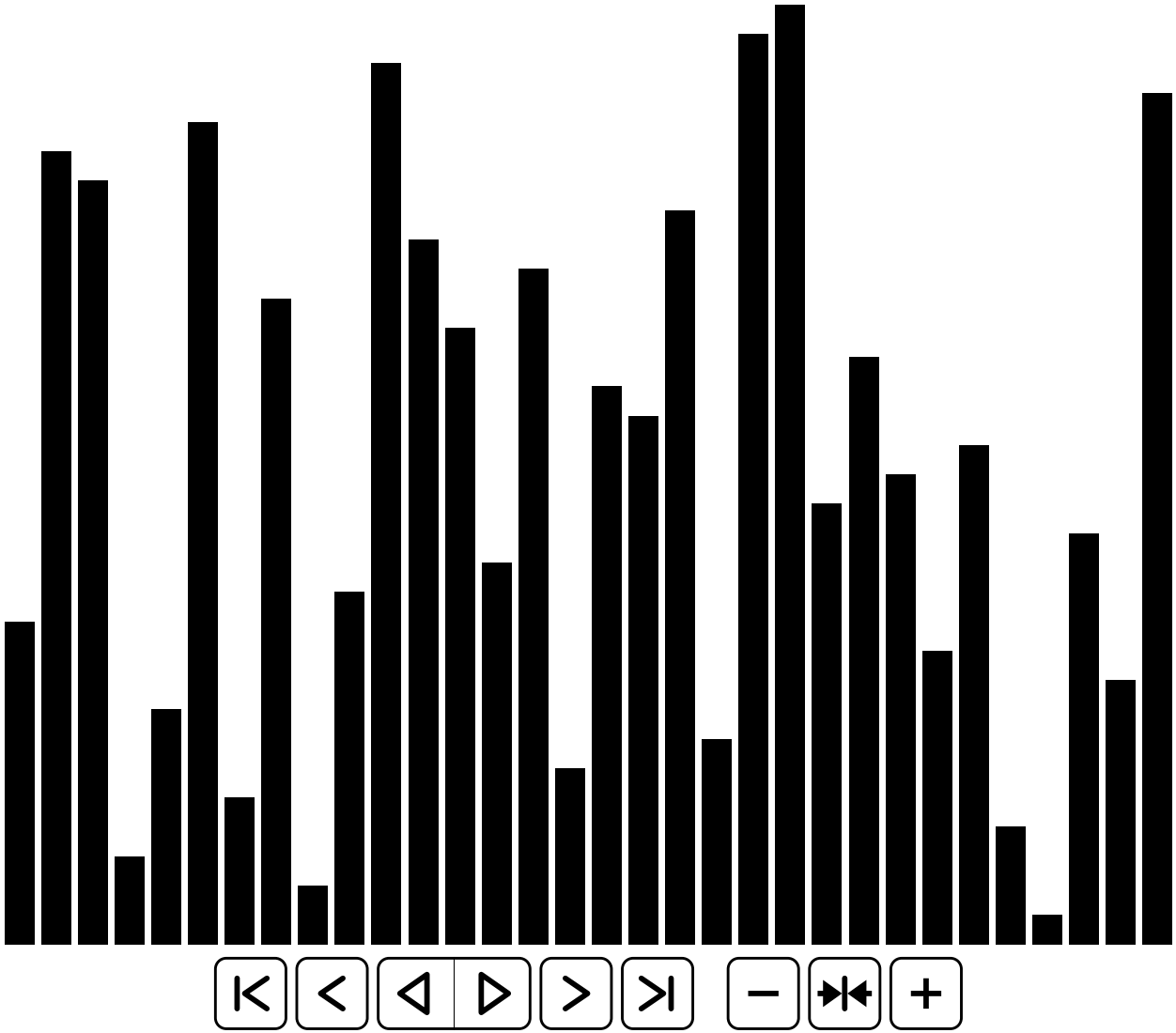
QuickSort – náhodná permutace, pivot minimum úseku



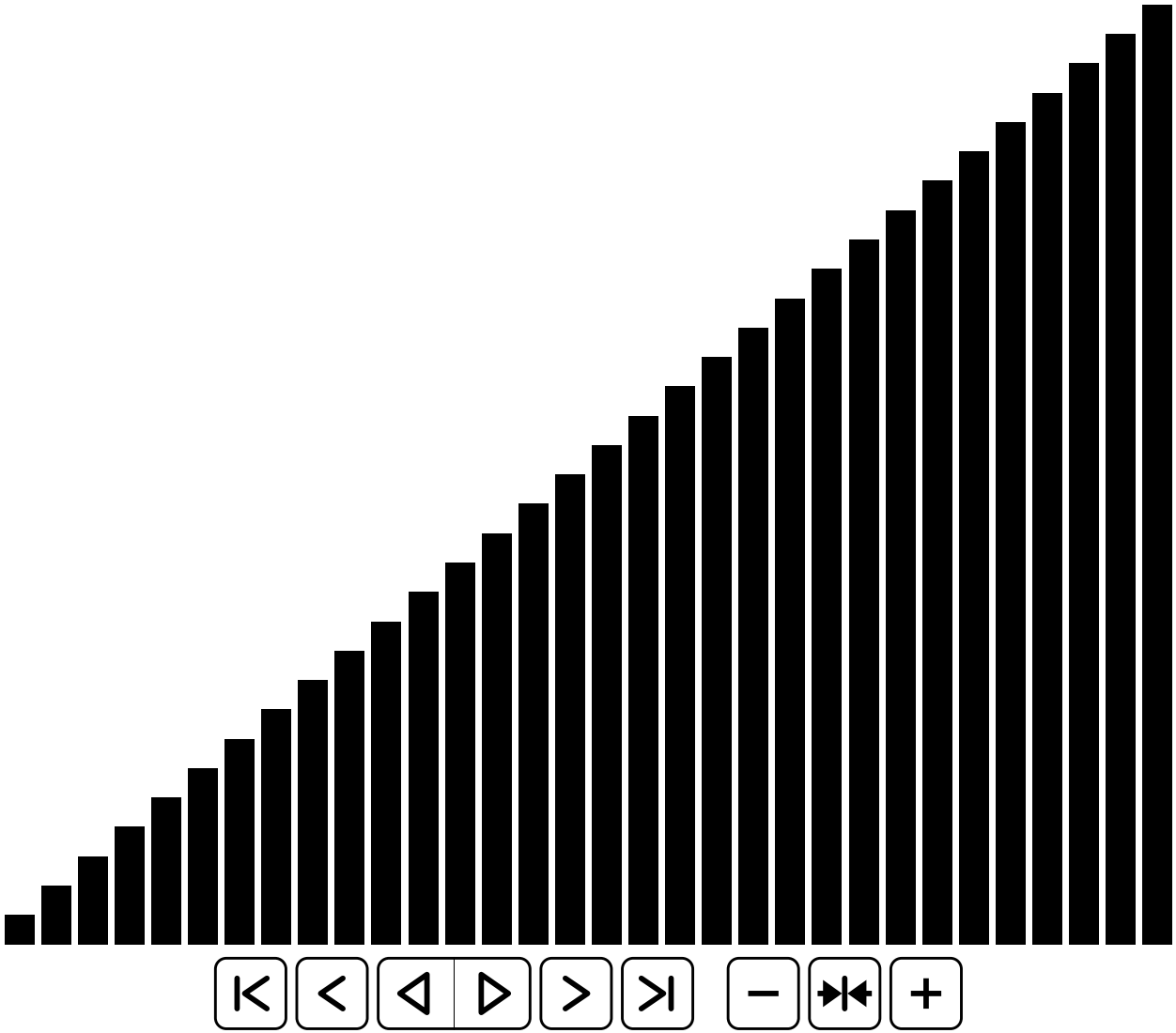
QuickSort – náhodná permutace, pivot levý prvek úseku



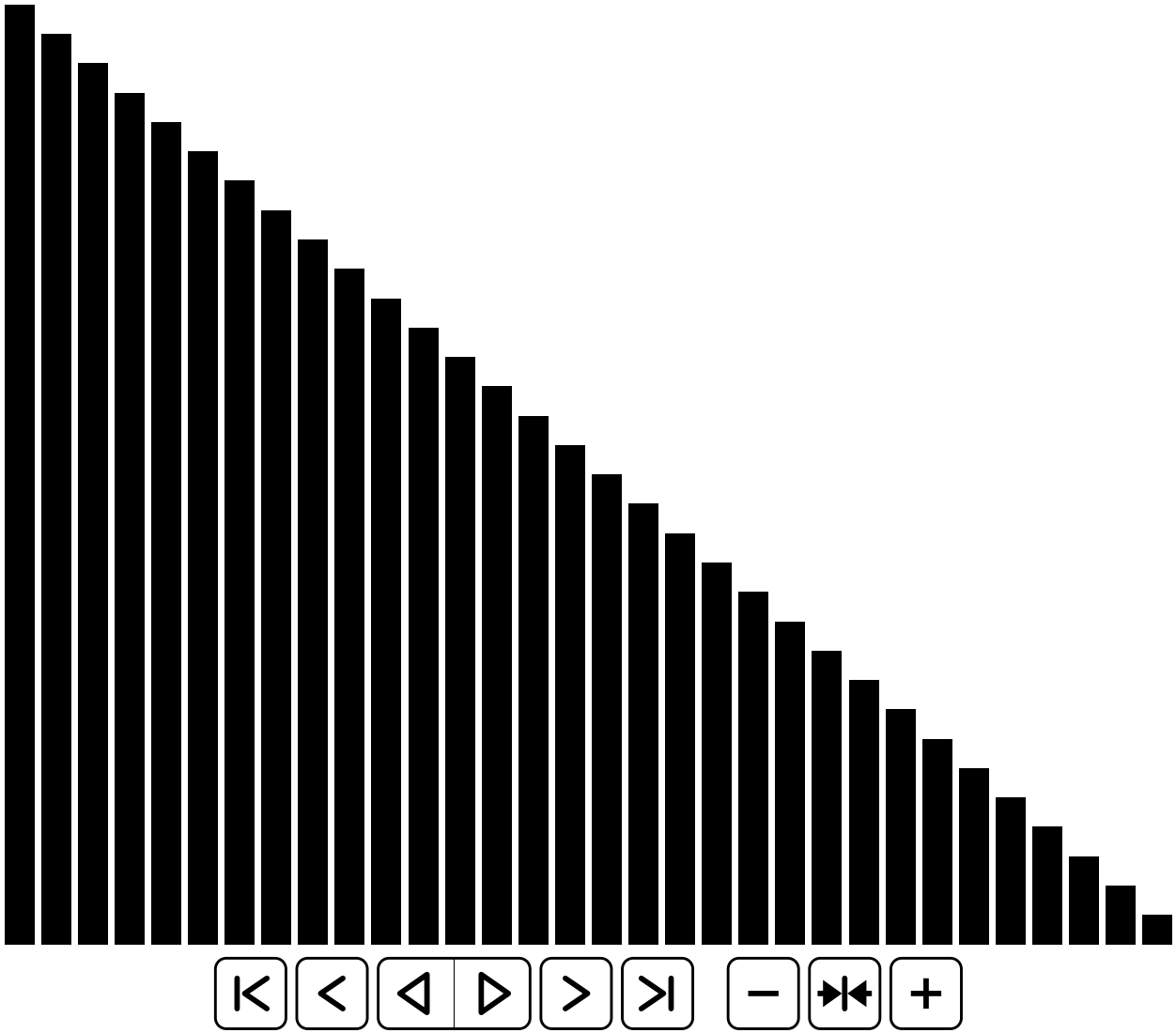
QuickSort – náhodná permutace, pivot náhodný z prvek úseku



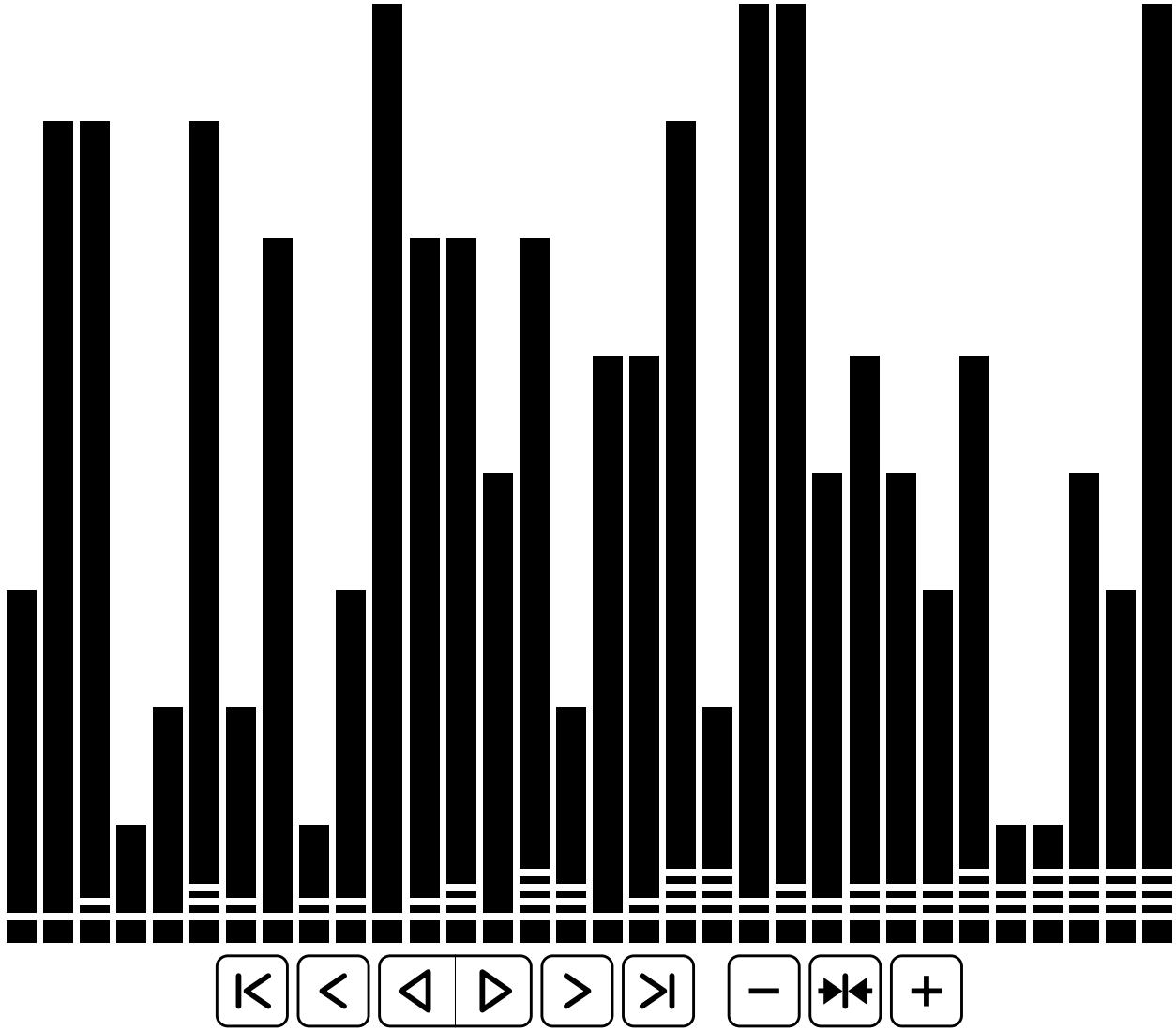
QuickSort – identická permutace, pivot prostřední prvek úseku



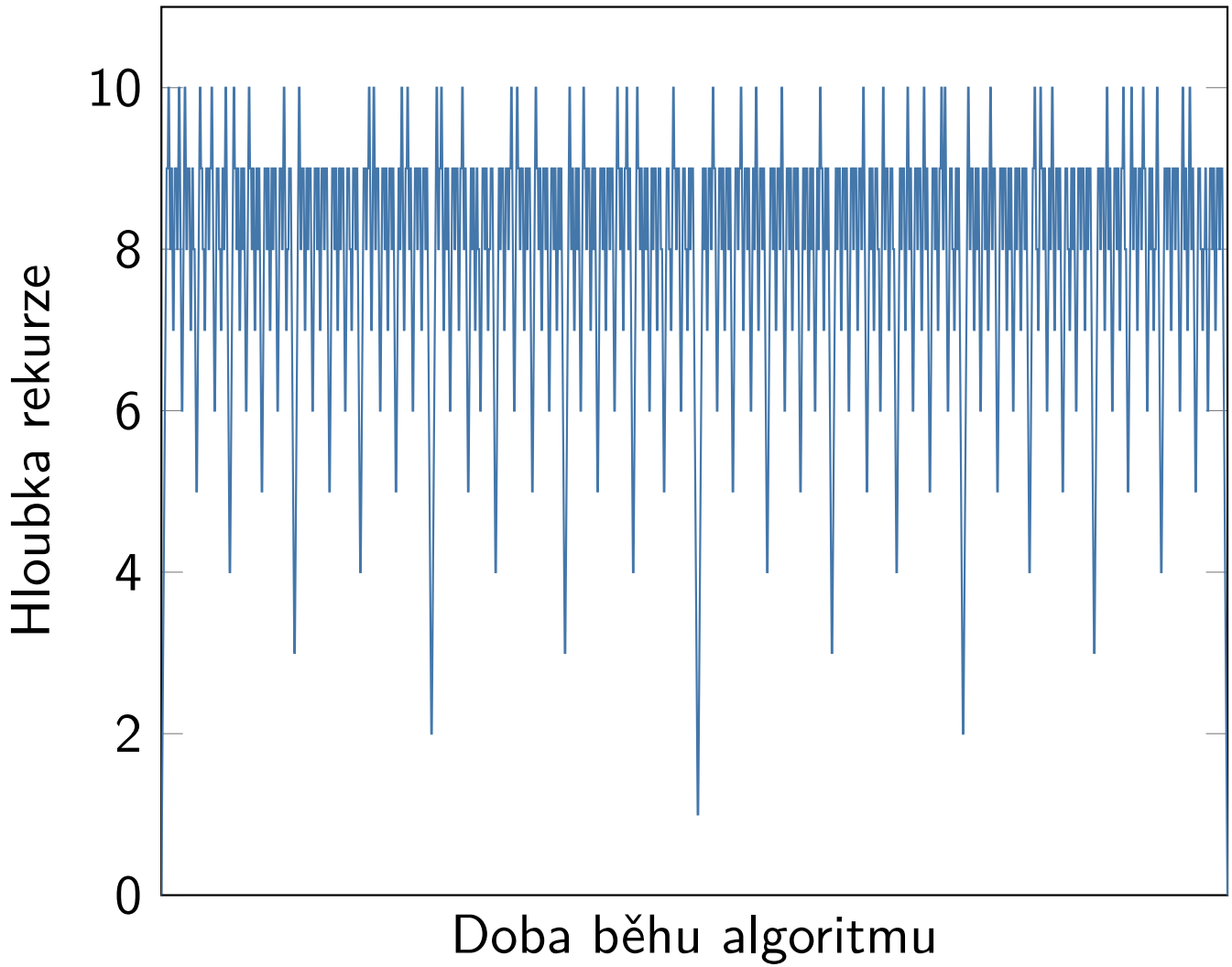
QuickSort – opačná permutace, pivot prostřední prvek úseku



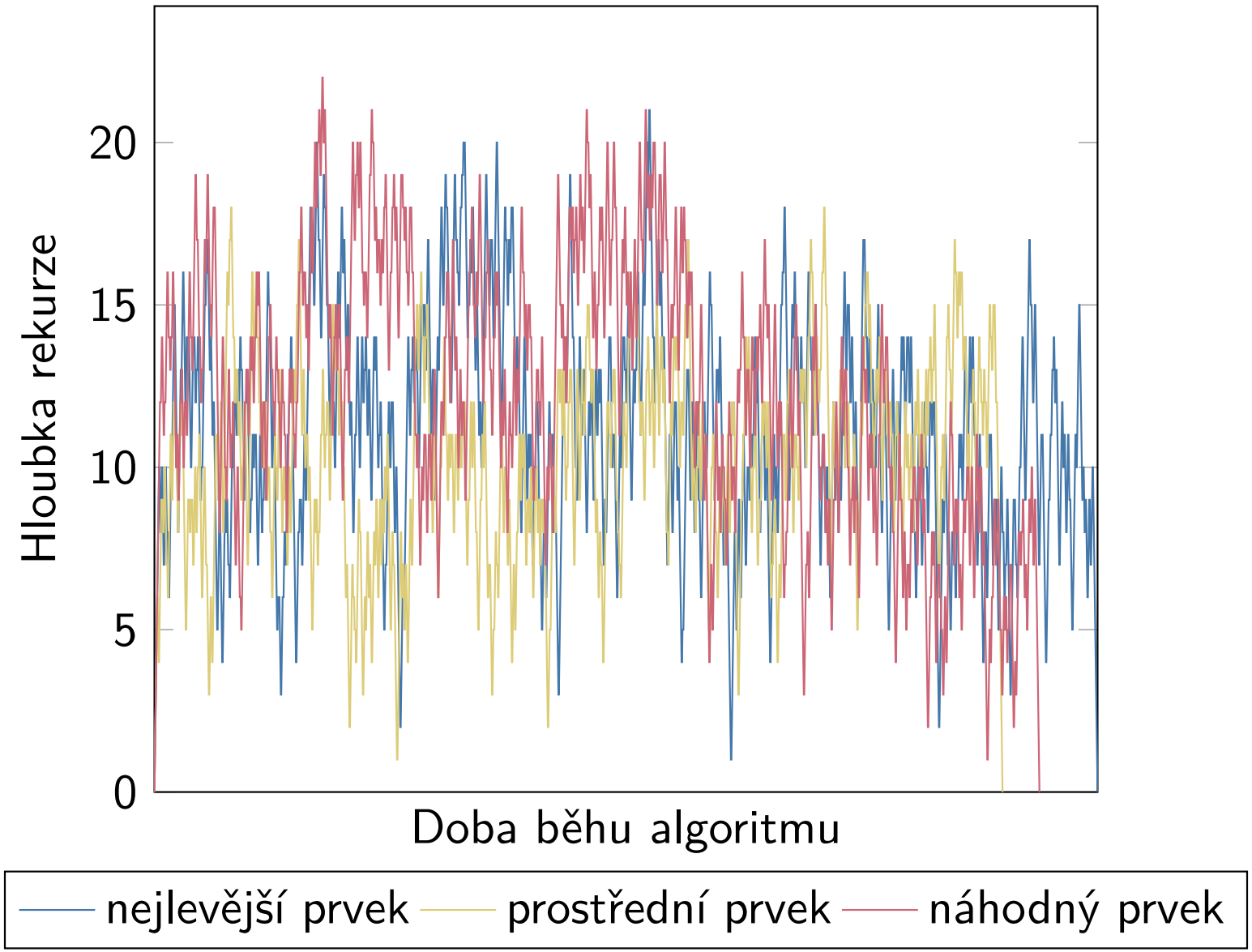
QuickSort – permutace pouze několika hodnot



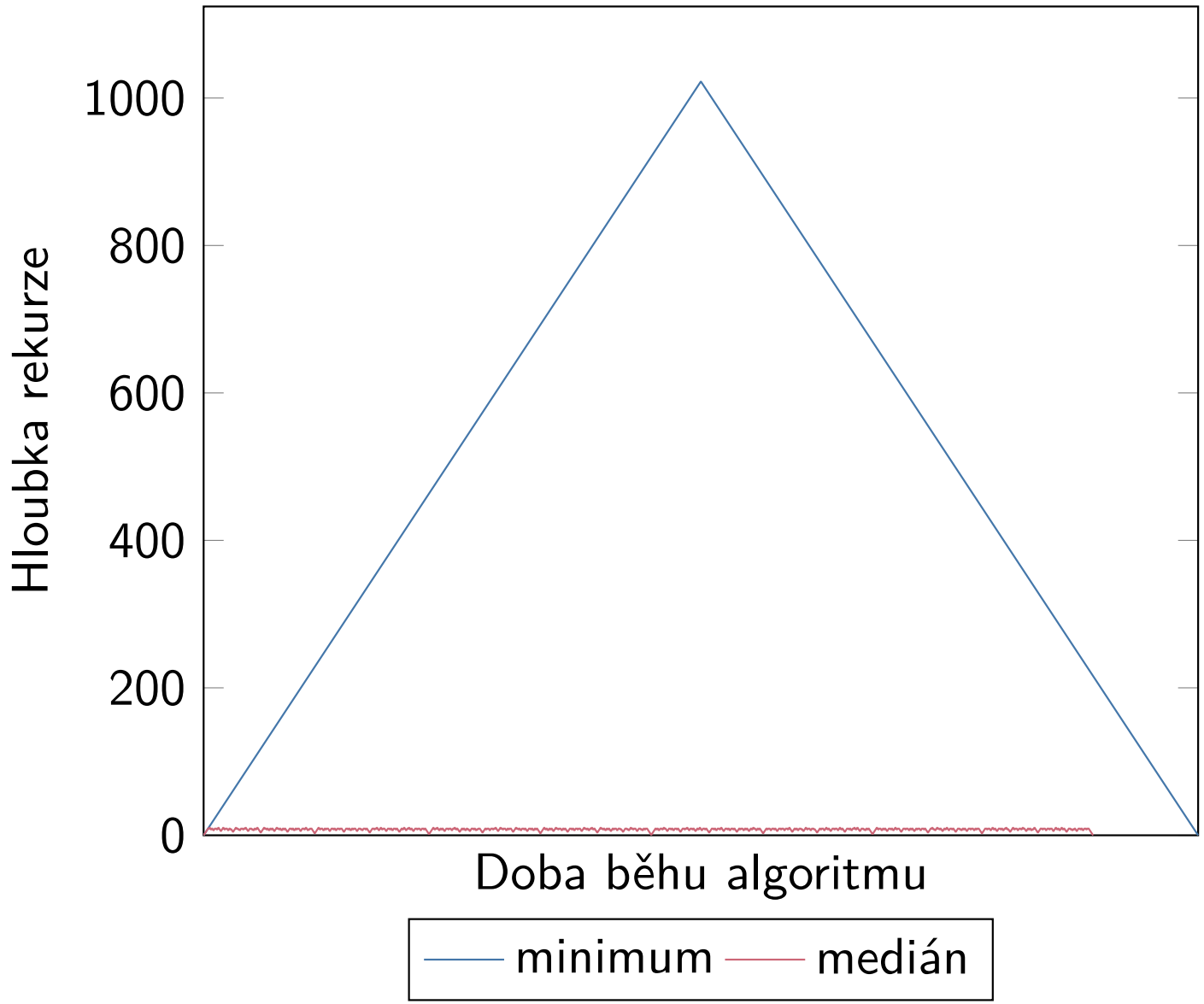
Hloubka rekurze v závislosti na volbě pivota – nejlepší případ, medián



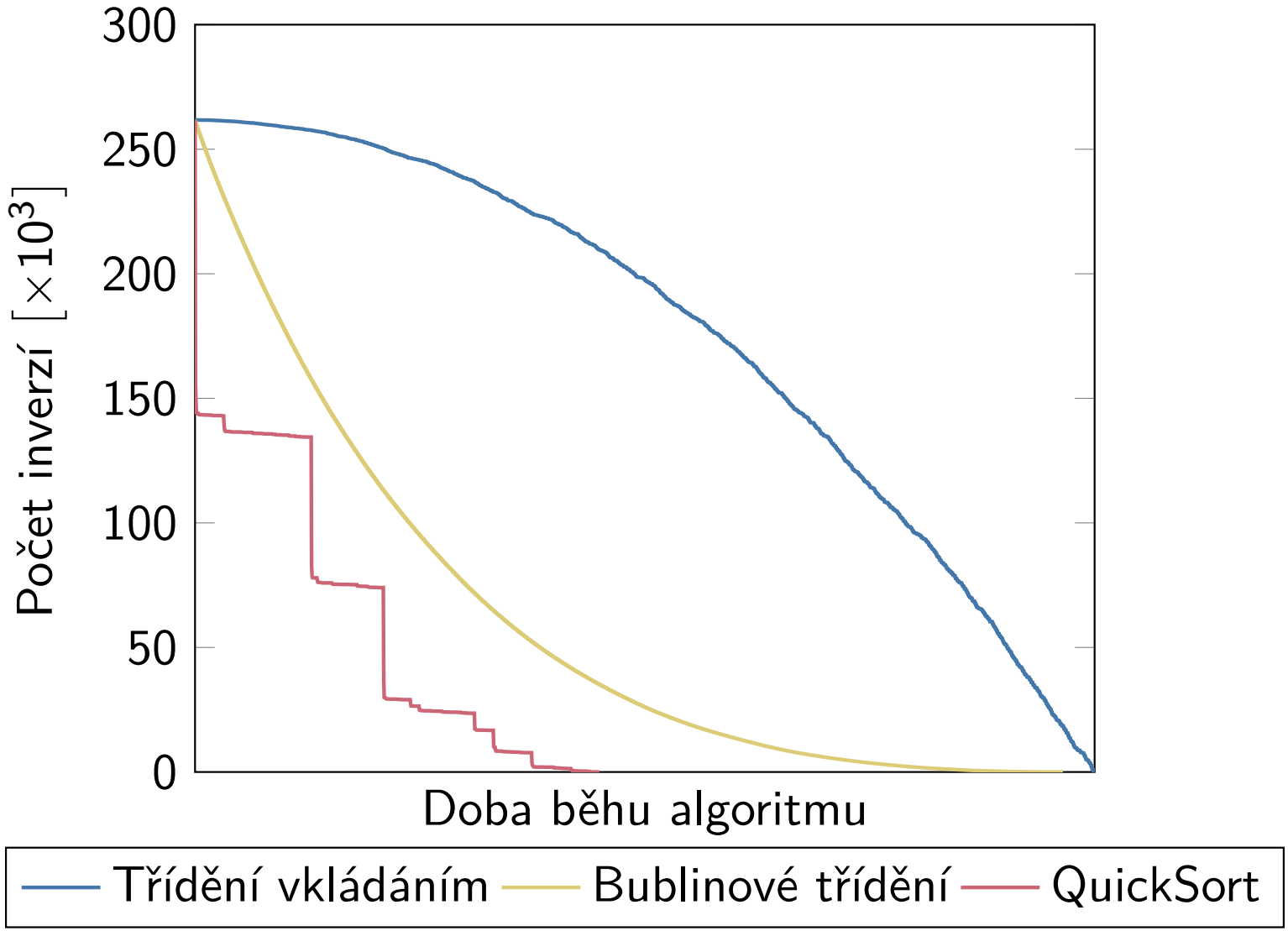
Hloubka rekurze v závislosti na volbě pivota



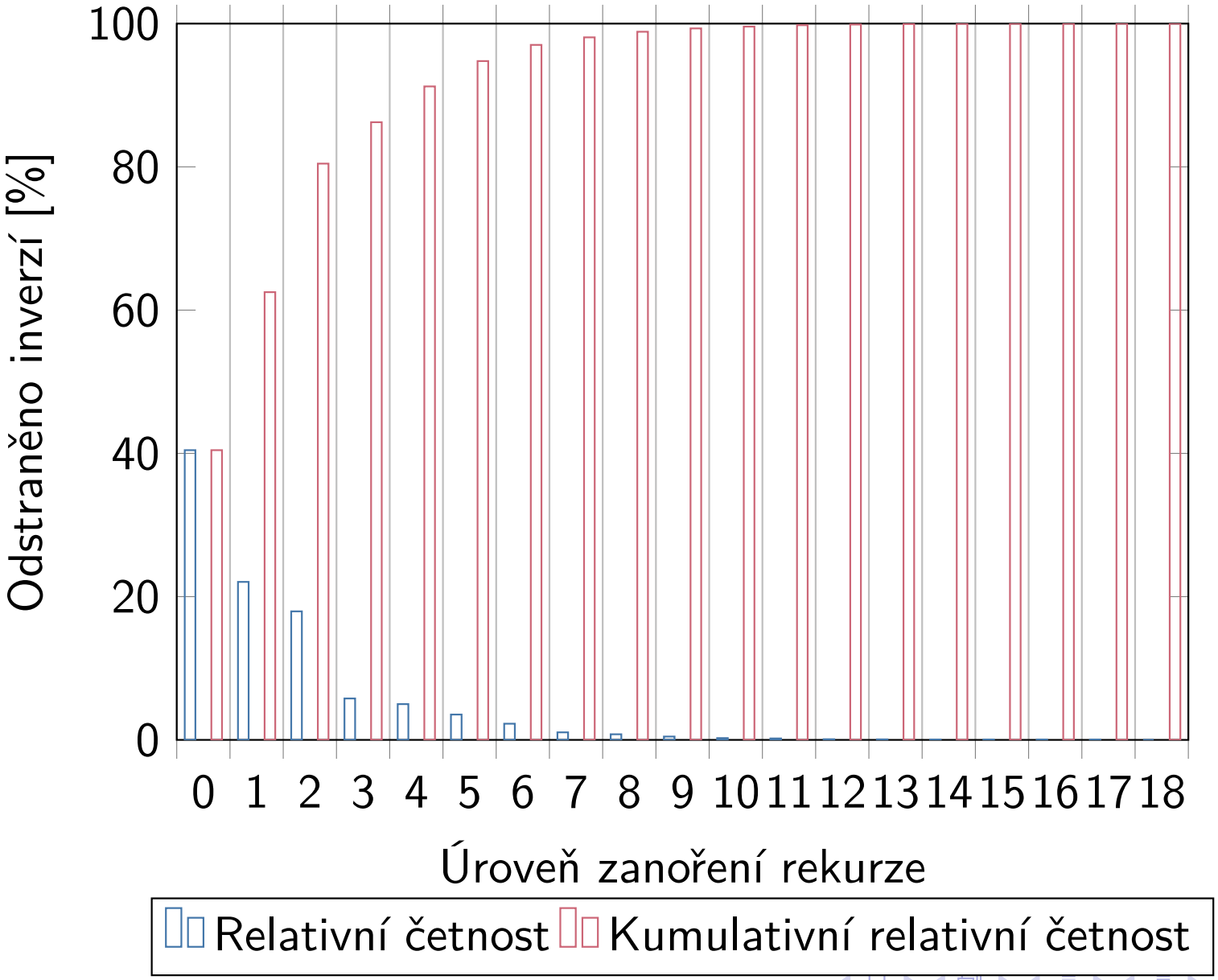
Hloubka rekurze v závislosti na volbě pivota – extrémy



Průběh odstraňování inverzí v čase



QuickSort – průběh odstraňování inverzí



Třídění haldou – HeapSort

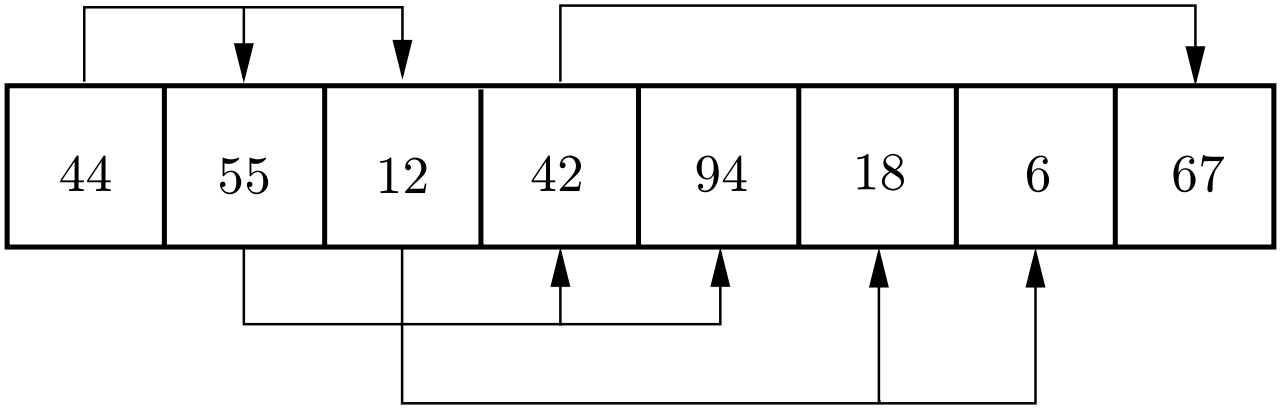
- lze říci, že zlepšuje třídění výběrem,
- efektivně řeší problém výběru i-tého největšího prvku,
- pro $i = 1$ je jasné, že potřebujeme $n - 1$ porovnání,
- ale pro další i by bylo vhodné si nějakým způsobem „zapamatovat“ výsledky předchozích porovnání,
- tomuto účelu nejlépe vyhovuje datová struktura **halda** (angl. heap).

Halda (Heap)

Halda reprezentující posloupnost S délky n, je úplný binární strom výšky $h \geq 1$ s n vrcholy, a následujícími vlastnostmi:

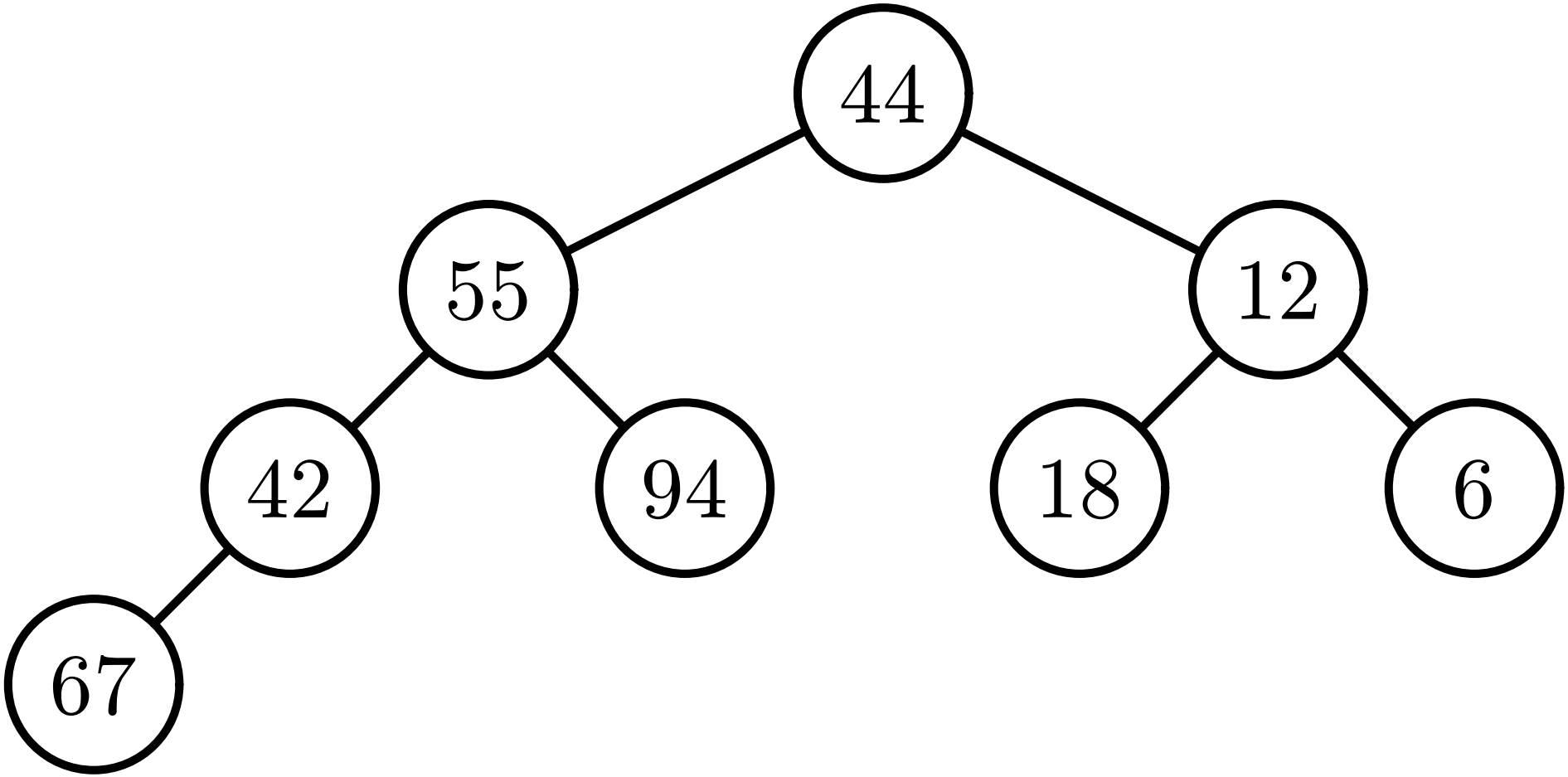
- všechny listy se nachází ve vzdálenosti h nebo $h-1$ od kořene,
- všechny listy na úrovni h jsou vlevo od listů na úrovni $h-1$,
- každému vrcholu tohoto stromu je přiřazen jeden prvek posloupnosti S tak, že všem jeho potomkům jsou přiřazeny menší prvky.

Konstrukce haldy – počáteční stav

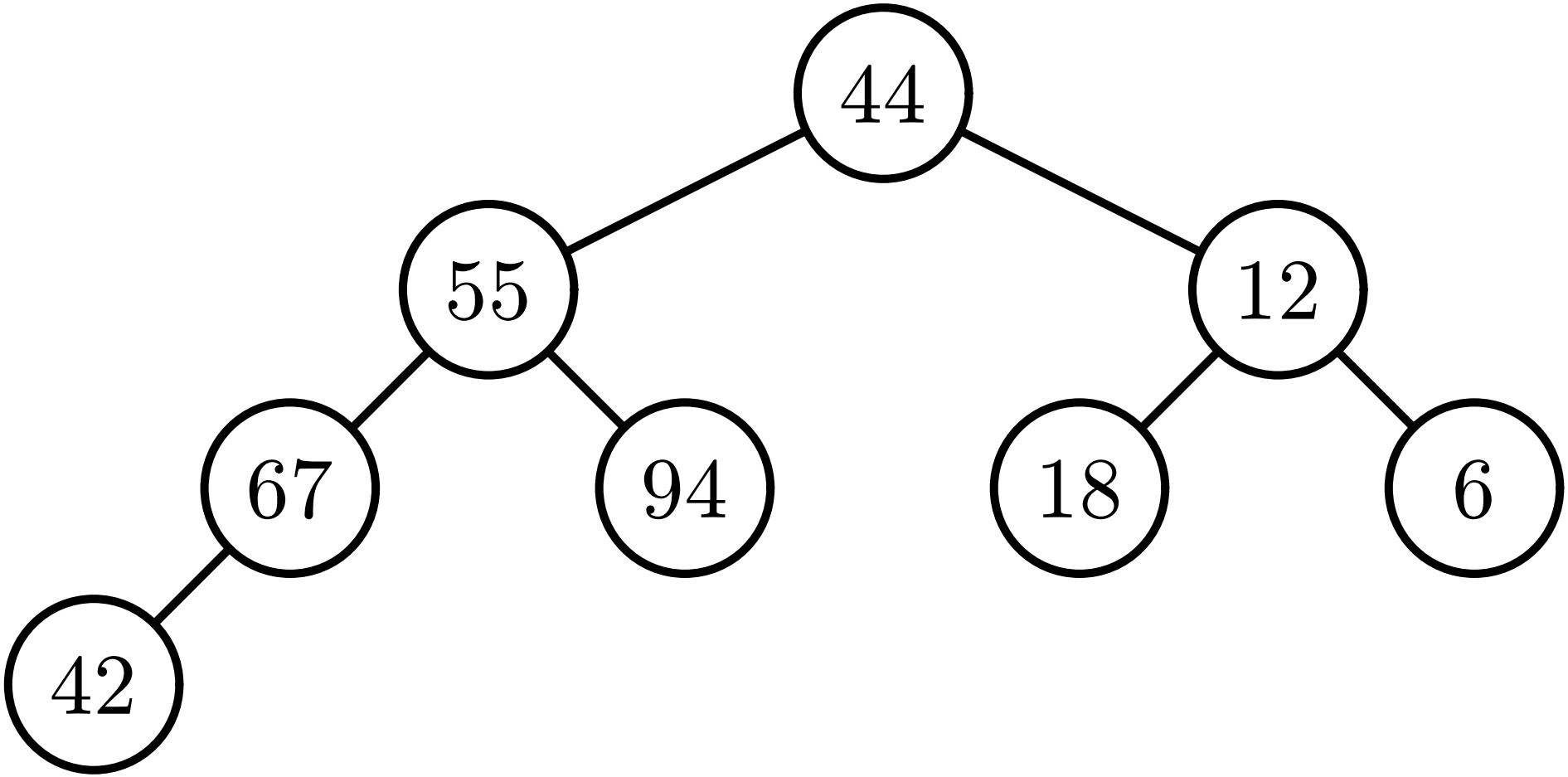


Haldu lze efektivně reprezentovat v poli tak, že leží-li rodič na indexu i, pak jeho levý potomek leží na indexu $2i$ a pravý potomek na indexu $2i + 1$.

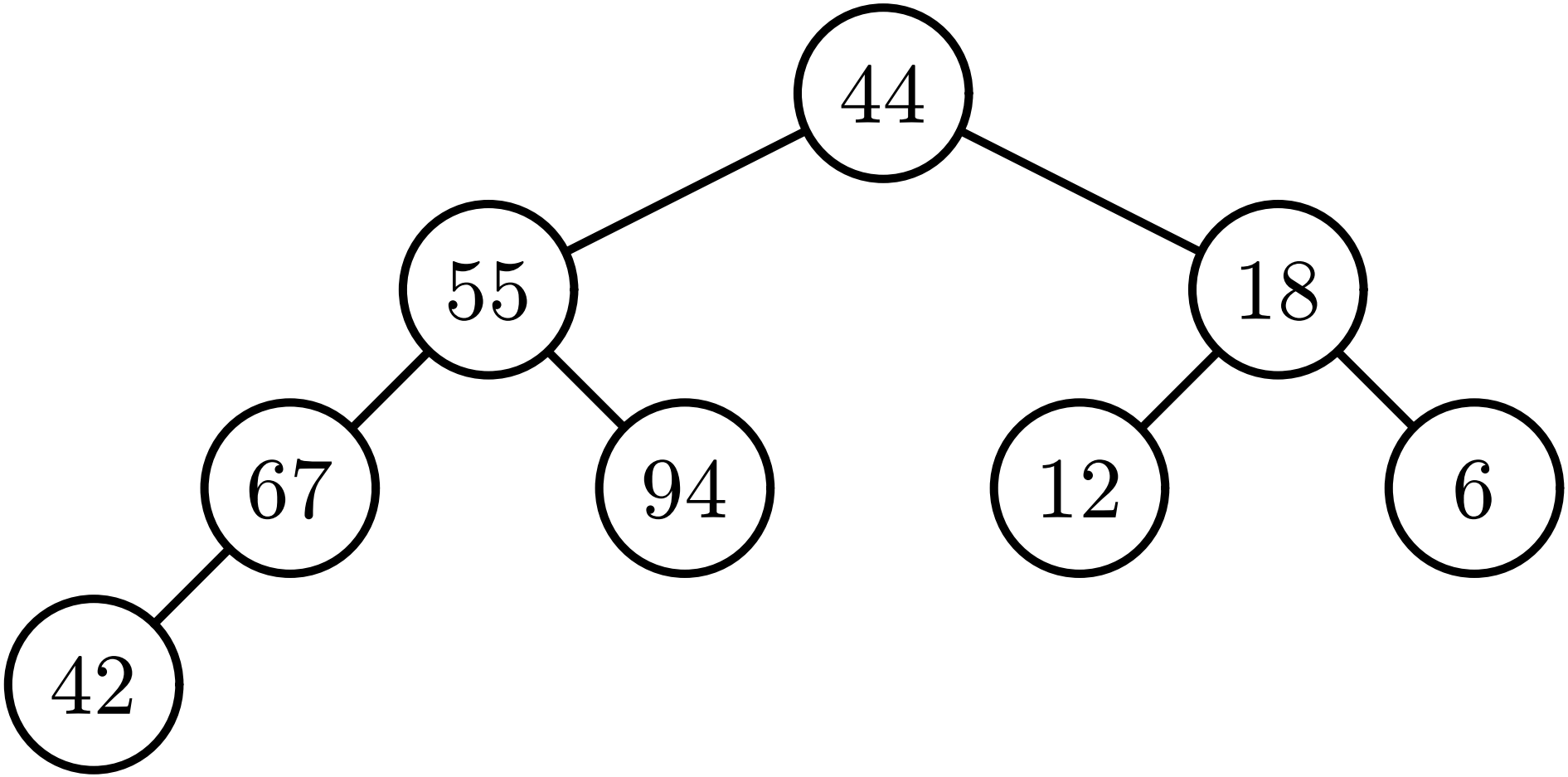
Původní halda



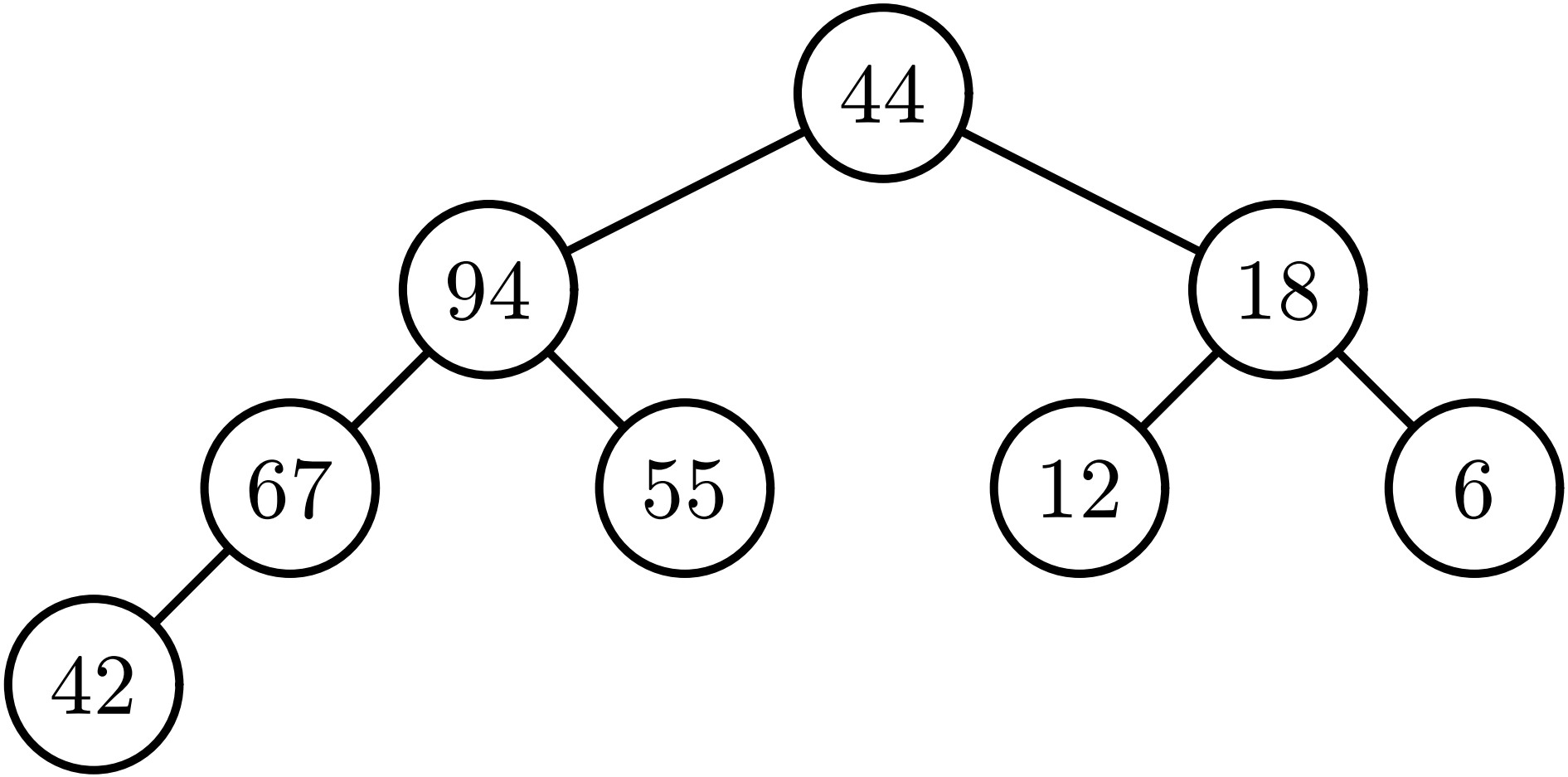
Přehozeno 42 a 67



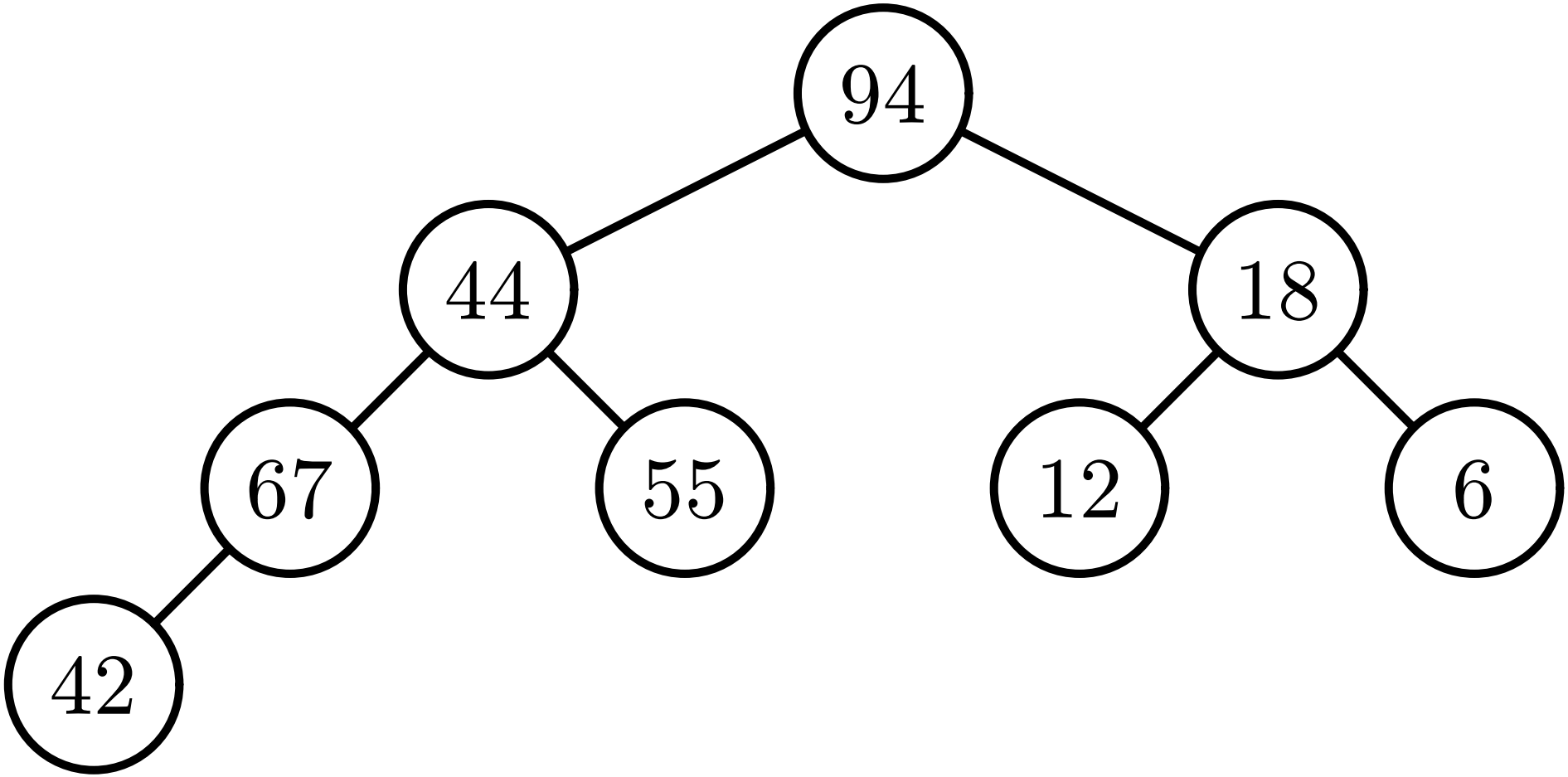
Přehozeno 18 s 12



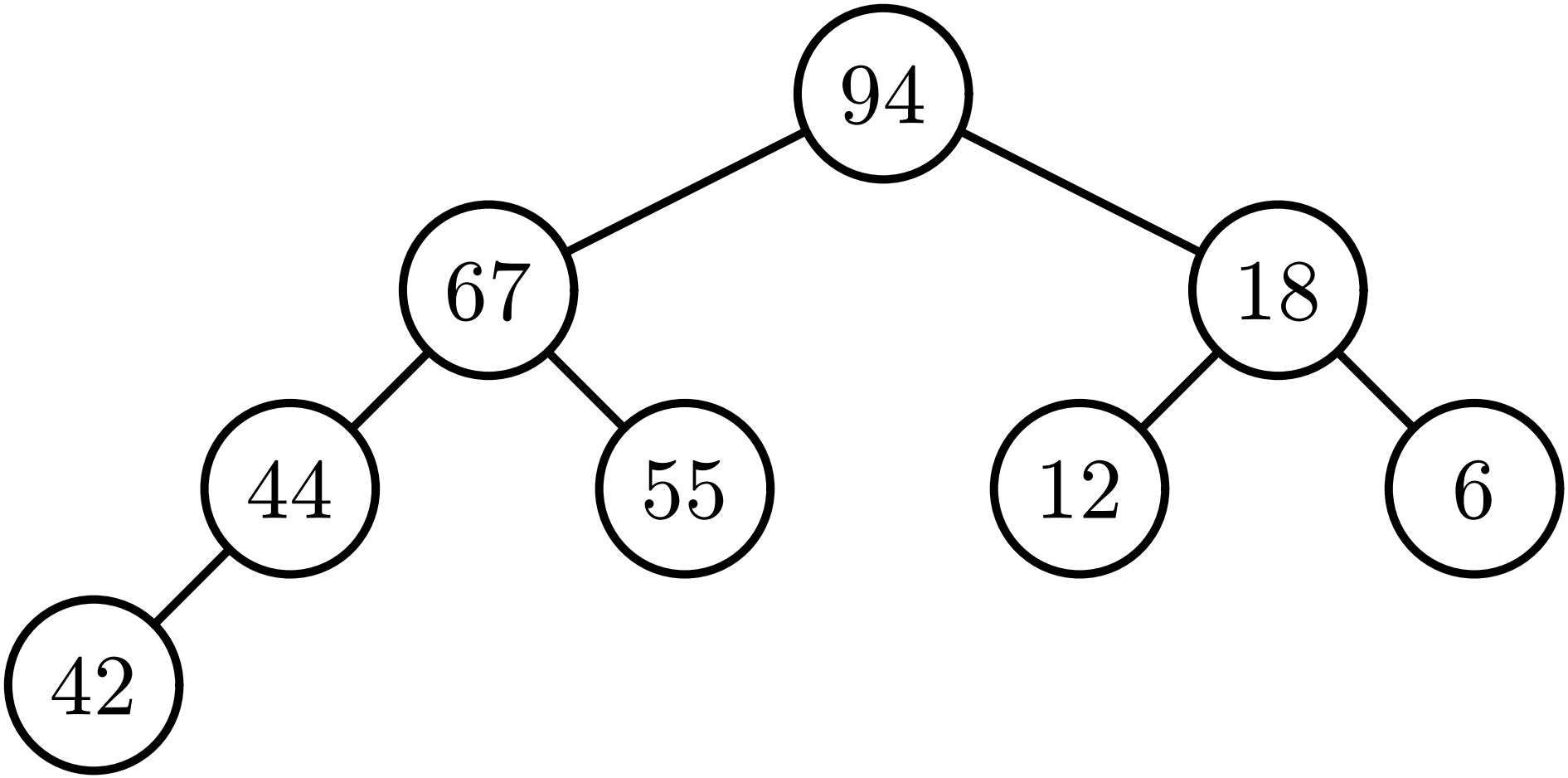
Přehozeno 55 a 94



Přehozeno 94 a 44

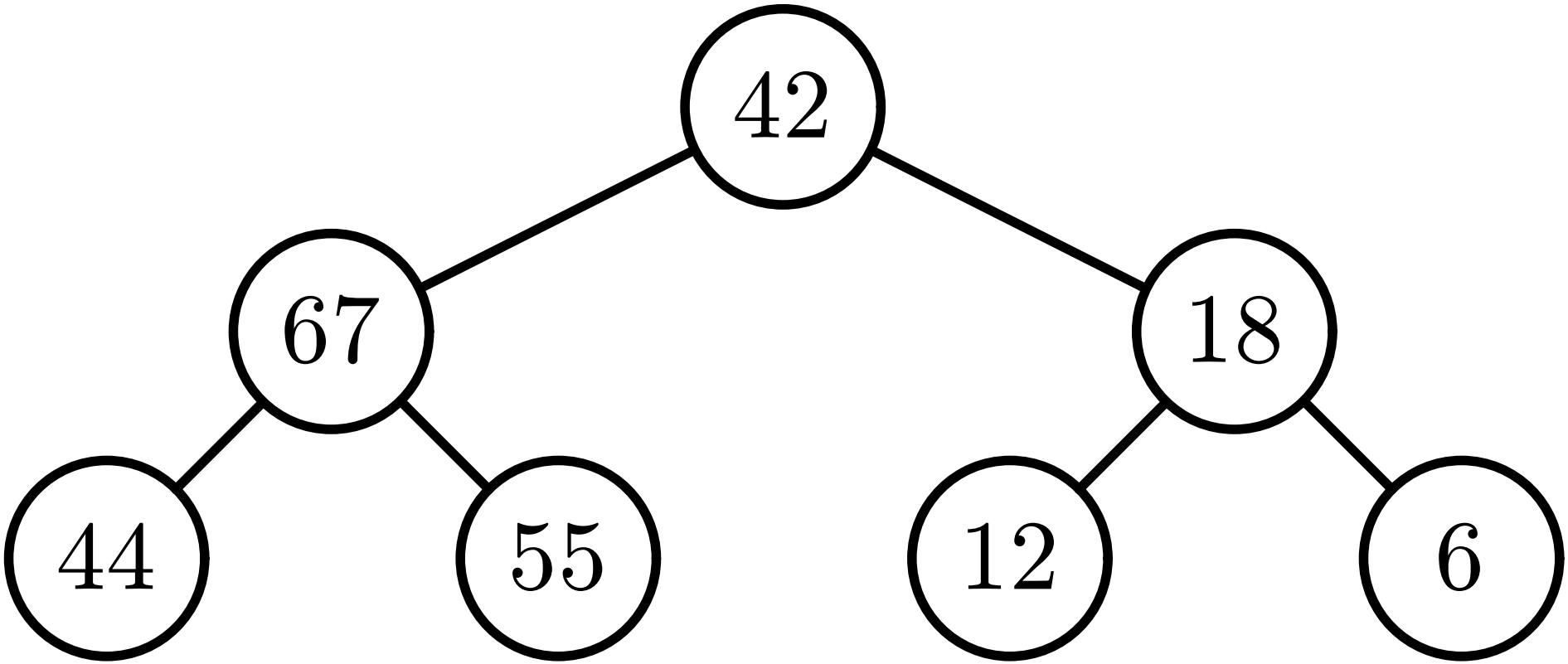


Přehozeno 44 a 67

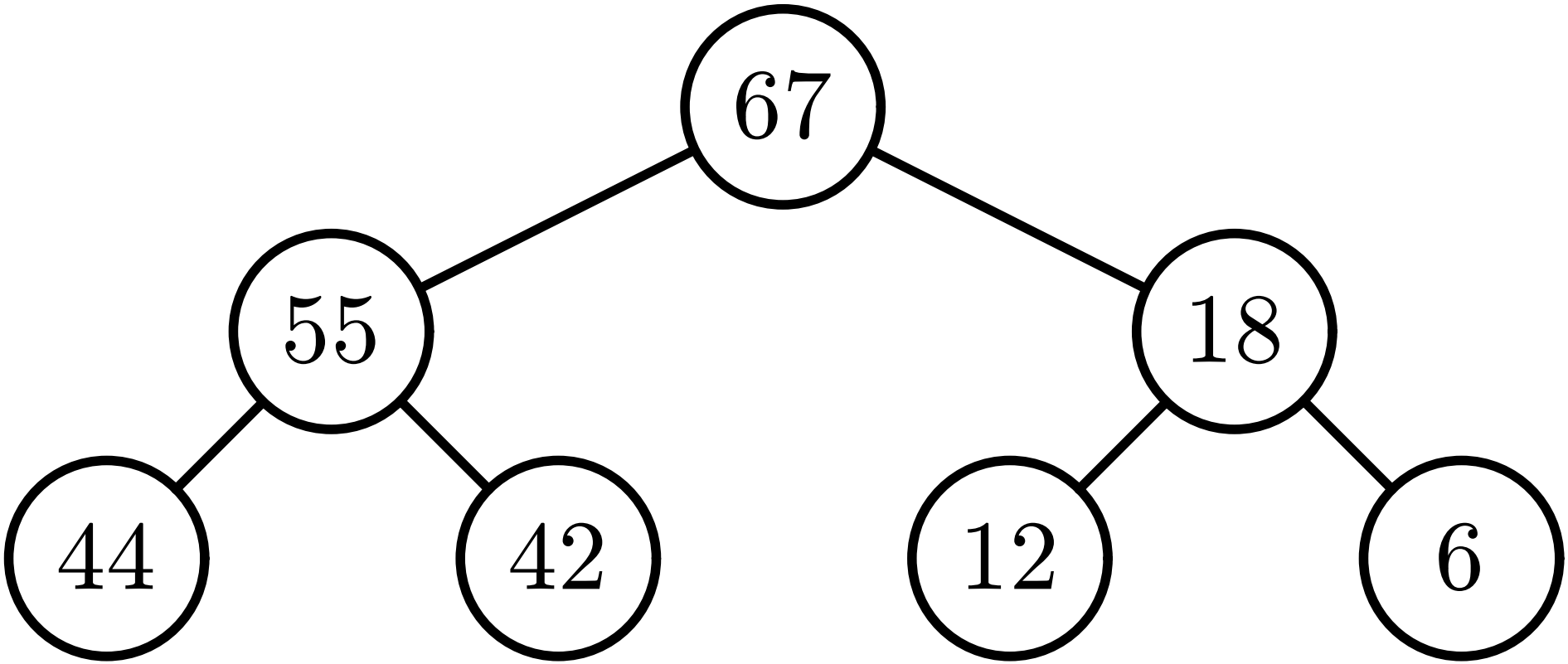


Tím je halda hotová!

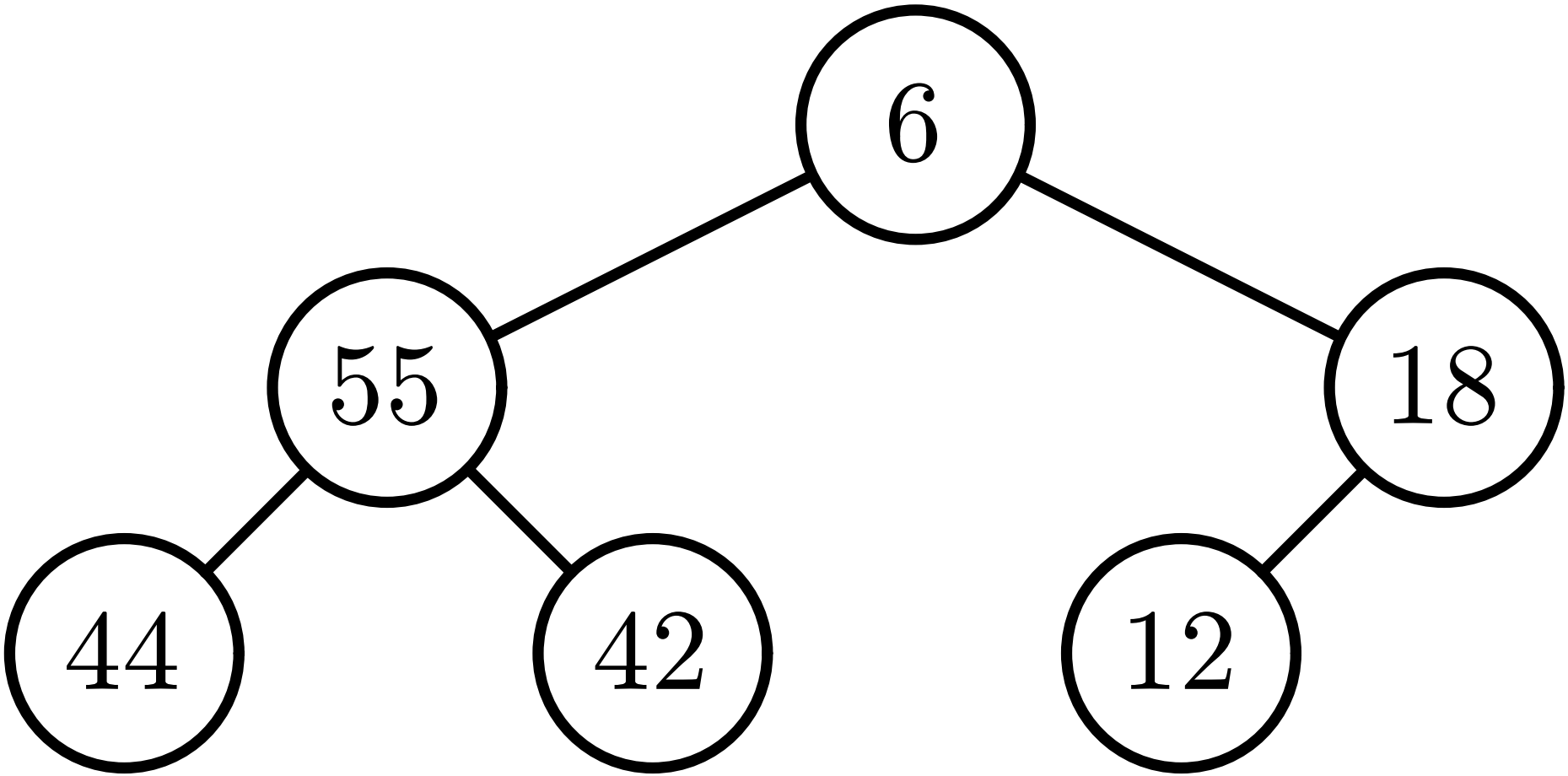
Odebrání 94



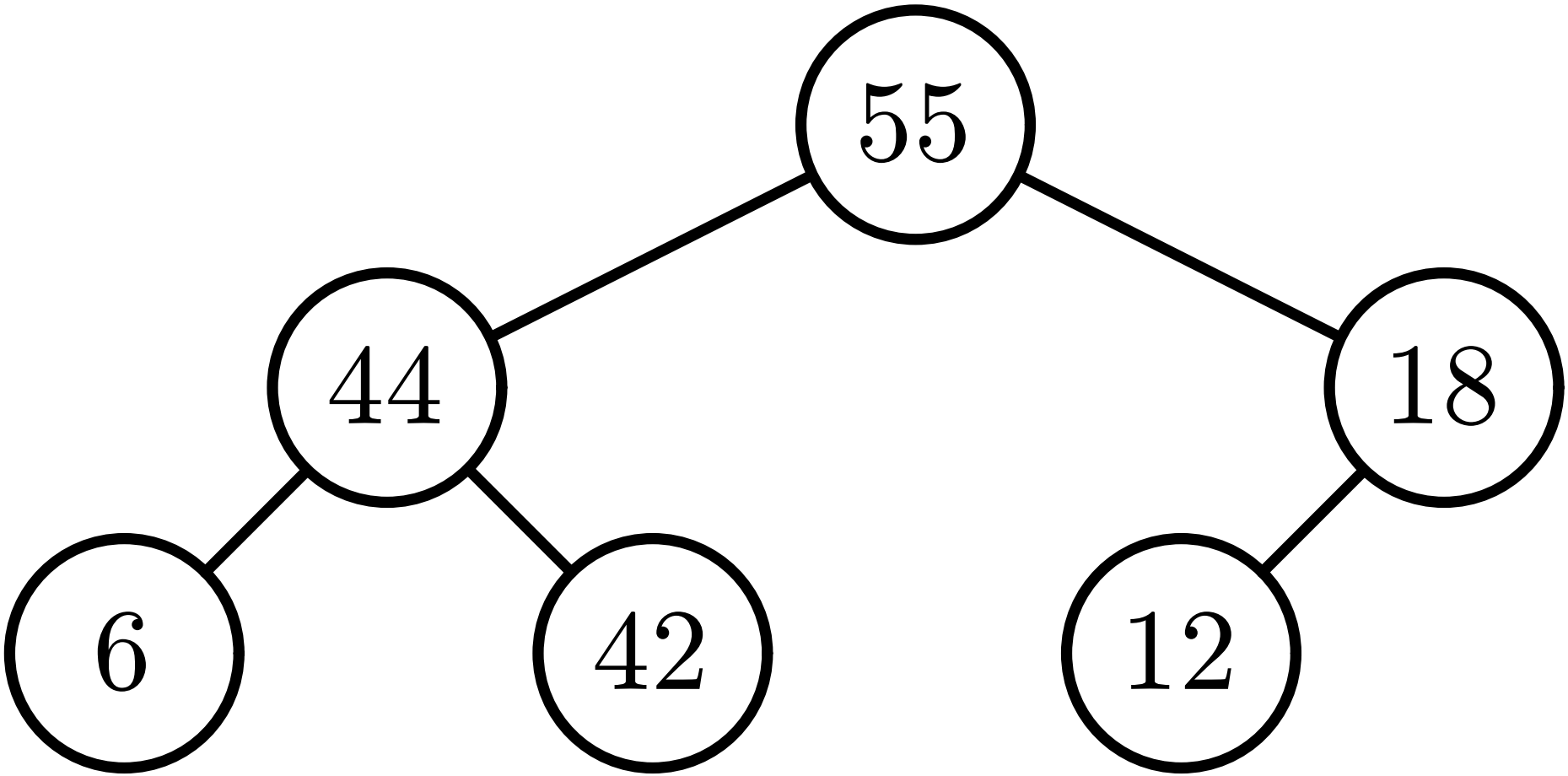
Oprava haldy



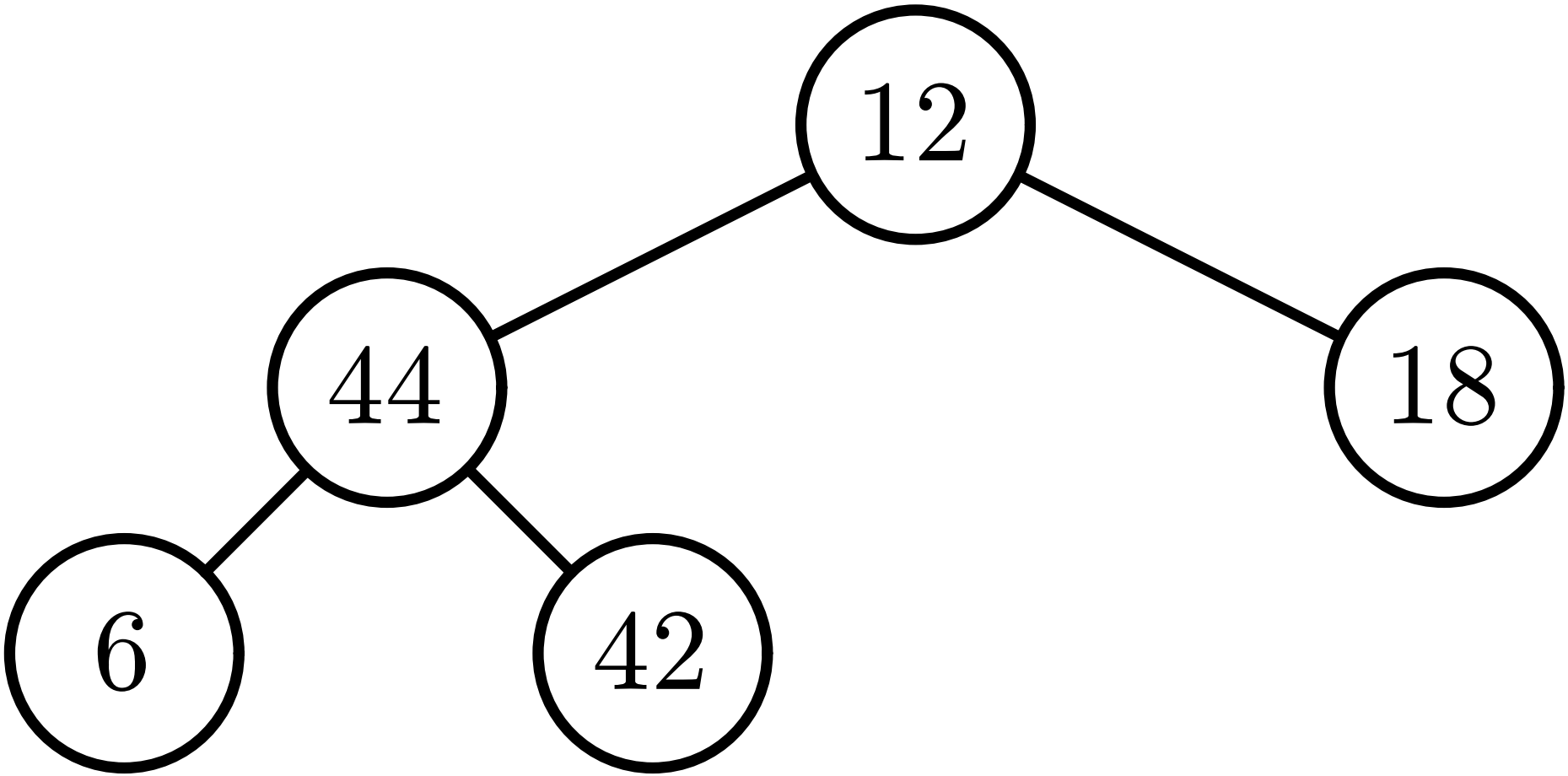
Odebrání 67



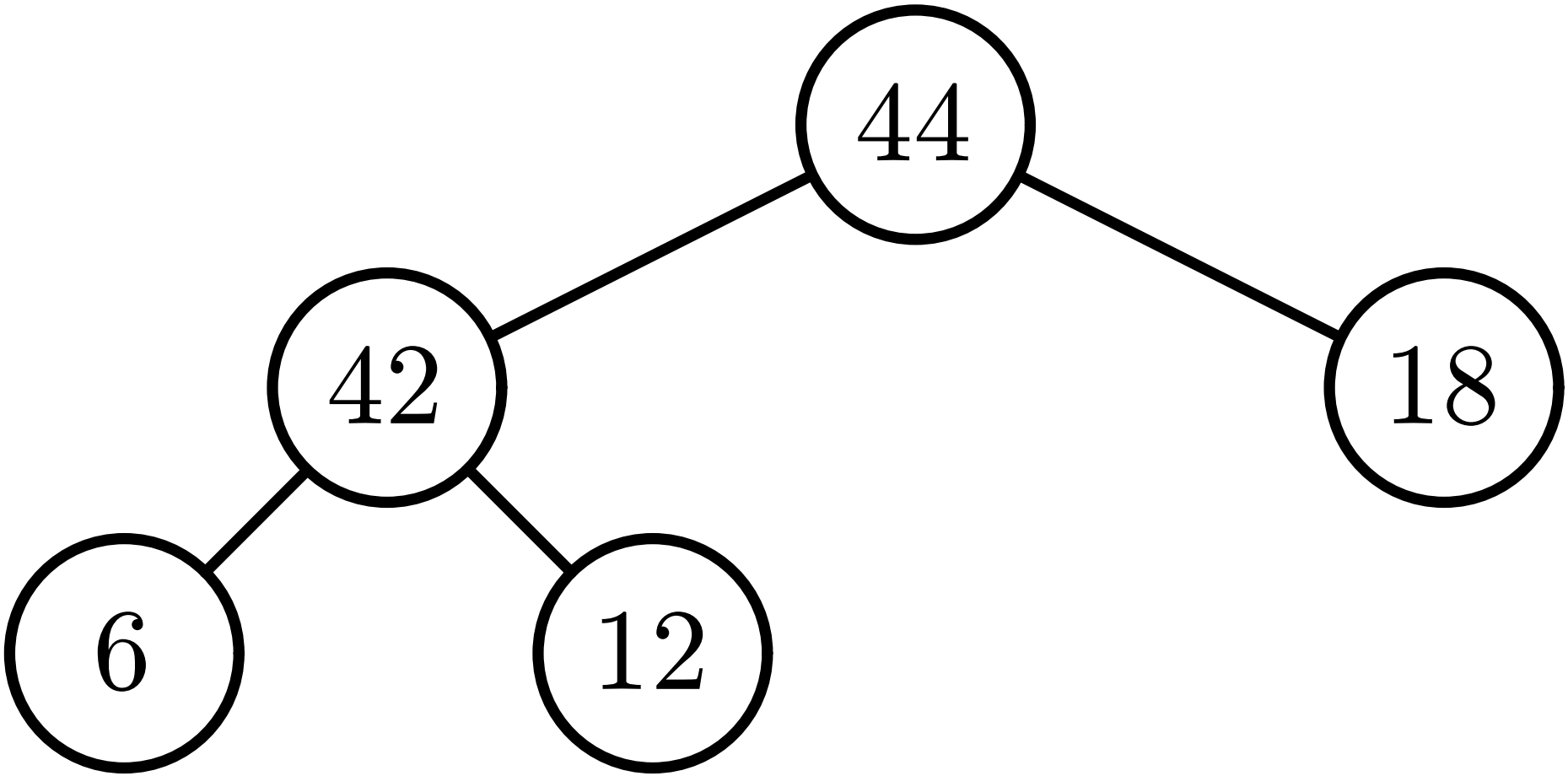
Oprava haldy



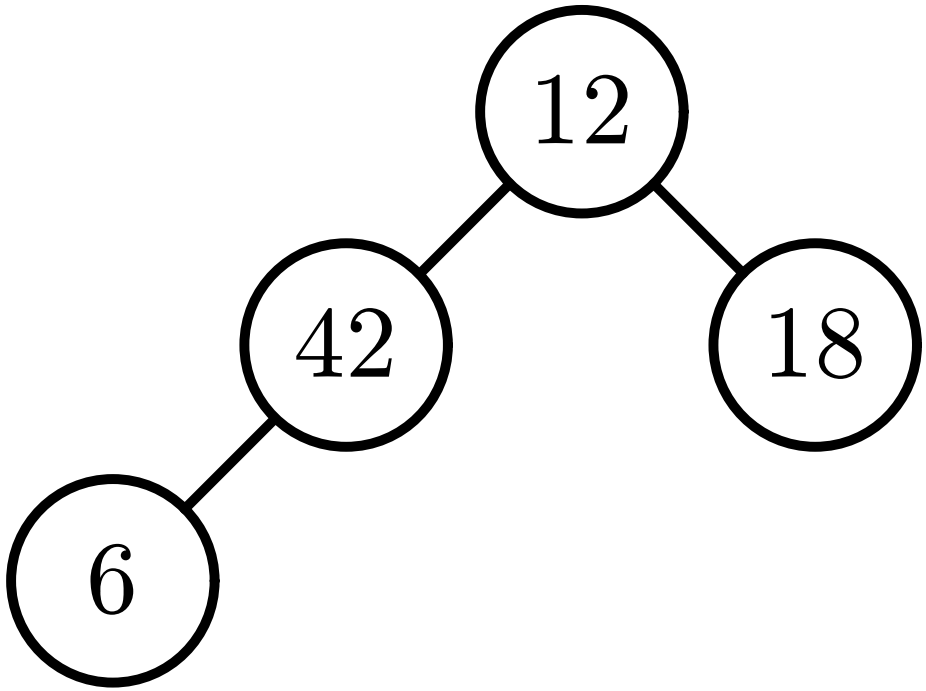
Odebrání 55



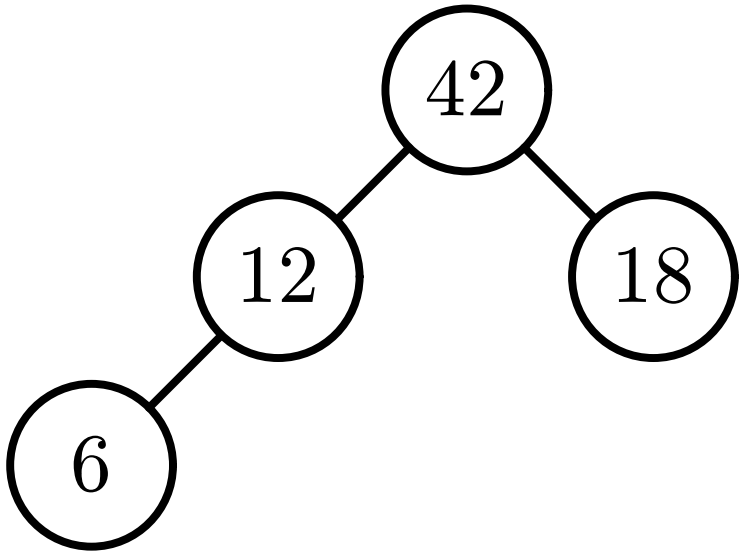
Oprava haldy



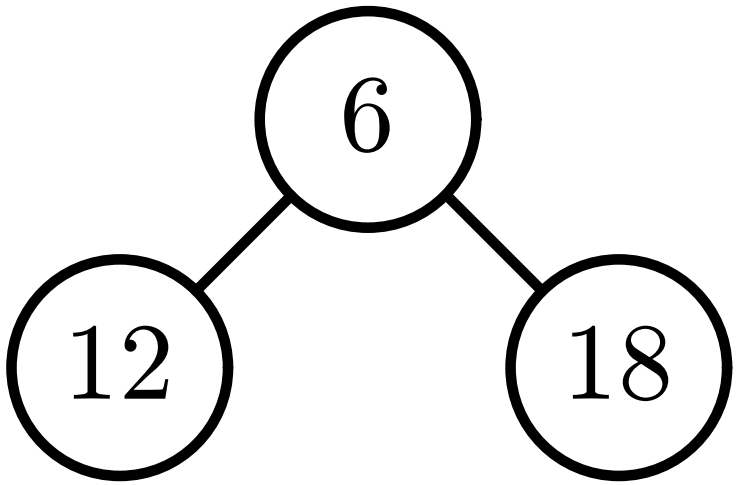
Odebrání 44



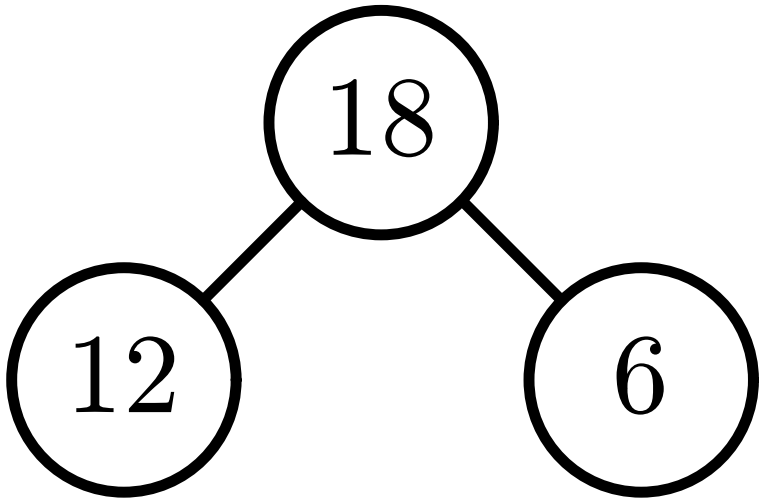
Oprava haldy



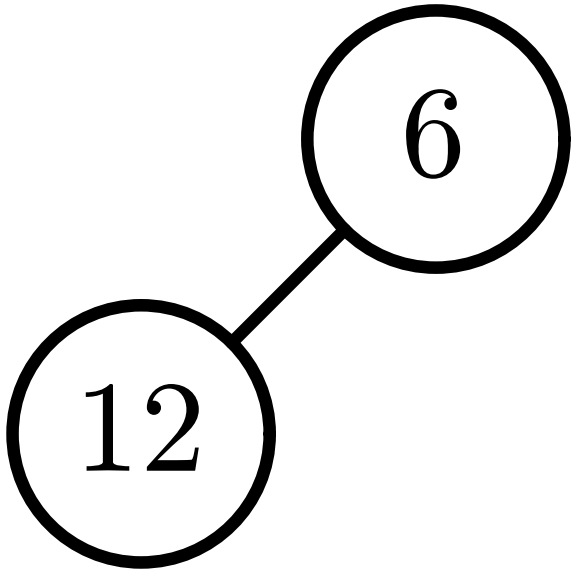
Odebrání 42



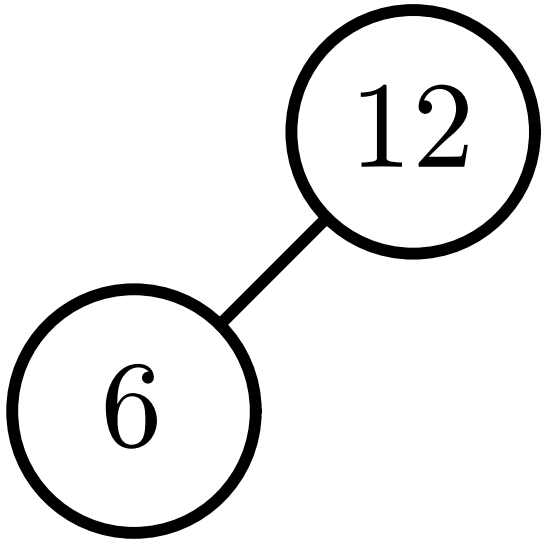
Oprava haldy



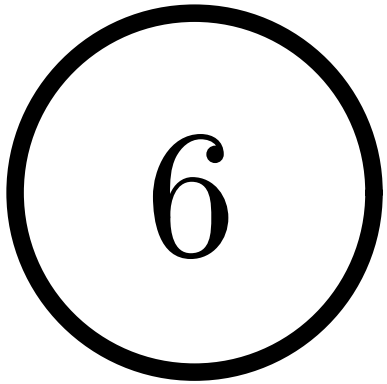
Odebrání 18



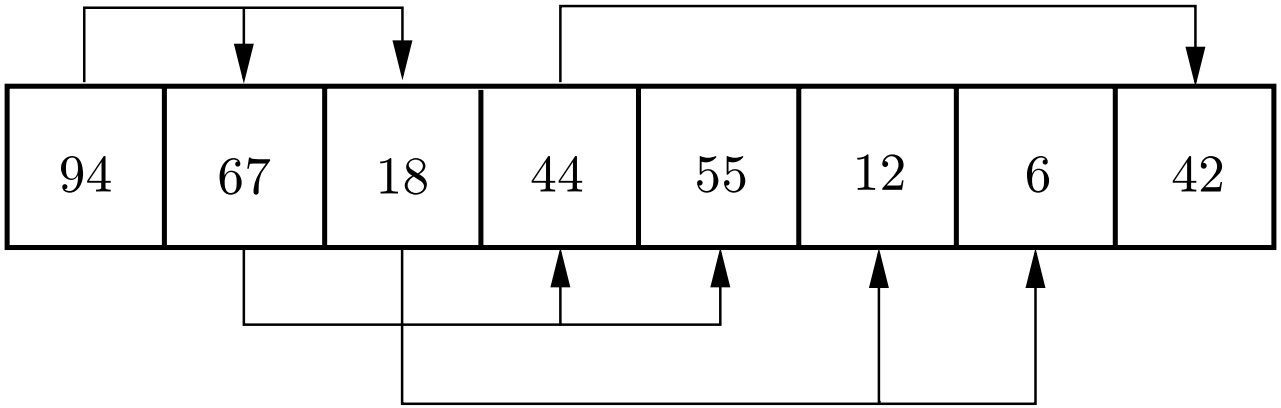
Oprava haldy



Koncový stav haldy



Halda reprezentovaná v poli



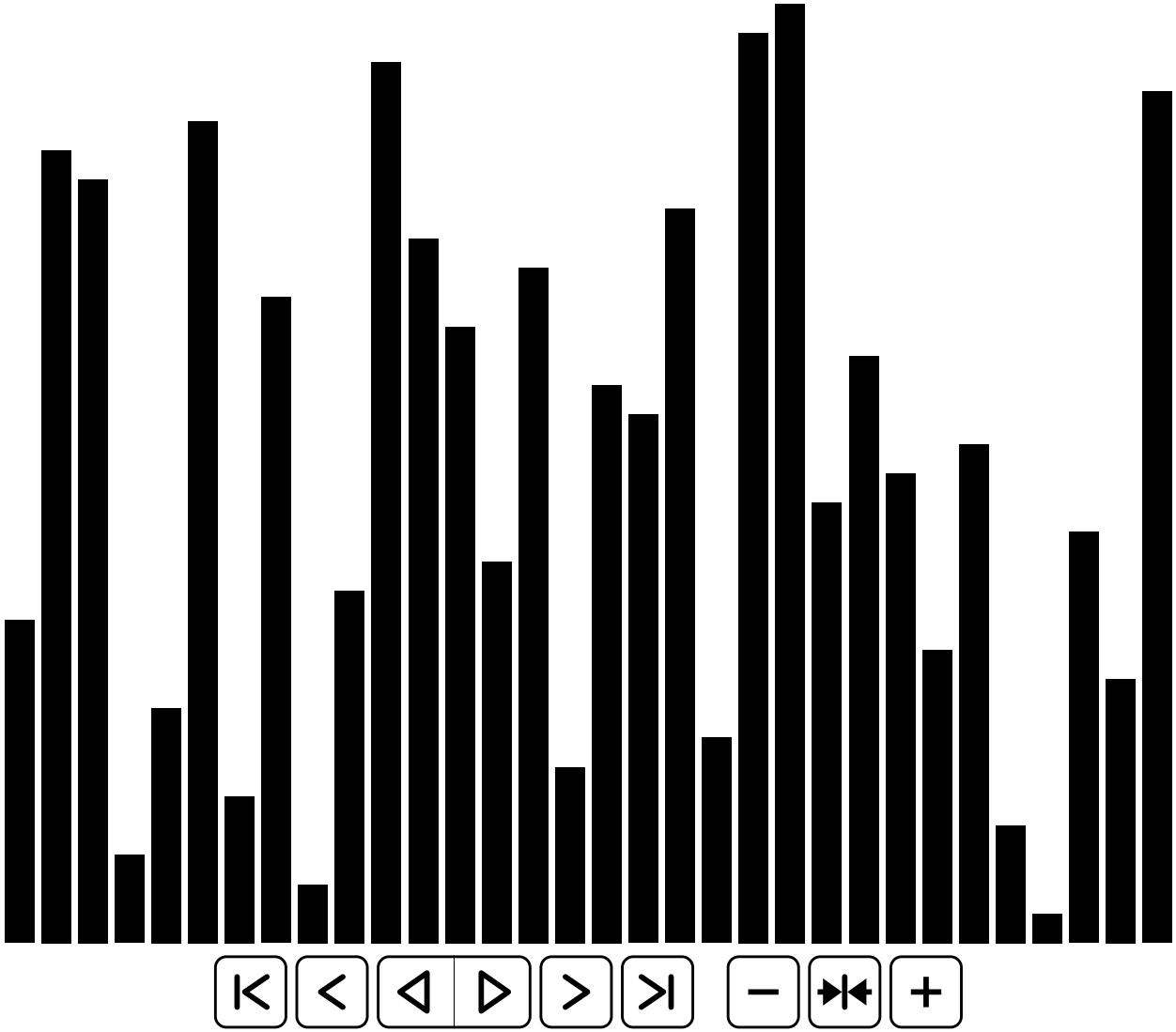
Třídění haldou – HeapSort

```
1  void BuildHeap(int a[], int n, int i)
2  {
3     int largest = i;
4     int left = 2 * i + 1;
5     int right = 2 * i + 2;
6     if (left < n && a[left] > a[largest])
7        largest = left;
8     if (right < n && a[right] > a[largest])
9        largest = right;
10    if (largest != i)
11    {
12       Exchange(a[i], a[largest]);
13       BuildHeap(a, n, largest);
14    }
15 }
```

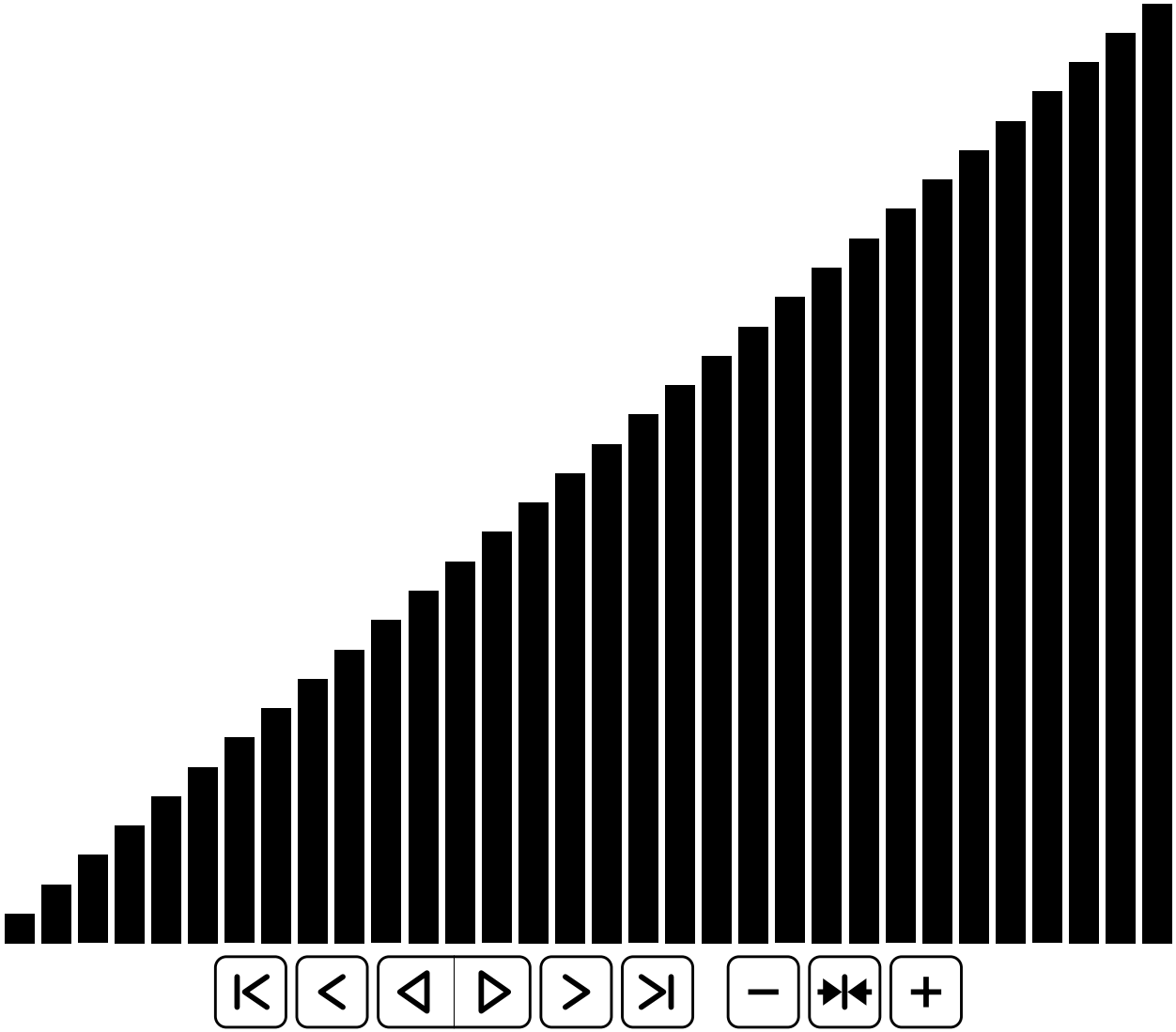
Třídění haldou – HeapSort

```
1  void HeapSort(int a[], const int n)
2  {
3     for (int i = n / 2 - 1; i >= 0; i--)
4        BuildHeap(a, n, i);
5     for (int i = n - 1; i >= 0; i--)
6     {
7        Exchange(a[0], a[i]);
8        BuildHeap(a, i, 0);
9     }
10 }
```

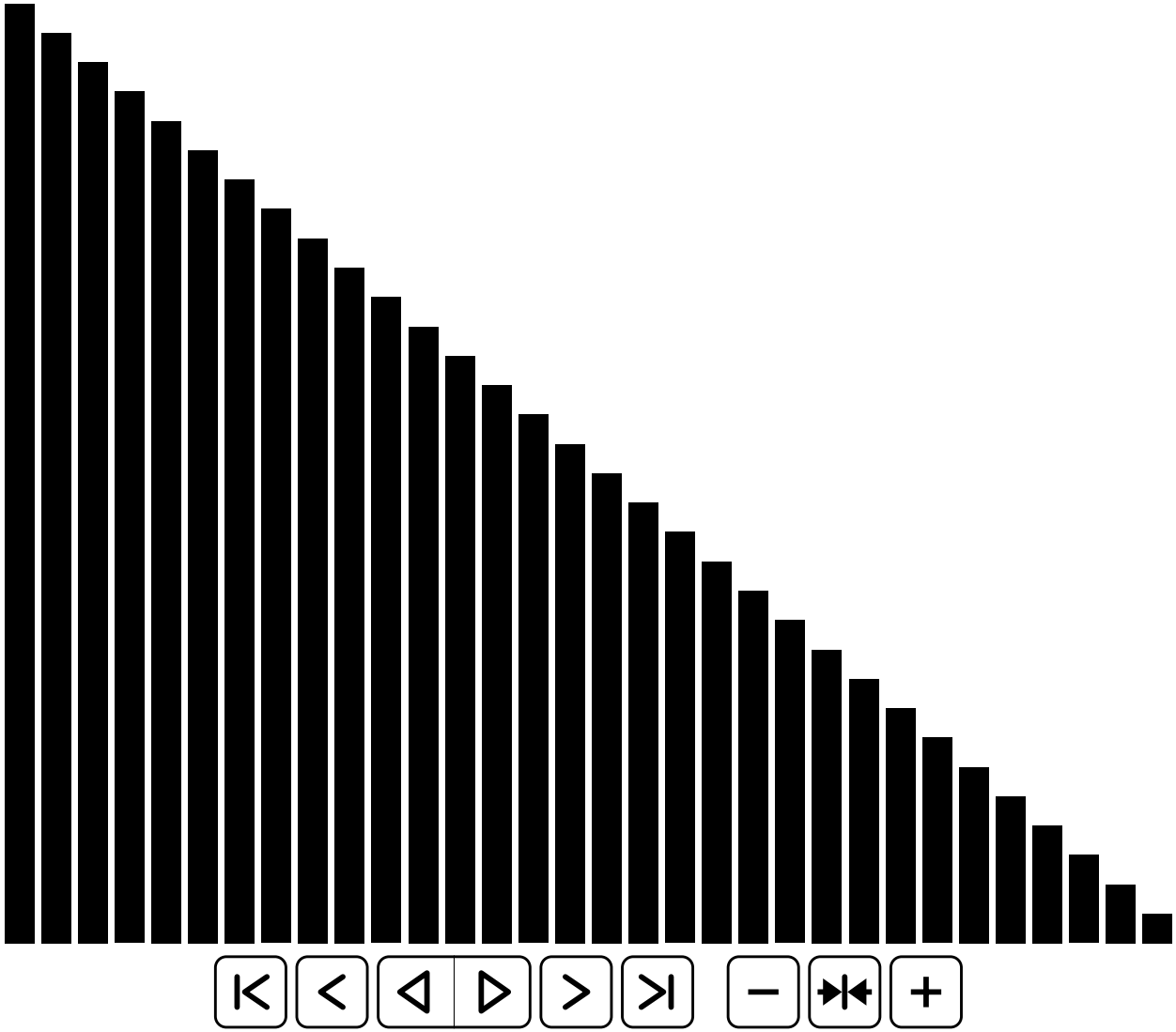
Třídění haldou – náhodná permutace



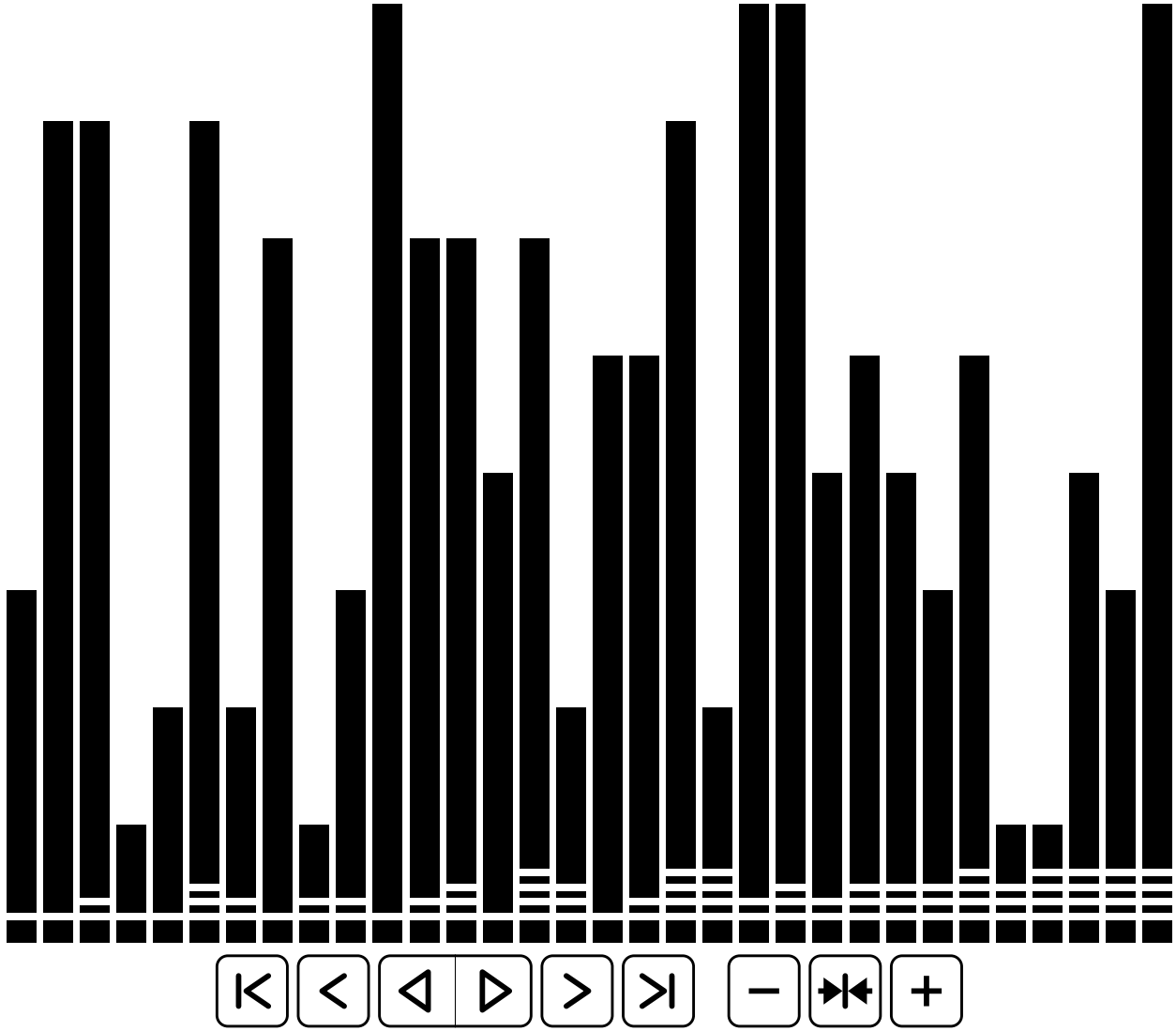
Třídění haldou – identická permutace



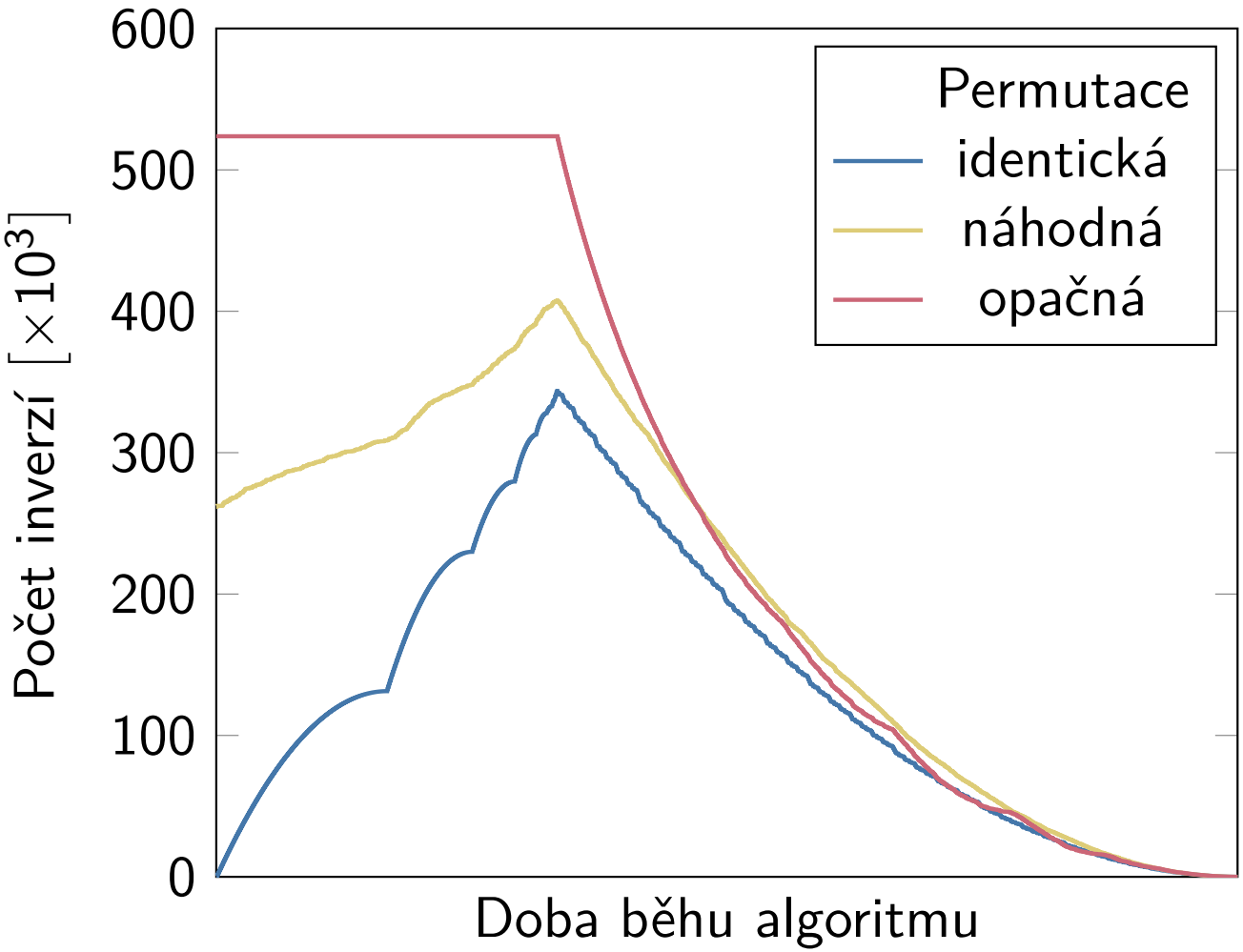
Třídění haldou – opačná permutace



Třídění haldou – permutace pouze několika hodnot



Průběh odstraňování inverzí v čase – třídění haldou



Děkuji za pozornost