

Bubble sort

pseudokód

```
function bubbleSort(array a)
  for i in 1 -> a.length - 1 do
    for j in 1 -> a.length - i - 1 do
      if a[j] < a[j+1]
        prohod'(a[j], a[j+1]);
```

java

```
//razeni od nejvyssiho
function bubbleSort(int[] a){
  for (i = 0; i < a.length-1; i++) {
    for (j = 0; j < a.length-i-1; j++) {
      if(a[j] < a[j+1]){
        help = a[j];
        a[j] = a[j+1];
        a[j+1] = help;
        swapped = true;
      }
    }
    if(!swapped) break;
  }
}
```

příklad

```
(3 2 8 7 6) //zadání pole, řadíme od největšího k nejmenšímu
(3 2 8 7 6) // 3 a 2 jsou v korektním pořadí, posuňme se o index
(3 2 8 7 6) // 8 > 2, prohodme je
(3 8 2 7 6) // 7 > 2, prohodme je (zde je vidět probublávání nejlehčí dvojky vzhůru)
(3 8 7 2 6) // 6 > 2, prohodme je
(3 8 7 6 2) // nový průchod polem: na posledním místě je nejlehčí prvek, tudíž se nám
              řazená úloha o jedna zkrátila, 8 > 3, prohodme je
(8 3 7 6 2) // 7 > 3, prohodme je
(8 7 3 6 2) // 6 > 3, prohodme je
(8 7 6 3 2) // seřazeno
```

Shaker sort

java

```
// radi od nejvyssiho
public static void shakerSort(int[] a) {
  for (int i = 0; i < a.length/2; i++) {
    boolean swapped = false;
    for (int j = i; j < a.length-i-1; j++) {
      if(a[j] < a[j+1]){
        int help = a[j];
        a[j] = a[j+1];
        a[j+1] = help;
      }
    }
  }
}
```

```

        swapped = true;
    }
}
for (int j = a.length-2-i; j > i; j--) {
    if(a[j] > a[j-1]){
        inthelp = a[j];
        a[j] = a[j-1];
        a[j-1] = help;
        swapped = true;
    }
}
if(!swapped) break;
}
}

```

Shell sort

```

java
public static int[] shellSort(int[] a) {
    int delka = a.length / 2;
    while (delka > 0) { //dokud mame co porovnavat
        for (int i = 0; i < a.length- delka; i++) {
            //upraveny insertion sort
            int j = i + delka;
            inthelp = a[j];
            while (j >= delka && help > a[j - delka]) {
                a[j] = a[j - delka];
                j -= delka; // j = j-delka;
            }
            a[j] = help;
        }
        if (delka == 2) { //zmena velikosti mezery
            delka = 1;
        } else {
            delka /= 2.2; //delka = delka/2.2;
        }
    }
    return a;
}

```

Insert sort

pseudokód I.

```

function insertionSort(array a)
    for i in 0 -> a.length - 2 do
        j = i + 1
        help = a[j]
        while j > 0 AND help > a[j-1] do //uvolni miesto hodnote
            a[j] = a[j-1]
            j--

```

```
a[j] = help //vloz hodnotu
```

pseudokód II.

```
insertion_sort(a)
  for j = 2 to a.length
    k = a[j]
    i = j - 1
    while i > 0 and a[i] > k
      a[i + 1] = a[i]
      i = i - 1
    a[i + 1] = k
```

java

```
public static void insertionSort(int[] a){
  for(int i = 0; i < a.length - 1; i++){
    int j = i + 1;
    int help = a[j];
    while(j > 0 && help > a[j-1]){
      a[j] = a[j-1];
      j--;
    }
    a[j] = help;
  }
}
```

příklad

```
(3 2 8 7 6) // Zadání, prvek 3 je triviálně seřazen
(3 2 8 7 6) // Vezmeme dvojku a vložíme jí na správné místo (tam už je)
(3 2 8 7 6) // 8 vložíme na první místo, zbytek čísel posuneme
(8 3 2 7 6) // 7 vložíme mezi 8 a 3, 3 a 2 posuneme
(8 7 3 2 6) // 6 vložíme mezi 7 a 3, čísla 3 a 2 posuneme
(8 7 6 3 2) // seřazeno
```

Selection Sort

pseudokód

```
function selectionSort(array a)
  for i in 0 -> a.length - 2 do
    maxIndex = i
    for j in (i + 1) -> (a.length - 1) do
      if a[j] > a[maxIndex]
        maxIndex = j
    prohod(a[i], a[maxIndex])
```

java

```
public static void selectionSort(int[] a){
  for(int i = 0; i < a.length-1; i++){
    int maxIndex = i;
    for(int j = i + 1; j < a.length; j++){
      if(a[j] > a[maxIndex]) {
```

```

        maxIndex = j;
    }
    int help = a[i];
    a[i] = a[maxIndex];
    a[maxIndex] = help;
}
}

```

příklad

```

(3 2 8 7 6) // zadání pole, řadíme od největšího k nejmenšímu
(3 2 8 7 6) // nejvyšší číslo je 8, prohodíme ho tedy s číslem 3 na indexu 0
(8 2 3 7 6) // nejvyšší číslo je 7, prohodíme ho tedy s číslem 2 na indexu 1
(8 7 3 2 6) // nejvyšší číslo je 6, prohodíme ho tedy s číslem 3 na indexu 2
(8 7 6 2 3) // nejvyšší číslo je 3, prohodíme ho tedy s číslem 2 na indexu 3
(8 7 6 3 2) // seřazeno

```

Merge sort

pseudokód - rekurze

```

merge_sort(a, p, r)
    if p < r
        q = b(p + r=2)c
        merge_sort(a, p, q)
        merge_sort(a, q - 1, r)
        merge(a, q, r)

```

java - rekurze

```

//razení od nejvyššího
//a2 = pomocné pole stejné délky jako array
//left = první index na který se smí sáhnout
//right = poslední index, na který se smí sáhnout
public static void mergeSort(int[] a, int[] a2, int left, int right){
    if(left == right) return;
    int middleIndex = (left + right)/2;
    mergeSort(a, a2, left, middleIndex);
    mergeSort(a, a2, middleIndex + 1, right);
    merge(a, a2, left, right);

    for(int i = left; i <= right; i++){
        a[i] = a2[i];
    }
}

```

java - iterativní

```

function merge(left, right)
    var list result
    while length(left) > 0 and length(right) > 0
        if first(left) <= first(right)
            append first(left) to result

```

```

        left = rest(left)
    else
        append first(right) to result
        right = rest(right)
    if length(left) > 0
        append left to result
    if length(right) > 0
        append right to result
    return result

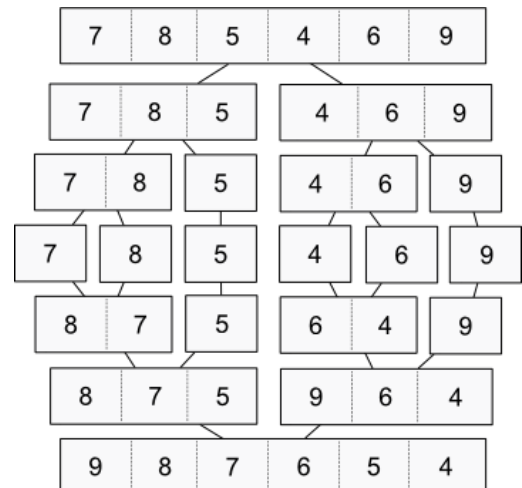
```

//slevani pro Merge sort

```

private static void merge(int[] a, int[] a2, int left, int right) {
    int middleIndex = (left + right)/2;
    int leftIndex = left;
    int rightIndex = middleIndex + 1;
    int auxIndex = left;
    while(leftIndex <= middleIndex && rightIndex <= right){
        if(a[leftIndex] >= a[rightIndex]){
            a2[a2Index] = a[leftIndex++];
        }else{
            a2[auxIndex] =
a[rightIndex++];
        }
        a2Index++;
    }
    while(leftIndex <= middleIndex){
        a2[a2Index] = a[leftIndex++];
        a2Index++;
    }
    while(rightIndex <= right){
        a2[a2Index] = a[rightIndex++];
        a2Index++;
    }
}

```



Quick sort

pseudokód - rekurze

```

procedure quicksort(List values)

```

```

    if values.size <= 1 then
        return values

```

pivot = náhodný prvek z values

Rozděl seznam values do 3 seznamů

```

        seznam1 = { prvky větší než pivot }

```

```

        seznam2 = { pivot }

```

```

        seznam3 = { prvky menší než pivot }

```

```

    return quicksort(seznam1) + seznam2 + quicksort(seznam3)

```

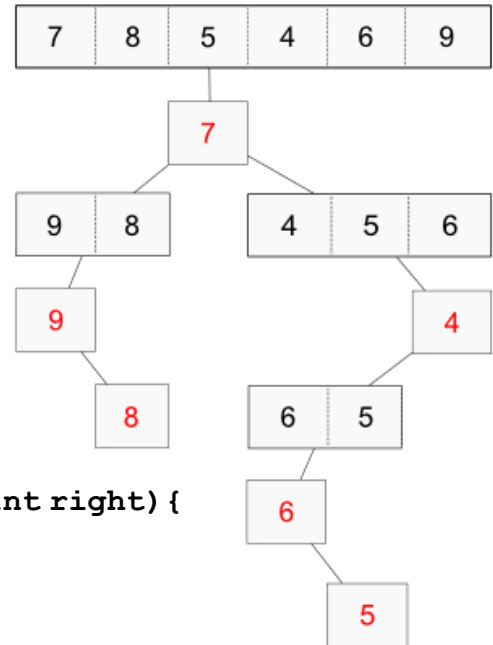
java - rekurze

```
//radi od nejvyssiho prvku  
//left index prvniho prvku, na který muzeme sahnout (leva mez (vcetne))  
//right index prvniho prvku, na který nemuzeme sahnout (prava mez (bez))
```

```
public static void quicksort(int[] a, int left, int right){  
    if(left < right){  
        int boundary = left;  
        for(int i = left + 1; i < right; i++){  
            if(a[i] > a[left]){  
                swap(a, i, ++boundary);  
            }  
        }  
        swap(a, left, boundary);  
        quicksort(a, left, boundary);  
        quicksort(a, boundary+1, right);  
    }  
}
```

```
//prohodi prvky v zadanem poli  
//left prvek 1  
//right prvek 2
```

```
private static void swap(int[] a, int left, int right){  
    int help = a[right];  
    a[right] = a[left];  
    a[left] = help;  
}
```



C - iterativní

```
void quicksort(int array[], int left_begin, int right_begin){  
    int pivot = array[(left_begin + right_begin) / 2];  
    int left_index, right_index, pom;  
    left_index = left_begin;  
    right_index = right_begin;  
    do {  
        while (array[left_index] < pivot && left_index < right_begin)  
            left_index++;  
        while (array[right_index] > pivot && right_index >  
            left_begin) right_index--;  
  
        if (left_index <= right_index){  
            pom = array[left_index];  
            array[left_index] = array[right_index];  
            array[right_index] = pom;  
            if (left_index < right_begin)  
                left_index++;  
            if (right_index > left_begin)  
                right_index--;  
        }  
    }  
    while (left_index < right_index);  
    if (right_index > left_begin) quicksort(array,
```

```
        left_begin, right_index);
    if (left_index < right_begin) quicksort(array,
        left_index, right_begin);
}
```

Counting sort

java

//return pole serazene od najnizsi honoty po nejvyssi

```
public static int[] countingSort(int[] a) {
    // pole do ktereho budeme radit, v pripade primitiv nema smysl
    // da se radit i bez nej, ale v pripade objektu by to jinak neslo
    int[] a2 = new int[a.length];

    // najdeme maximum a minimum
    int min = a[0];
    int max = a[0];
    for(int i = 1; i < a.length; i++) {
        if(a[i] < min) min = a[i];
        else if(a[i] > max) max = a[i];
    }

    // vytvorime pole do ktereho budeme pocitat
    int[] counts = new int[max - min + 1];

    // zinicilizujeme pocety vyskytu
    for(int i = 0; i < a.length; i++) {
        counts[a[i] - min]++;
    }

    // prepocitame vyskyty na posledni index dane hodnoty
    counts[0]--;
    for(int i = 1; i < counts.length; i++) {
        counts[i] = counts[i] + counts[i-1];
    }

    // Serad pole zprava doleva
    // 1) vyhledej posledni vyskyt dane hodnoty v poli vyskutu
    // 2) uloz hodnotu na prislusne misto v serazenem poli
    // 3) sniz index posledniho vyskytu dane hodnoty
    // 4) pokracuj s predchozi hodnotou vstupniho pole (goto: 1),
    terminuj, pokud jiz vsechny hodnoty byly zarazeny
    for(int i = a.length - 1; i >= 0; i--) {
        a2[counts[a[i] - min]--] = a[i];
    }

    return a2;
}
```

příklady

```
Vstupní pole: 9 6 6 3 2 0 4 2 9 3
Pole četností: 1 0 2 2 1 0 2 0 0 2
Pole výskytů: 0 0 2 4 5 5 7 7 7 9
Seřazené pole: 0 2 2 3 3 4 6 6 9 9
```

```
Vstupní pole: 2 8 9 8 0 8 8 9 4 6
Pole četností: 1 0 1 0 1 0 1 0 4 2
Pole výskytů: 0 0 1 1 2 2 3 3 7 9
Seřazené pole: 0 2 4 6 8 8 8 8 9 9
```

```
Vstupní pole: 9 2 1 9 4 1 5 7 5 3
Pole četností: 2 1 1 1 2 0 1 0 2
Pole výskytů: 1 2 3 4 6 6 7 7 9
Seřazené pole: 1 1 2 3 4 5 5 7 9 9
```

Radix sort

pseudokód

```
function radixSort(String s)
    for i in (s.length - 1) -> 0 do
        stableSort(s[i])
```

java

```
//radi od nejnižší hodnoty - vnitřní stabilní řazení: counting sort
//dimension = rozměr dat (fixní)
public static int[] radixSort(int[] a, int dimension){
    for(int i = dimension - 1; i >= 0; i--){
        a = countingSortForRadix(a, i);
        //stabilně seřadí dle i-te pozice
    }
    return a;
}

//Counting sort pro Radix sort - radi pole dle hodnoty na pozici
//position = pozice, podle které se bude radit

public static int[] countingSortForRadix (int[] a, int position) {
    // pole do kterého budeme radit, v případě primitiv nemá smysl
    // dá se radit i bez něj, ale v případě objektu by to jinak neslo
    int[] a2 = new int[a.length];

    // najdeme maximum a minimum
    int min = a[0];
    int max = a[0];
    for(int i = 1; i < a.length; i++) {
        if(a[i] < min) min = a[i];
        else if(a[i] > max) max = a[i];
    }
}
```

```

// vytvorime pole do ktoreho budeme pocitat
int[] counts = new int[max - min + 1];

// zinicilizujeme pocty vyskytu
for(int i = 0; i < a.length; i++) {
    counts[a[i] - min]++;
}

// prepocitame vyskyty na posledni index dane hodnoty
counts[0]--;
for(int i = 1; i < counts.length; i++) {
    counts[i] = counts[i] + counts[i-1];
}

// Serad pole zprava doleva
// 1) vyhlej posledni vyskyt dane hodnoty v poli vyskutu
// 2) uloz hodnotu na prislusne miesto v serazenem poli
// 3) sniz index posledniho vyskytu dane hodnoty
// 4) pokracuj s predchozi hodnotou vstupniho pole (goto: 1),
terminuj, pokud jiz vsechny hodnoty byly zarazeny
for(int i = a.length - 1; i >= 0; i--) {
    a2[counts[a[i] - min]--] = a[i];
}

return a2;
}

```

Bucket sort

java

```

//bucketCount = pocet bucketu
//return serazene pole (od nejmensiho k nejvyssimu)
public static int[] bucketSort(int[] a, int bucketCount){
    if(bucketCount <= 0) throw new IllegalArgumentException("Neplatny
pocet bucketu");
    if(a.length <= 1) return a;    //trivialne serazeno

    int high = a[0];
    int low = a[0];
    for(int i = 1; i < a.length; i++){ //najdeme nejvyssi a nejniysi
        if(a[i] > high) high = a[i];
        if(a[i] < low) low = a[i];
    }
    double interval = ((double)(high - low + 1))/bucketCount;
    //pocet cisel krytych jednim bucketem =
    pocet_cisel_celkem/pocet_bucketu

    ArrayList<Integer> buckets[] = new ArrayList[bucketCount];
    for(int i = 0; i < bucketCount; i++){
        //inicializace bucketu
        buckets[i] = new ArrayList();
    }
}

```

```
for(int i = 0; i < a.length; i++){
    //rozhozeni cisel do bucketu
    buckets[(int)((a[i] - low)/interval)].add(a[i]);
}
int pointer = 0;
for(int i = 0; i < buckets.length; i++){
    Collections.sort(buckets[i]); //mergeSort
    for(int j = 0; j < buckets[i].size(); j++){
        //sesypat zpet
        a[pointer] = buckets[i].get(j);
        pointer++;
    }
}
return a;
}
```