```
Metóda SelectSort
      Pre i od 0 do počet prvkov - 2, i++
            min := i;
            Pre j od i + 1 do počet prvkov - 1, j++
                  Ak Kľúč[j] < Kľúč[min] tak
                        min := j;
            Vymeň (i, min); //vymení prvky na miestach i a min
Koniec
```

```
Metóda InsertSort
      Pre i od 1 do počet prvkov - 1, i++
            j := 0;
            Pokiaľ (j < i) ∧ (Kľúč[j] < Kľúč[i])
                  j++;
            Vsuň (i, j); //vsunie prvok z miesta i na miesto j
Koniec
```

```
Metóda BubbleSort
      definuj premennú výmena;
      Rob
            výmena := false;
            Pre i od 0 do počet prvkov - 1, i++
                  Ak Kľúč[i] > Kľúč[i + 1] tak
                        Vymeň (i, i + 1);
                        výmena := true;
      Pokiaľ (výmena)
Koniec
```

```
Metóda QuickSort
      quick (0, počet prvkov – 1);
Koniec

Metóda quick (min, max)
      pivot = (min + max) div 2;
      ľavý = min;
      pravý = max;
            Rob
                  Pokiaľ (Kľúč[ľavý] < Kľúč[pivot])
                        ľavý++;
                  Pokiaľ (Kľúč[pravý] > Kľúč[pivot])
                        pravý--;
                  Ak ľavý ≤ pravý tak
                        Vymeň (ľavý, pravý);
                        ľavý++;
                        pravý--;
            Pokiaľ (ľavý ≤ pravý)
            Ak min < pravý tak
                  quick (min, pravý);
            Ak ľavý < max tak
                  quick (ľavý, max);
Koniec
```

```
Metóda HeapSort
	definuj premennú výmena;
	// postavenie haldy
	Pre i od 1 do počet prvkov - 1, i++
		aktuálny =  i;
		Rob
			výmena := false;
			otec := (aktuálny – 1) div 2;
			Ak (aktuálny > 0) ∧ (Kľúč[aktuálny] > Kľúč[otec]) tak
				Vymeň (aktuálny, otec);
				aktuálny := otec;
				výmena := true;
		Pokiaľ (výmena)
	// vyberanie z haldy
	Pre i od počet prvkov – 1 do 1, i-- // OPAČNÝ CYKLUS!!!
		Vymeň (0, i);
		aktuálny := 0;
		Rob
			výmena := false;
			ľavý := aktuálny * 2 + 1;
			pravý := aktuálny * 2 + 2;
			Ak (ľavý < i) ∧ (pravý < i) tak
				max := Kľúč[ľavý] > Kľúč[pravý] ? ľavý : pravý;
			inak
				max := ľavý < i ? ľavý : pravý;
			Ak (max < i) ∧ (Kľúč[max] > Kľúč[aktuálny]) tak
				Vymeň (max, aktuálny);
				aktuálny := max;
				výmena := true;
		Pokiaľ (výmena)
Koniec
```

```
Metóda ShellSort
      shell (⌈log10(počet prvkov)⌉); // je potrebné vhodne zvoliť krok
Koniec

Metóda shell (krok)
      Pre delta od 0 do krok - 1, delta++ // triedenie častí vzdialených o krok
            Pre i od delta do počet prvkov - 1, i += krok
                  j := i;
                  Pokiaľ (j – krok ≥ delta) ∧ (Kľúč[j] < Kľúč[j - krok])
                        Vymeň (j, j - krok);
                        j -= krok;
      Ak (krok > 1) tak
            shell (krok – 1);
Koniec
```

RadixSort predpokladá existenciu „priehradiek“. Priehradky sú implementované ako fronty a spolu sú držané v poli, ktoré je **indexované** jednotlivými **komponentami** kľúča. Počet priehradiek závisí od počtu možných hodnôt komponentu kľúča. Teda ak sú kľúče čísla, jednotlivé komponenty čísla sú číslice, je potrebné vytvoriť pole 10 frontov (jeden komponent nadobúda hodnoty od 0 po 9). Ak by bol kľúč reťazec a jednotlivé komponenty sú (základné) znaky, tak je potrebné vytvoriť 26 frontov (komponent nadobúda hodnoty od a po z), atď.

```
Metóda RadixSort
      list := nový (linked) list párov;
      priehradky := VytvorPriehradky;
      max := 0;
      Pre každý pár z tabuľky
            list.Vlož(pár);
            Ak (dĺžka kľúča (pár.Kľúč) > max) tak
                  max := dĺžka kľúča (pár.Kľúč);
      Pre i od 1 do max, i++ // ideme od najmenej významnej zložky kľúča!
            NaplňPriehradky (i, list, priehradky);
            NaplňList (list, priehradky);
      vyčisti tabuľku;
      Pre každý pár z listu
            tabuľka.Vlož(pár.Kľúč, pár.Hodnota);
Koniec

Metóda NaplňPriehradky(n, list, priehradky)
      Pre každý pár z listu
            index := Vyber n-tý komponent z kľúča pár.Kľúč;
            priehradky[index].Push(pár);
Koniec

Metóda NaplňList(list, priehradky)
      vyčisti list;
      Pre každý front z priehradiek
            Pokiaľ (front nie je prázdny)
                  list.Vlož(front.Pop);
Koniec
```

MergeSort potrebuje tri pomocné štruktúry – fronty párov tabuliek. Tieto štruktúry využíva na rozraďovanie a spájanie prvkov tabuľky.

```
Metóda MergeSort
	frontRozdeľ1 := nový front párov;
	frontRozdeľ2 := nový front párov;
	frontSpoj := nový front párov;
	Pre každý pár z tabuľky
		frontSpoj.Push(pár);
	i := 1;
	Pokiaľ (i < počet prvkov)
		Rozdeľ(i, frontSpoj, frontRozdeľ1, frontRozdeľ2);
		Spoj(i, frontSpoj, frontRozdeľ1, frontRozdeľ2);
		i *= 2;
	Rozdeľ(i, frontSpoj, frontRozdeľ1, frontRozdeľ2);
	Spoj(i, frontSpoj, frontRozdeľ1, frontRozdeľ2);
	vyčisti tabuľku;
	Pokiaľ (frontSpoj nie je prázdny)
		pár := frontSpoj.Pop;
		tabuľka.Vlož(pár.Kľúč, pár.Hodnota);
Koniec


Metóda Rozdeľ(n, spoj, rozdeľ1, rozdeľ2)
	počet := 0;
	prvý := true;
	Pokiaľ (spoj nie je prázdny)
		Ak (počet % n = 0) tak
			počet := 0;
			prvý := !prvý;
		Ak (prvý) tak
			rozdeľ1.Push(spoj.Pop);
		inak
			rozdeľ2.Push(spoj.Pop);
		počet++;
Koniec
```

```
Metóda Spoj(n, spoj, rozdeľ1, rozdeľ2)
	prvých := 0;
	druhých := 0;
	Rob
		Ak (prvých = 0) ∧ (druhých = 0) tak
			prvých := minimum(n, rozdeľ1.Počet);
			druhých := minimum(n, rozdeľ2.Počet);
		kľúč1 := prvých > 0 ? rozdeľ1.Peek.Kľúč : null;
		kľúč2 := druhých > 0 ? rozdeľ2.Peek.Kľúč : null;
		Ak (kľúč1 ≠ null) ∧ (kľúč2 ≠ null) tak
			Ak (kľúč1 < kľúč2) tak
				prvých--;
				spoj.Push(rozdeľ1.Pop);
			inak
				druhých--;
				spoj.Push(rozdeľ2.Pop);
		inak
			Ak (kľúč1 ≠ null) tak
				prvých--;
				spoj.Push(rozdeľ1.Pop);
			inak
				Ak (kľúč2 ≠ null) tak
					druhých--;
					spoj.Push(rozdeľ2.Pop);
	Pokiaľ ((rozdeľ1 nie je prázdny) ∨ (rozdeľ2 nie je prázdny))
Koniec
```