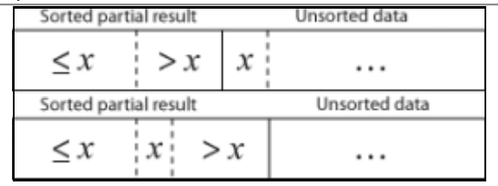
třídící algoritmy

sekvenční třídění

Select sort - najde nejmenší prvek a vloži ho na konec, opakuje
$O(n^2)$ - vždy

- \+ vhodný pro hodně přeházené
- − nevhodný pro předtříděné
- − nevhodný pro větší data

Insert sort - vzdy vezme prvek z nesetridene casti pole a vlozi ho do setridene casti
$O(n^2)$ - v prům.i nejhorším případě v nejlepším O(n)
O(n+1) až O(1) - paměti

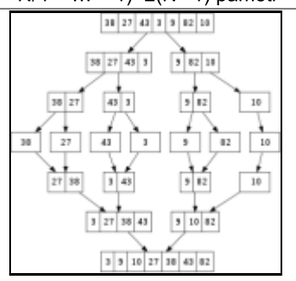


- \+ vhodný pro předtříděné
- − nevhodný pro hodně přeházené
- \+ stabilní
- − nevhodný pro větší data

Bubble sort - prochází pole a prohazuje prvky podle velikosti
-velké prvky probublávají na konec
$O(n^2)$ - v prům.i nejhorším případě, O(n) v nejlepším
O(n+1) až O(1) - paměti

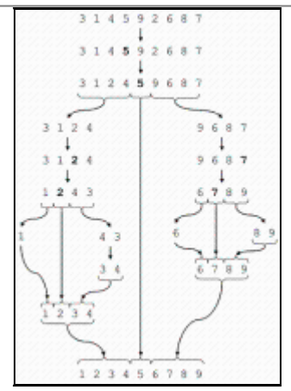
- \+ vhodný pro předtříděné
- − nevhodný pro hodně přeházené
- \+ stabilní
- − nevhodný pro větší data

Merge sort - rekurzivně děl posloupnost napůl a pak ji slij
O(nlogn) vždy
O(2n)=2*(N/2 + N/4 + … + 1)=2(N −1) paměti



- \+ stabilní
- − v základu paměťově náročný
- \+ stabilní časová náročnost
- − v základu nevhodný pro malinká data

Quick sort - vezme prvek(pivot) ze sekvence a rozdělí jí na 2 podle
toho kdo je větší nebo menší než pivot, ty pak rekurzivně setřídí
nejlepší - pivot medián: T(n)=2T(n/2)+O(n) ⇒ a=2 b=2 d=1 ; 2=21 ⇒ Θ(n log n)
nejhorší - pivot špatně: T(n)=T(n-1)+Θ(n)=Θ(n^2)
průměr - pivot náhodně: O(n log n)
O(n) až O(log n) paměti

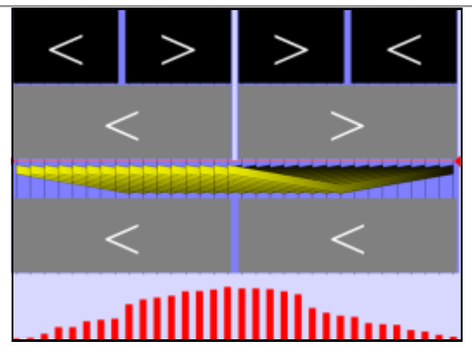


Heapsort - nasypeme prvky do max-haldy a pak je po jednom odebirame
O(n log n)

porovnávací algoritmy

třídící síť - sestava komparátorů (hradel co porovnaní dvě čísla)
pro setřídění sekvence, může pracovat paralelne

Bitonické třídění- alg.má 2 části:
1. rekurzivně setřídí 1/2 sekvence jednu vzestupne
a druhou sestupně a tím vytvoří bit. posloupnost
2. zmerguje je:
   a) vytvoří z nich 2 bit. posloupnosti (menších a větších čísel)
   b) zmerguje je rekurzivně
T(n)=T(n/2)+log n => logn+logn/2+..+1 => O(log^2 n)



bitonická posloupnostje takova co po zacyklení má dva monotóní úseky

přihrádkové třídění - lineární třídění
O(n+počet přihrádek) nemusí být rychlejší než klasické algoritmy

- \+ stabilní

radix/pidgeonhole/count sort- vytvoří pole podle rozsahu hodnot vstupu
a pak do něj jedním průchodem vloží hodnoty ze vstupu
O(|rozsah vstupu|) paměti

bucket sort - vytvoří malé přihrádky které lze setřídit v konst.čase
O(n) paměti v nejhorším případě může mít O(n+n) složitost