

Téma 4 - Řazení

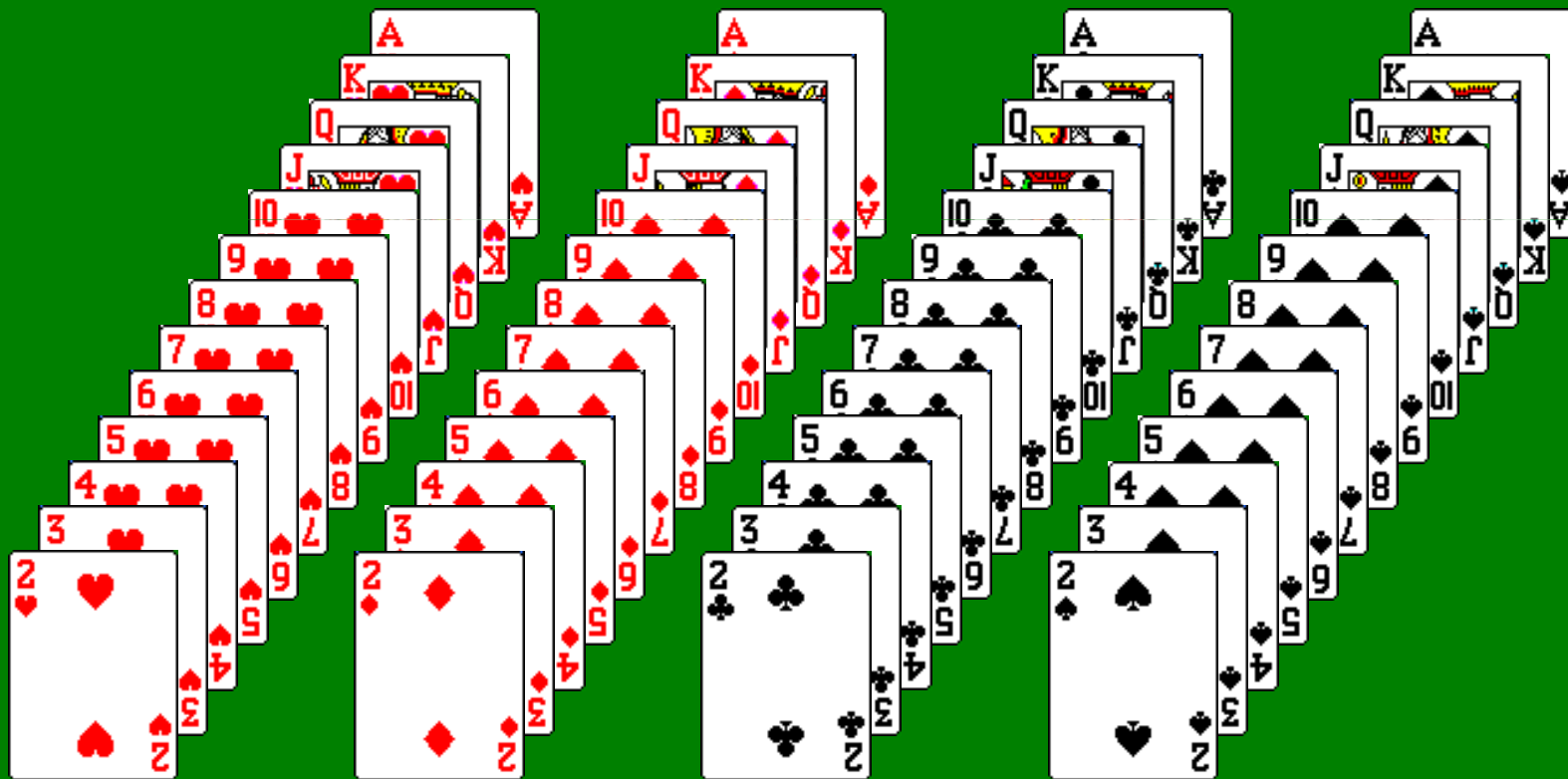
Podle originálu RNDr. M. Berezovského
s opravdu minimem úprav ...

Sorting

**Různé algoritmy
mají
různou složitost**

Sorting

...to this!



Různé algoritmy mají různou složitost: $O(n)$, $\Omega(n^2)$, $\Theta(n \cdot \log_2(n))$, ...

Sorting

Cvičení

Navrhněte a popište algoritmus pro seřazení balíčku karet.

- **Smíte používat obě ruce, v každé smíte držet vždy nanejvýš jednu kartu.**
- **Během řazení můžete odkládat karty do několika pomocných balíčků.**
- **V každém okamžiku jsou vidět a známe pouze hodnoty dvou karet.**

Úkolem je sestavit algoritmus tak, aby využíval co nejmenšího počtu kroků.

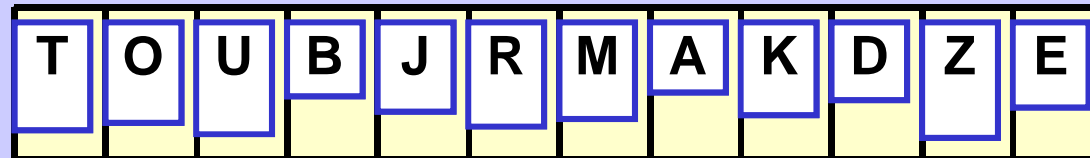
Sorting

Selection Sort

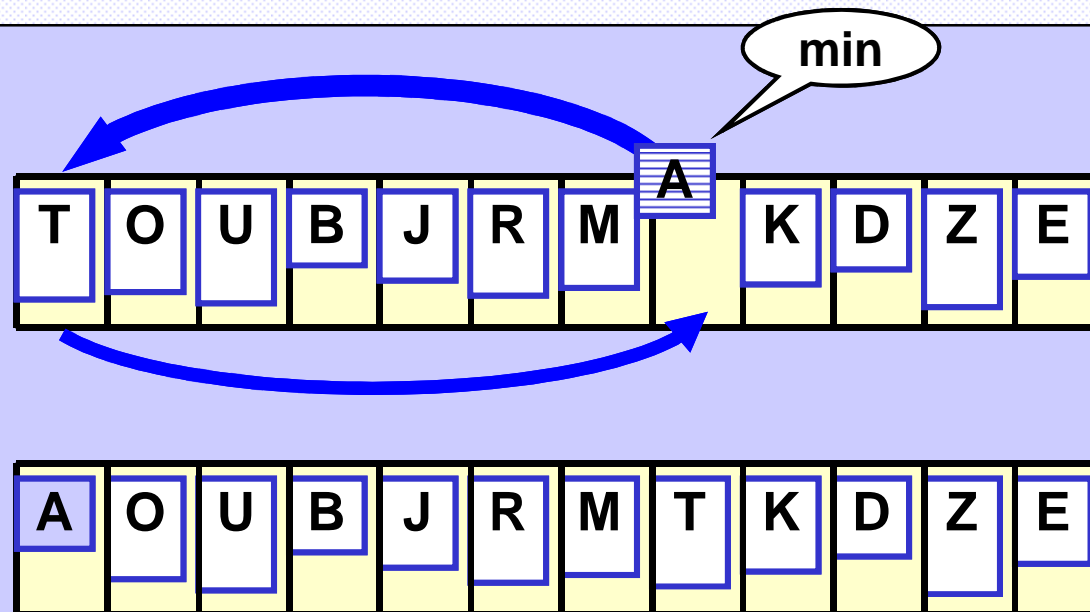
Řazení výběrem
(minima nebo maxima)

Selection Sort

Start

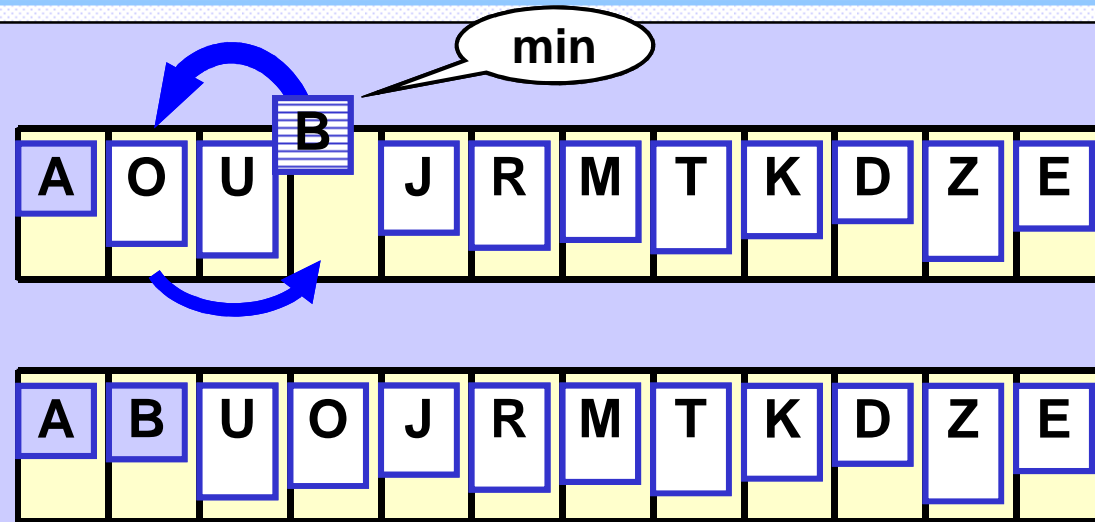


Step 1

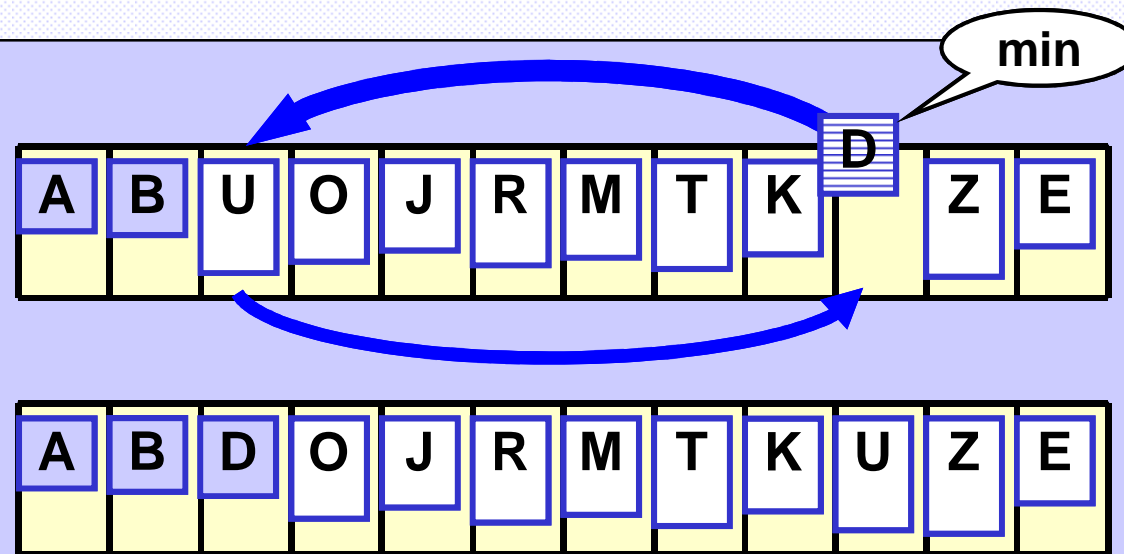


Selection Sort

Step 2



Step 3

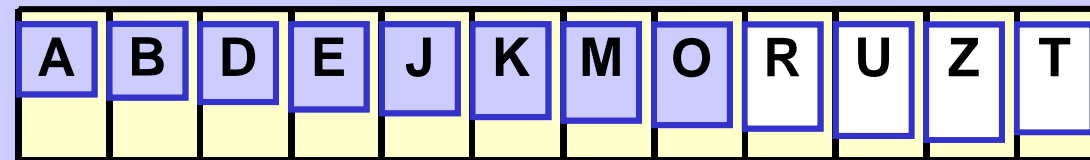
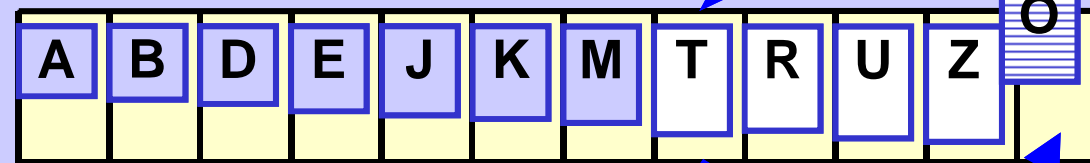


Selection Sort

...

...

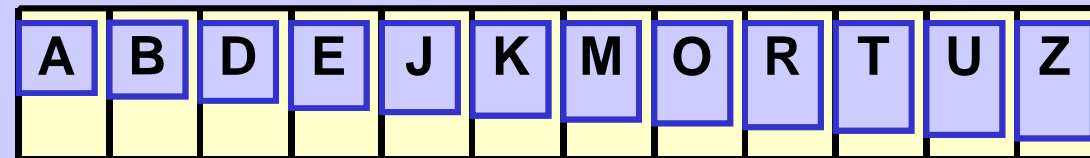
Step i



...

...

Sorted

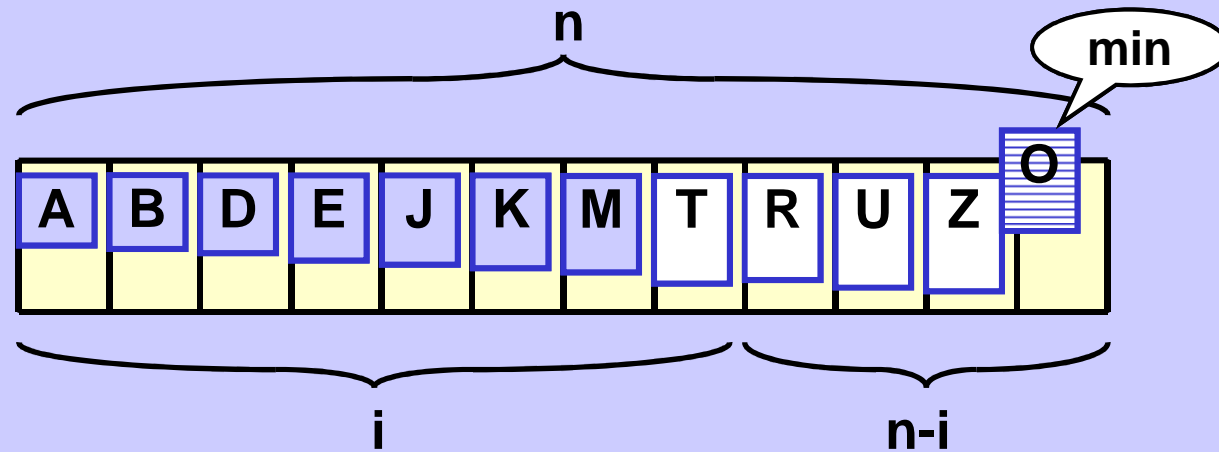


Selection Sort

```
for (i = 0; i < n-1; i++) {  
    // select min  
    jmin = i;  
    for (j = i+1; j < n; j++) {  
        if (a[j] < a[jmin]) {  
            jmin = j;  
        }  
    }  
    // swap min  
    min = a[jmin];  
    a[jmin] = a[i];  
    a[i] = min;  
}
```

Selection Sort

Step i



Select minimum



.....

(n-i) tests

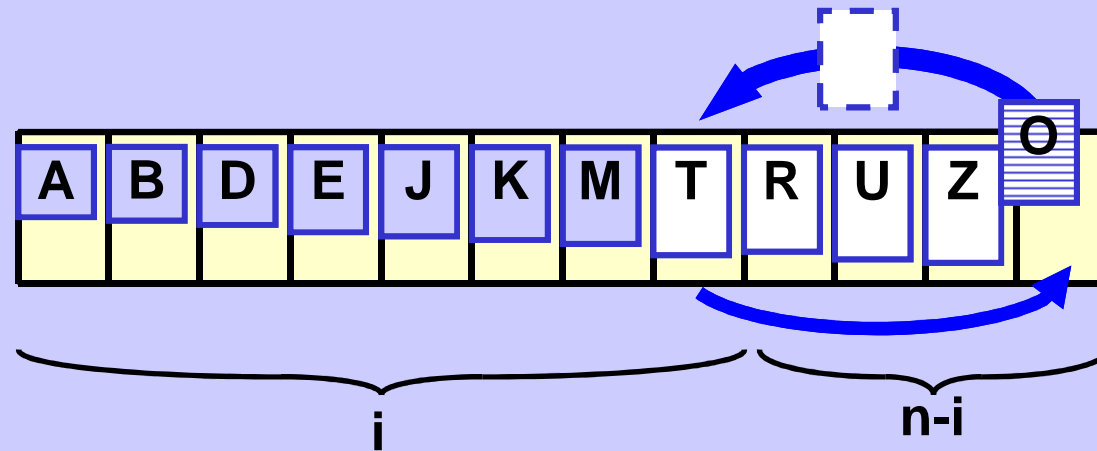
Tests total

$$\sum_{i=1}^{n-1} (n-i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i = n(n-1) - \frac{n(n-1)}{2} =$$

$$\frac{1}{2}(n^2 - n)$$

Selection Sort

Step i



moves 3

Moves total

$$\sum_{i=1}^{n-1} 3 = 3(n-1)$$

Selection Sort

Summary

Tests total

$$\frac{1}{2}(n^2 - n) = \Theta(n^2)$$

Moves total

$$3(n - 1) = \Theta(n)$$

**Operations
total**

$$\frac{1}{2}(n^2 - n) + 3(n - 1) = \Theta(n^2)$$

Asymptotic complexity of Selection Sort is $\Theta(n^2)$

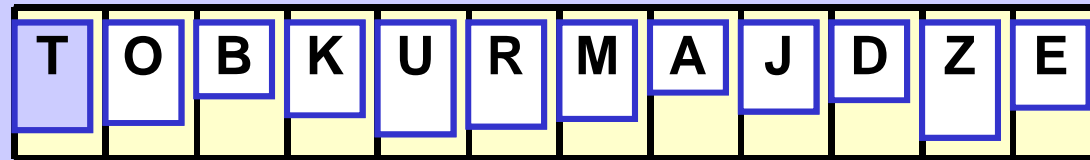
Sorting

Insertion Sort

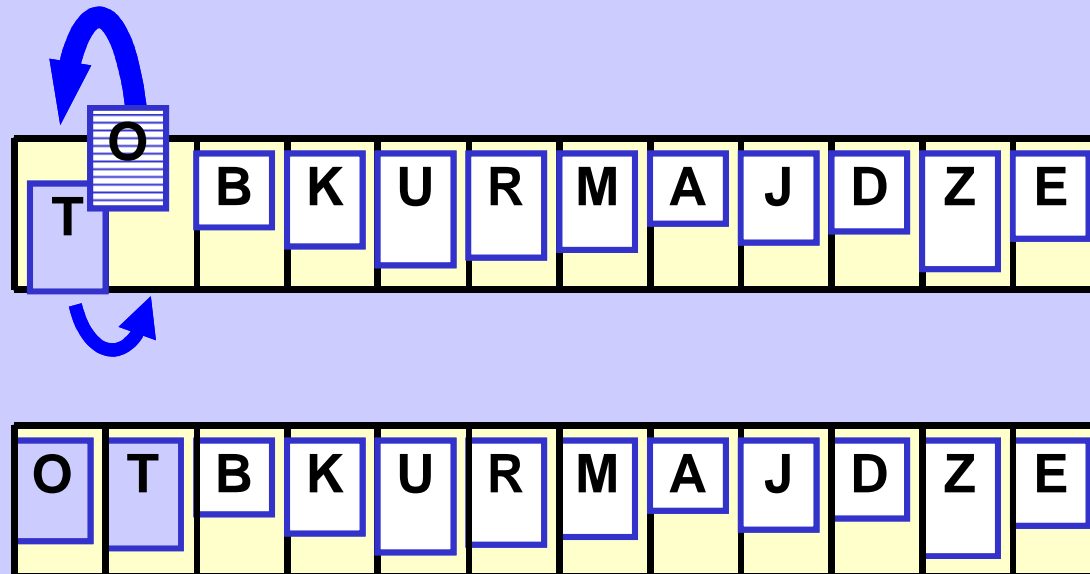
Řazení vkládáním (na adekvátní pozici)

Insertion Sort

Start

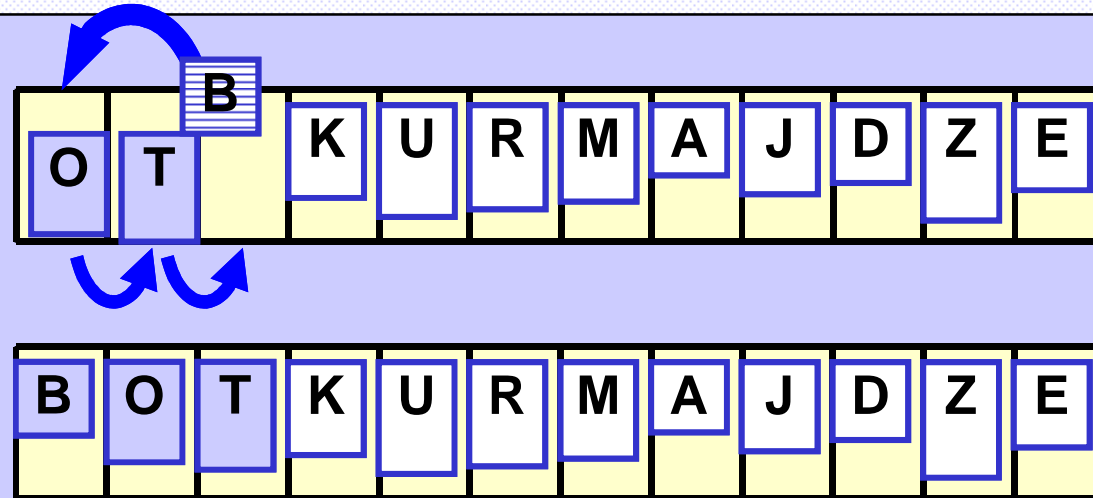


Step 1

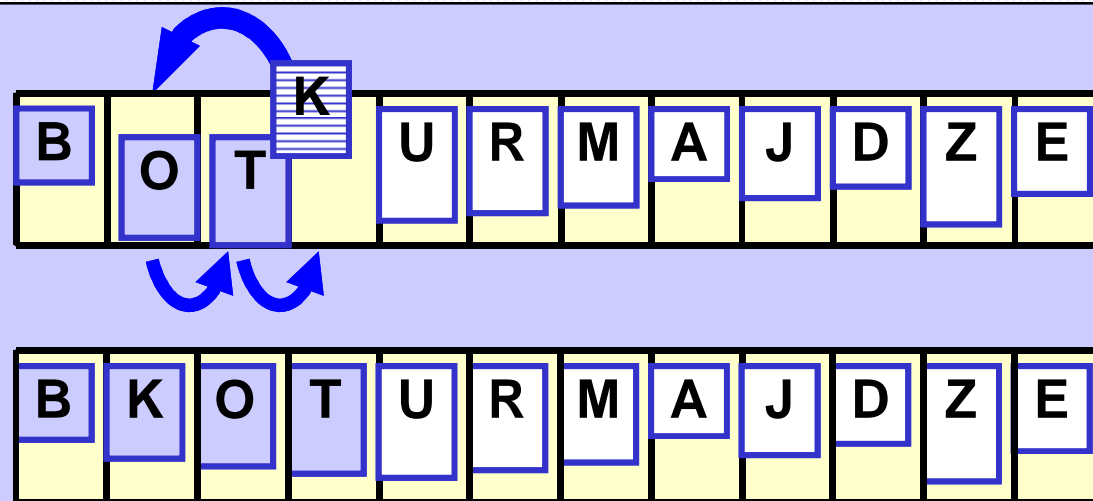


Insertion Sort

Step 2



Step 3

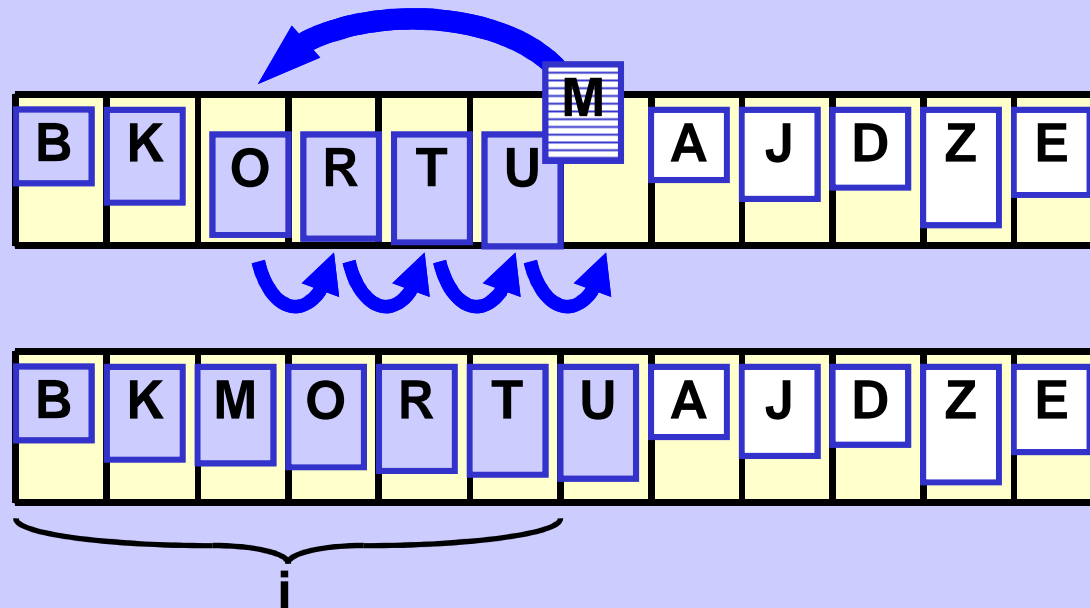


Insertion Sort

...

...

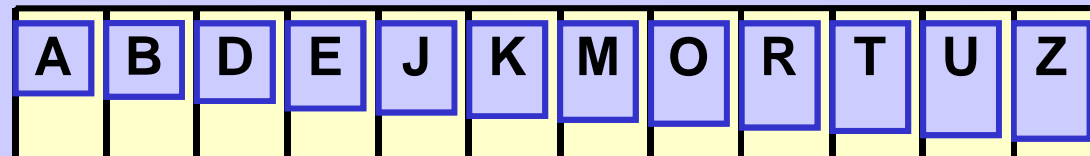
Step i



...

...

Sorted

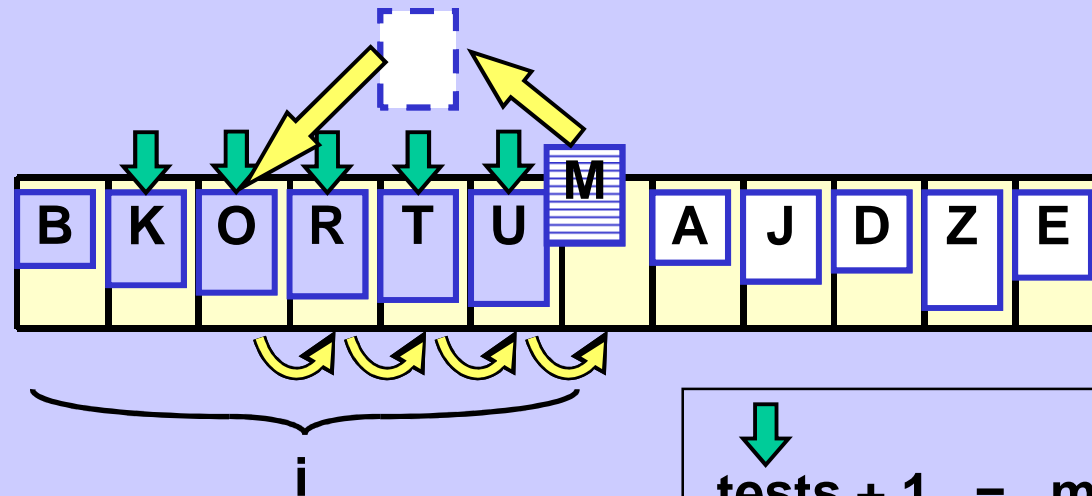




Insertion Sort

```
for (i = 1; i < n; i++) {  
    // find & make  
    // place for a[i]  
  
    insVal = a[i];  
    j = i-1;  
    while ((j >= 0) && (a[j] > insVal)) {  
        a[j+1] = a[j];  
        j--;  
    }  
  
    // insert a[i]  
    a[j+1] = insVal;  
}
```

Insertion Sort

Step i



 tests + 1 =  moves

tests

1

best case

i

worst case

$(i+1)/2$

average case

moves

2

best case

i+1

worst case

$(i+3)/2$

average case

Insertion Sort

Summary

Tests total

$$n - 1 = \Theta(n) \quad \text{best case}$$

$$(n^2 - n)/2 = \Theta(n^2) \quad \text{worst case}$$

$$(n^2 + n + 2)/4 = \Theta(n^2) \quad \text{average case}$$

Moves total

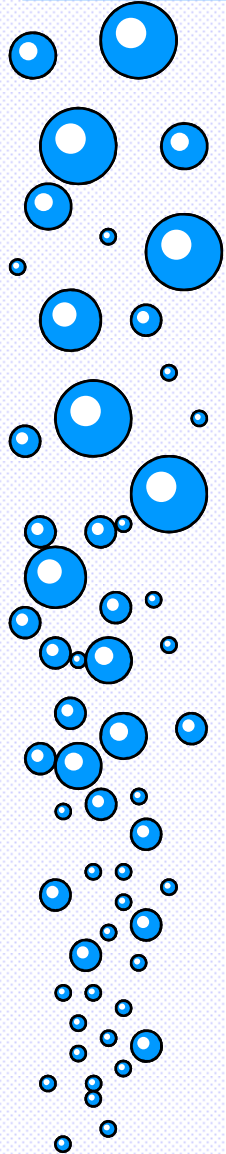
$$2n - 2 = \Theta(n) \quad \text{best case}$$

$$(n^2 + n - 2)/2 = \Theta(n^2) \quad \text{worst case}$$

$$(n^2 + 5n - 6)/4 = \Theta(n^2) \quad \text{average case}$$

Asymptotic complexity of Insertion Sort is $O(n^2)$ (!!)

Sorting



Bubble Sort

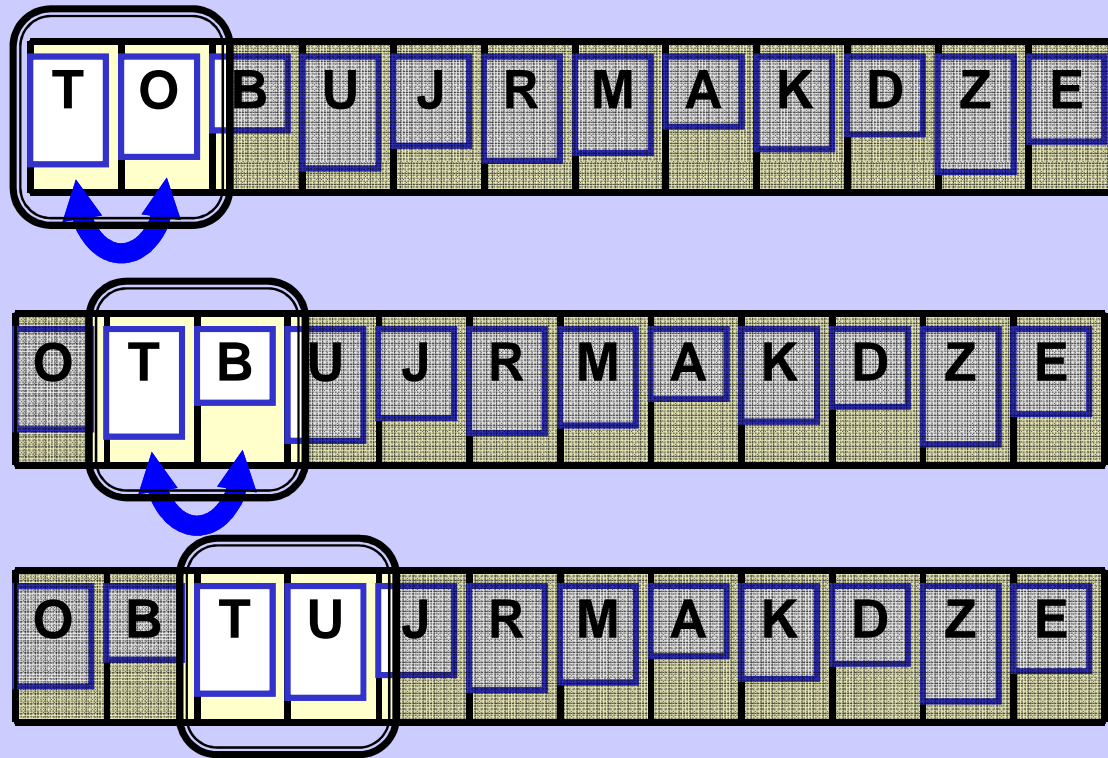
Bublínkové třídění

Bubble Sort

Start

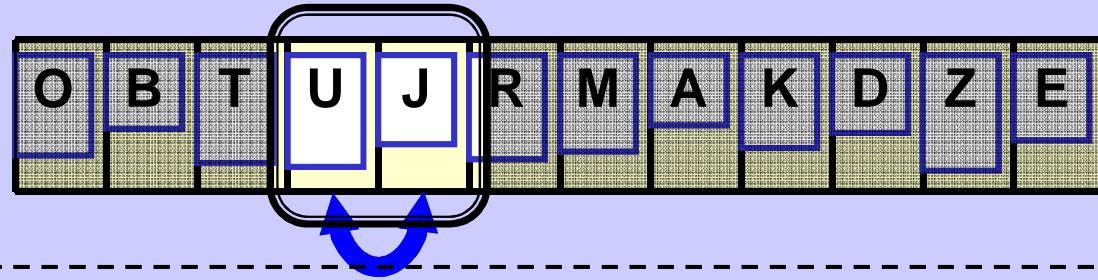
T	O	B	U	J	R	M	A	K	D	Z	E
---	---	---	---	---	---	---	---	---	---	---	---

Phase 1

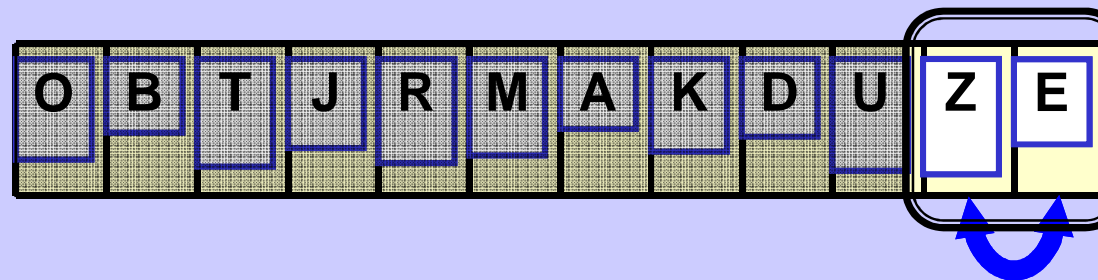


Bubble Sort

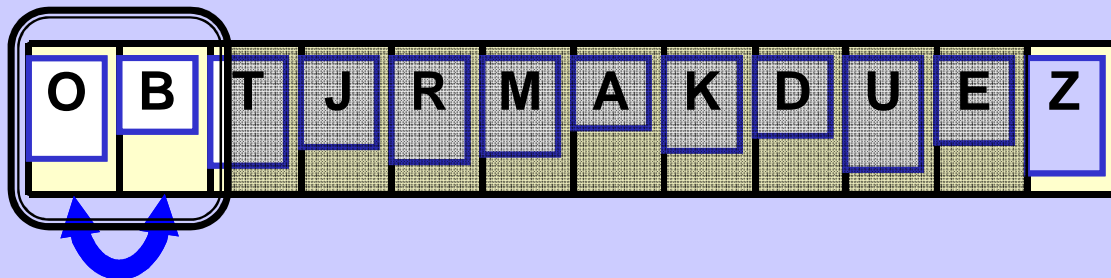
Phase 1



... etc ...



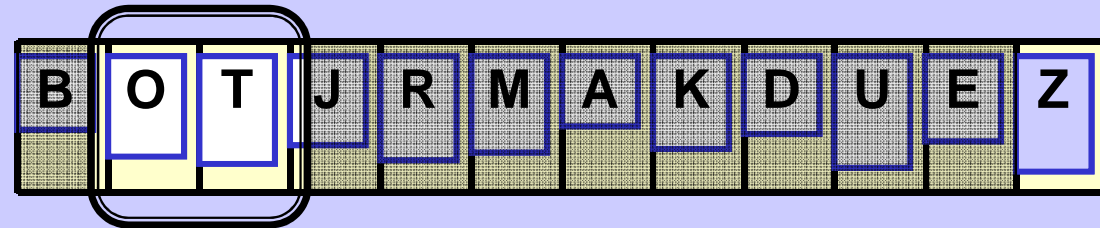
Phase 2



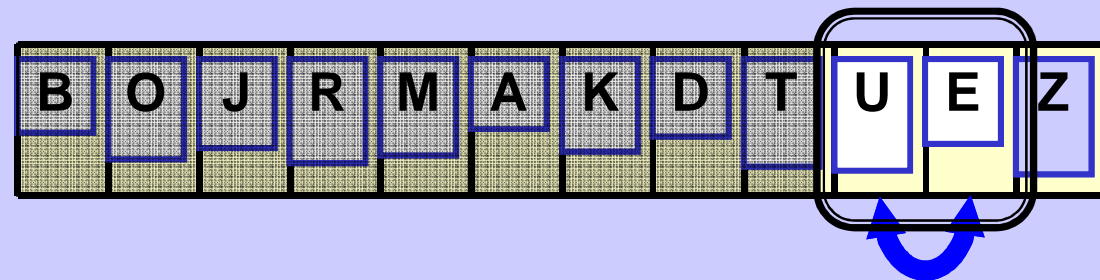
Různé algoritmy mají různou složitost: $O(n)$, $\Omega(n^2)$, $\Theta(n \cdot \log_2(n))$, ...

Bubble Sort

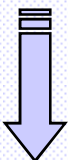
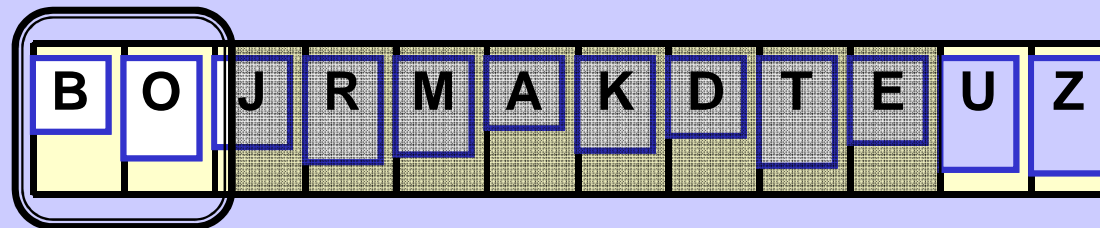
Phase 2



... etc ...



Phase 3

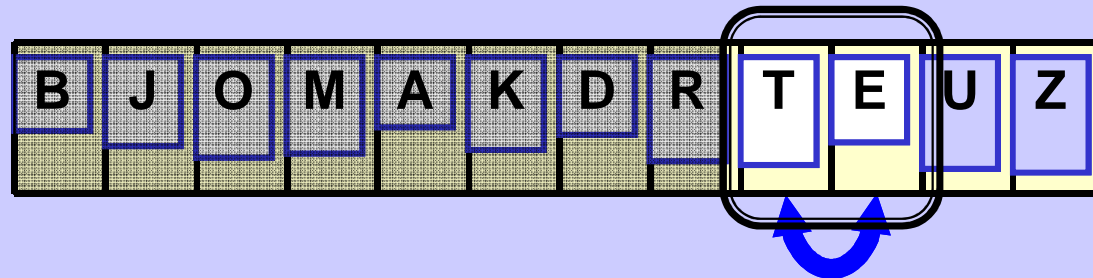


Různé algoritmy mají různou složitost: $O(n)$, $\Omega(n^2)$, $\Theta(n \cdot \log_2(n))$, ...

Bubble Sort

Phase 3

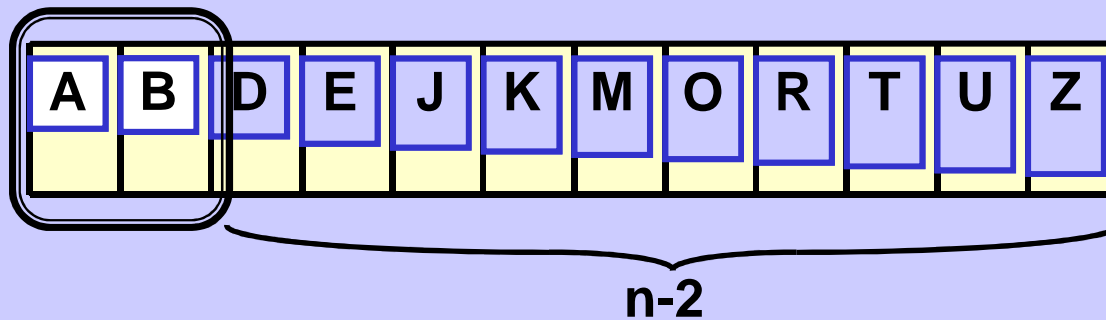
... etc ...



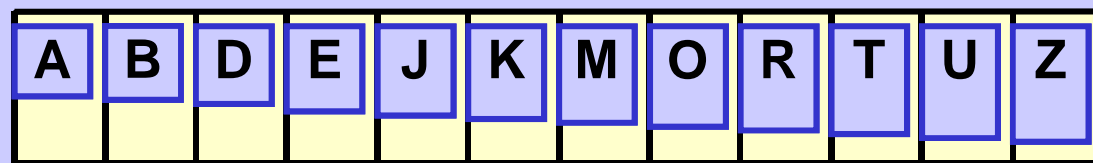
...

...

Phase n-1



Sorted



Bubble Sort

```
for (lastPos = n-1; lastPos > 0; lastPos--) {  
    for (j = 0; j < lastPos-1; j++) {  
        if (a[j] > a[j+1]) swap(a, j, j+1); } }  

```

Summary

Tests total

$$(n-1) + (n-2) + \dots + 2 + 1 = \frac{1}{2}(n^2 - n) = \Theta(n^2)$$

Moves total

$$0 = \Theta(1)$$

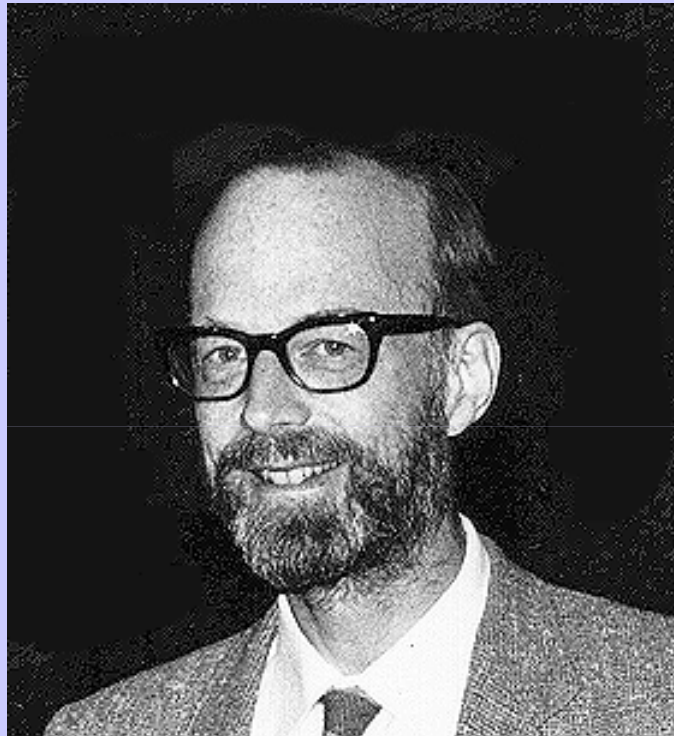
best case

$$\frac{1}{2}(n^2 - n) = \Theta(n^2)$$

worst case

Asymptotic complexity of Bubble Sort is $\Theta(n^2)$

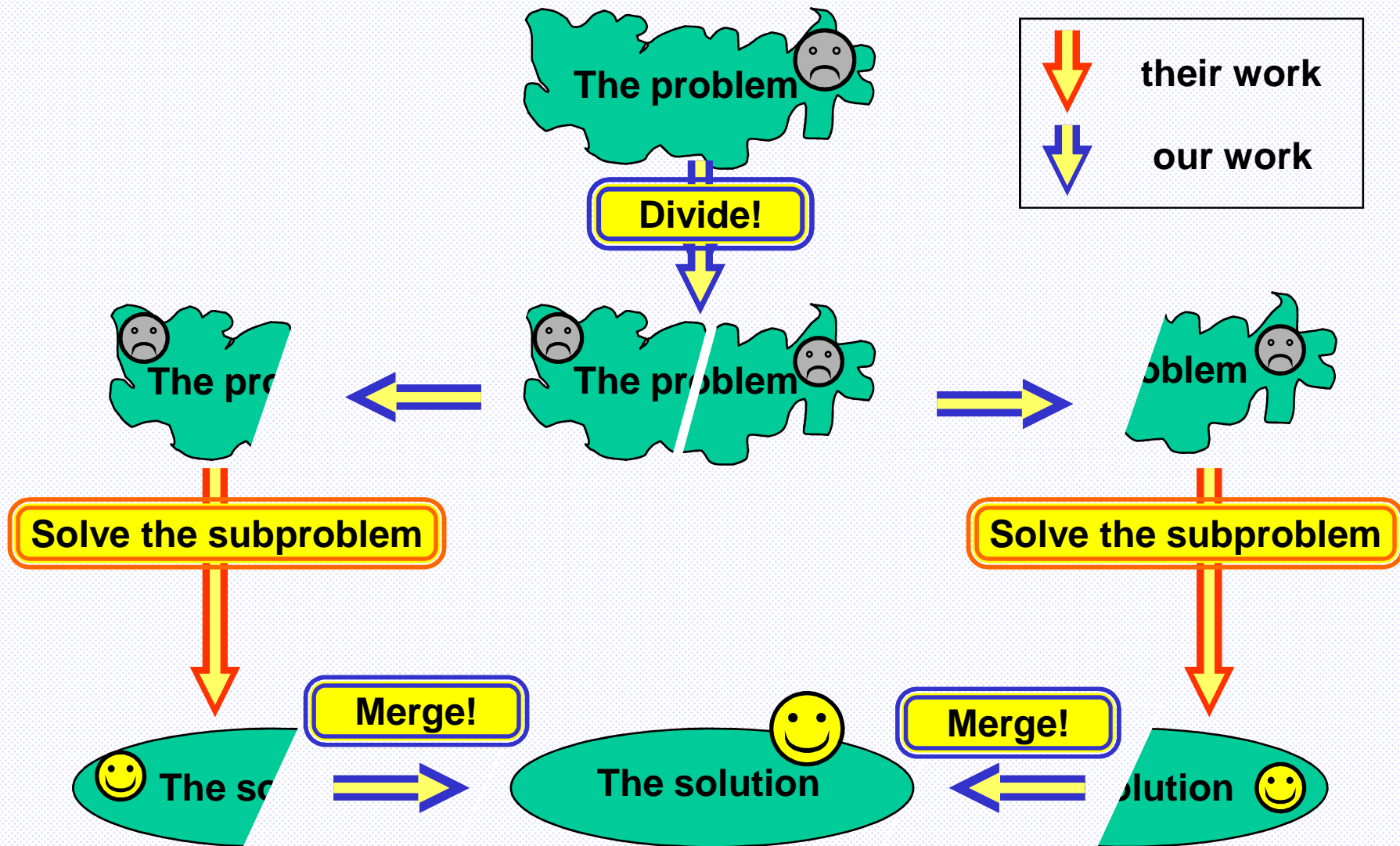
QuickSort



Sir Charles Antony Richard Hoare

C.A.R. Hoare: Quicksort. Computer Journal, Vol. 5, 1, 10-15 (1962)

Divide & Conquer! Divide et Impera!

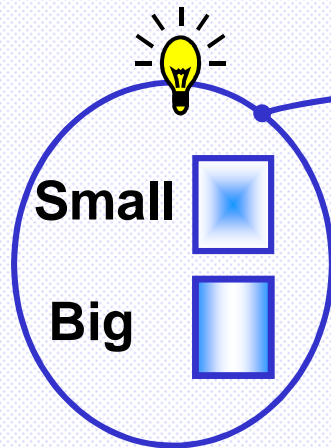


QuickSort

The idea

Divide & Conquer!

Start



M A K D R B T O J U Z E

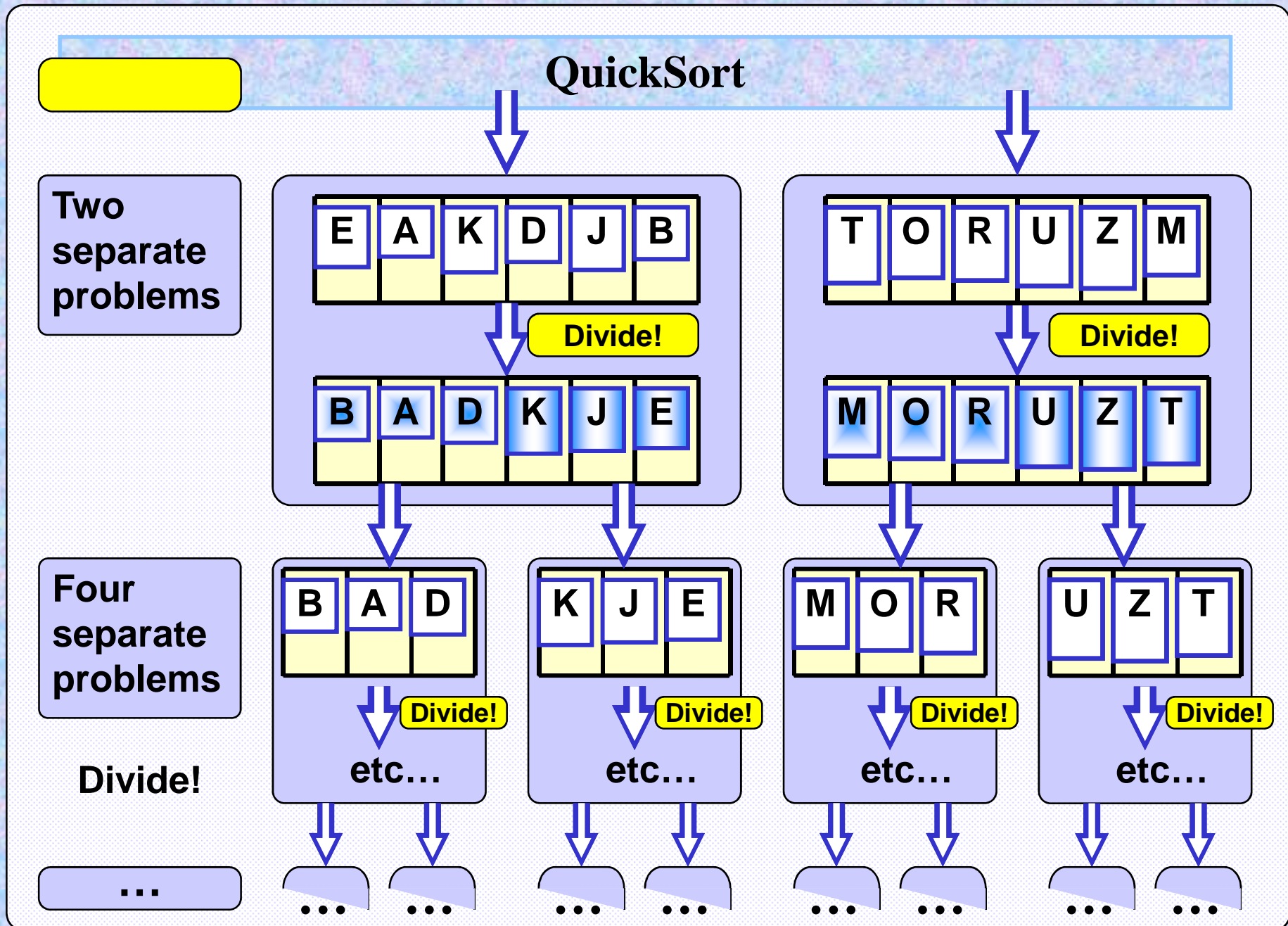
M A K D R B T O J U Z E

Divide!

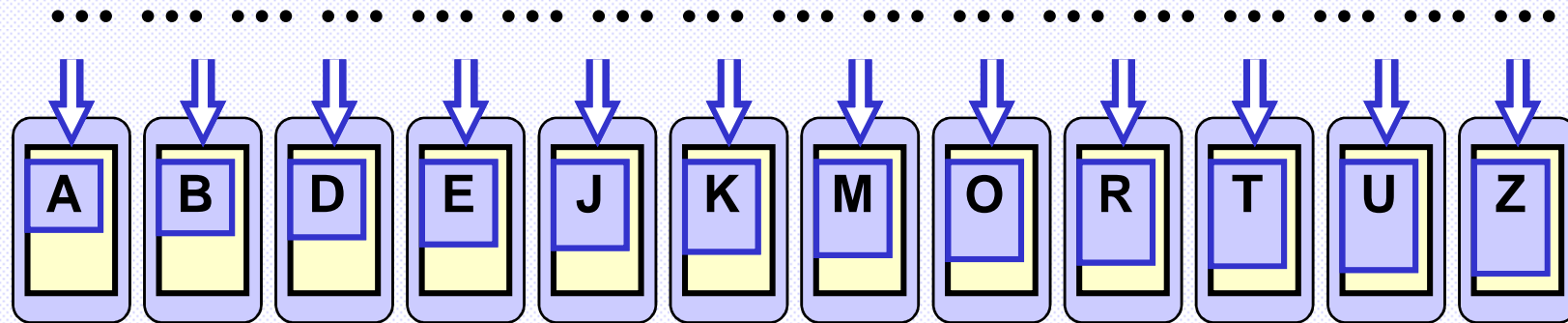
E A K D J B T O R U Z M

Small

Big



QuickSort

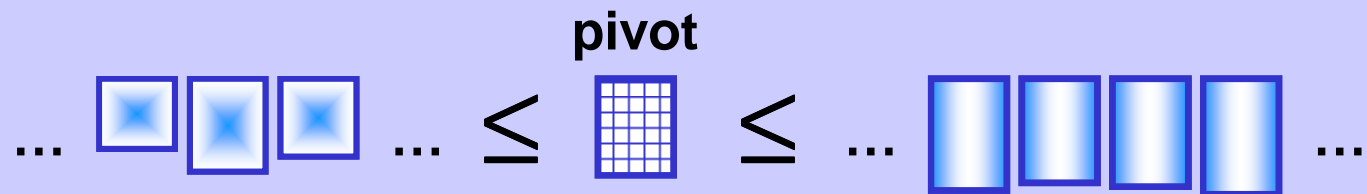


Conquered!

QuickSort

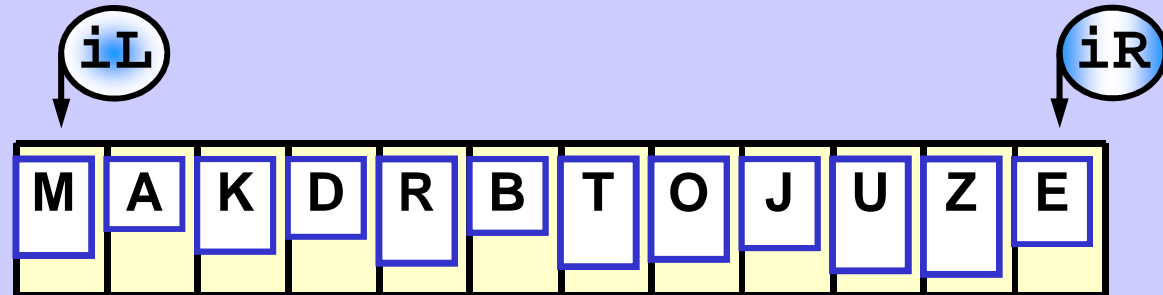
Partitioning

pivot



Init

Pivot

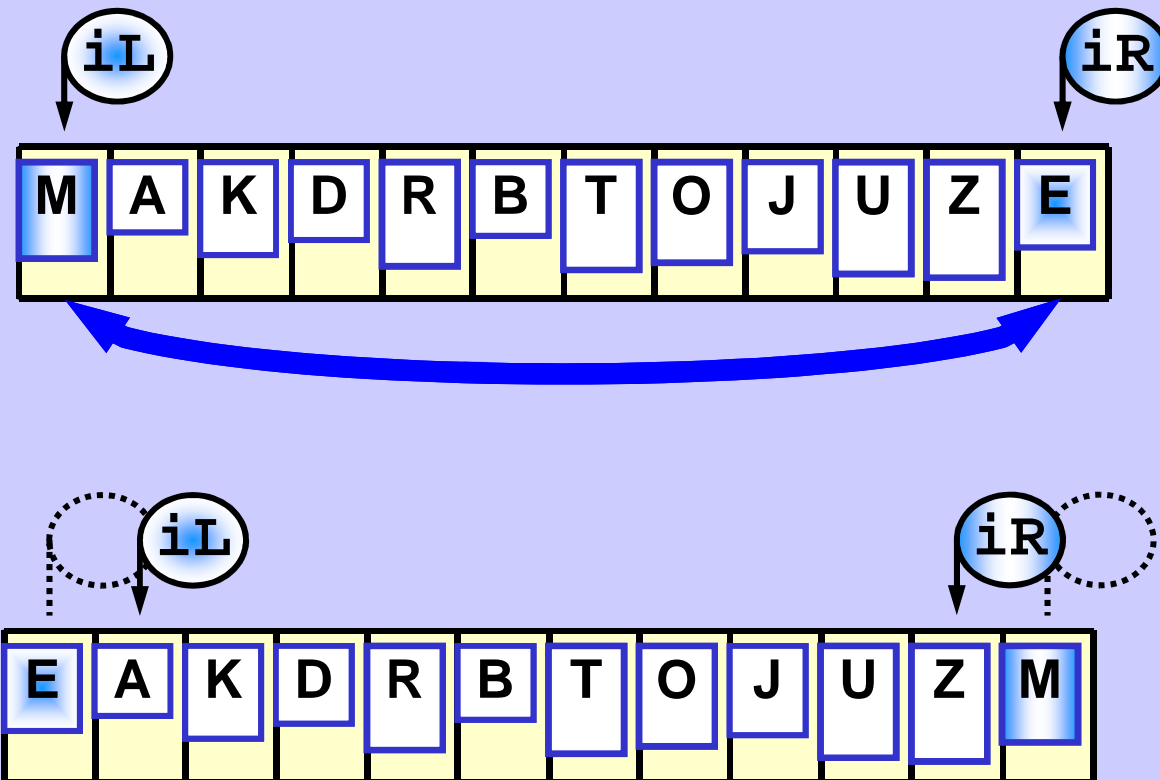


QuickSort

Partitioning

Step 1

Pivot

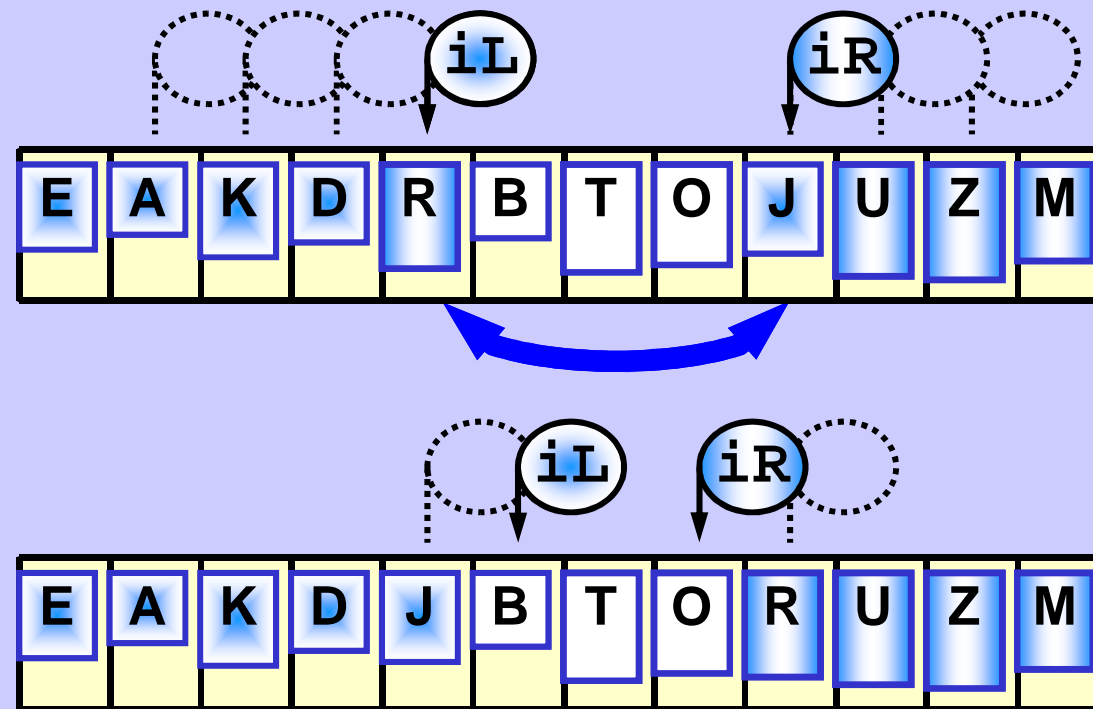


QuickSort

Partitioning

Step 2

Pivot

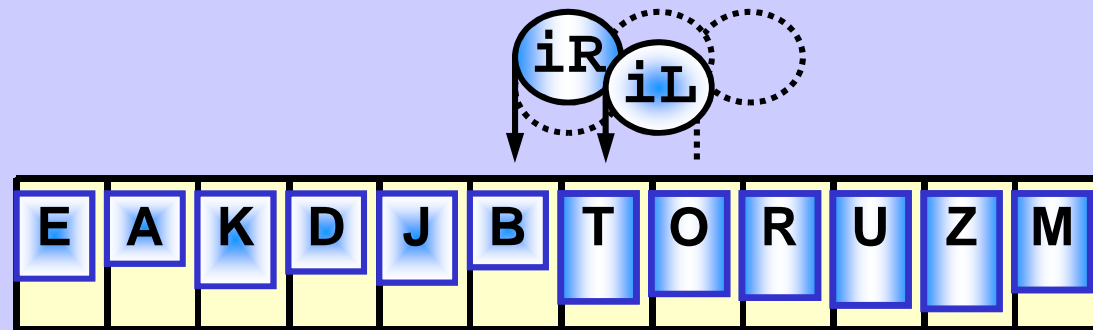
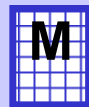


QuickSort

Partitioning

Step 3

Pivot

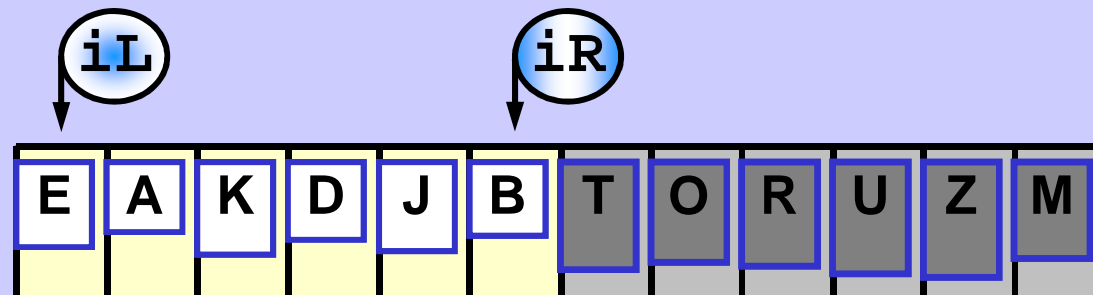


$iR < iL$ Stop

Divide!

Init

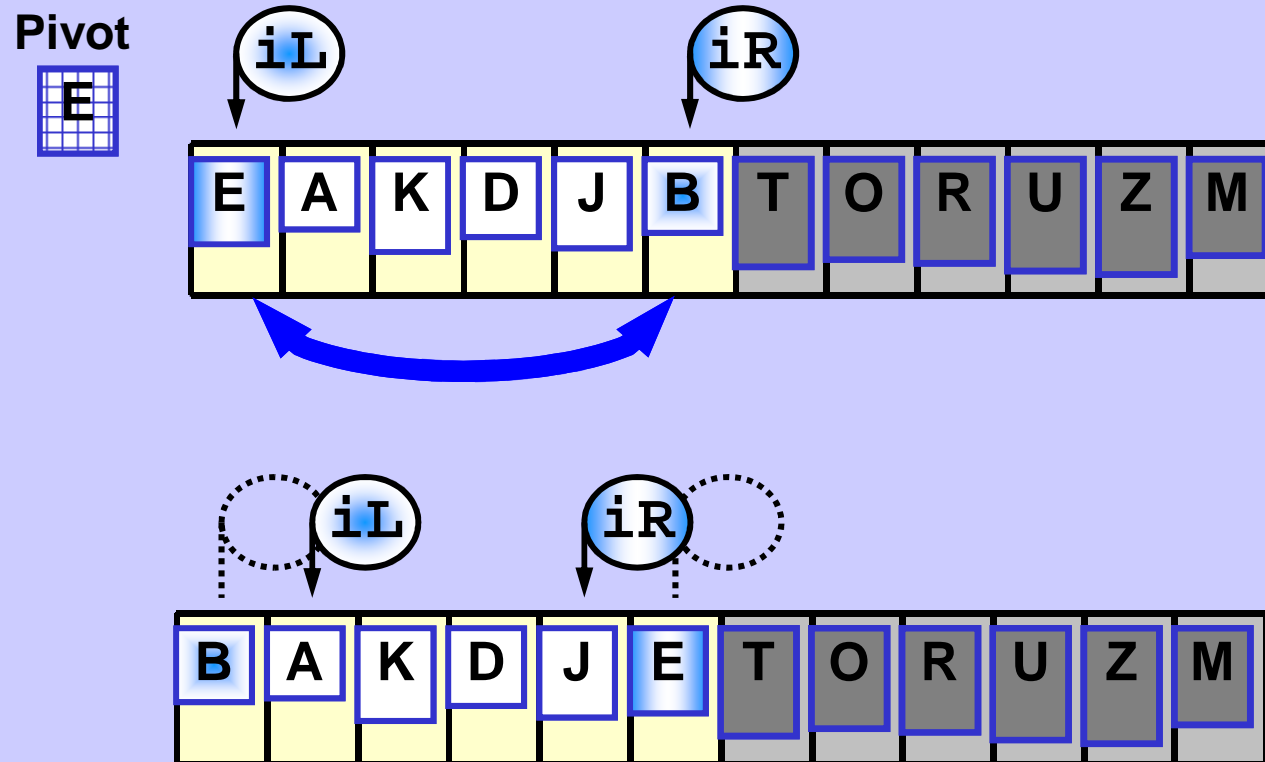
Pivot



QuickSort

Partitioning

Step 1



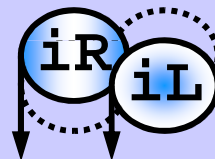
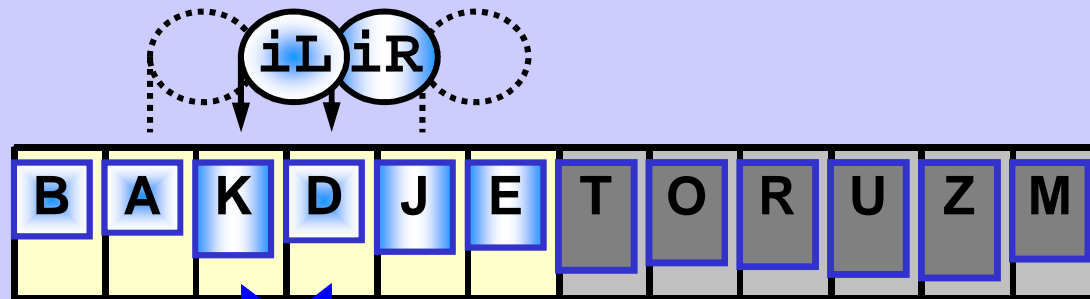
QuickSort

Partitioning

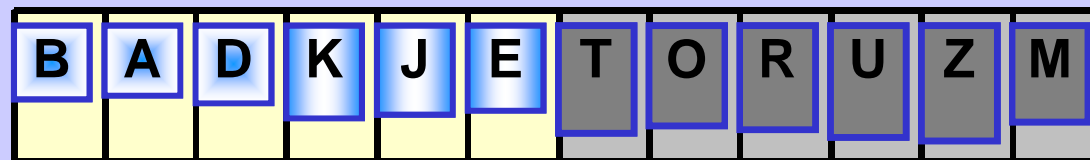
Step 2

Pivot

E



$iR < iL$ Stop

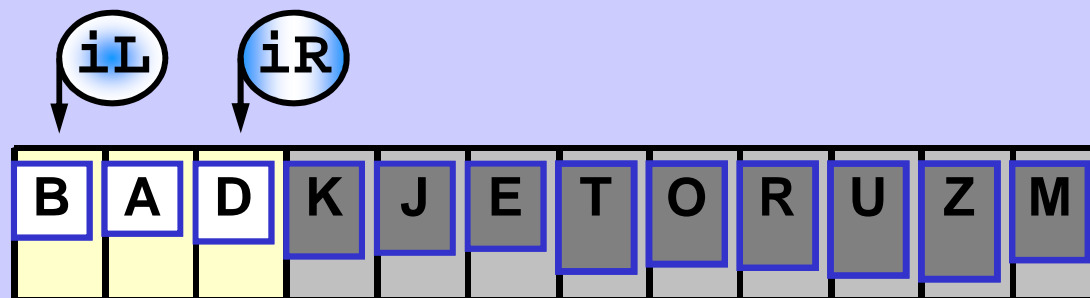


Divide!

Init

Pivot

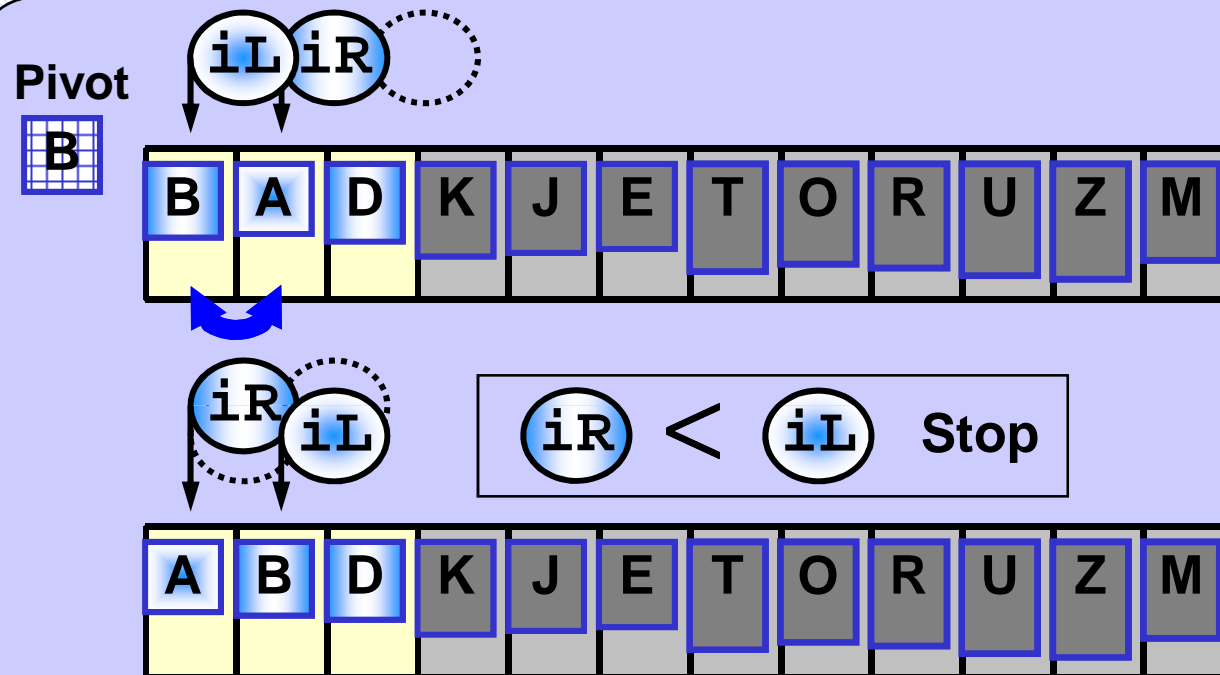
B



QuickSort

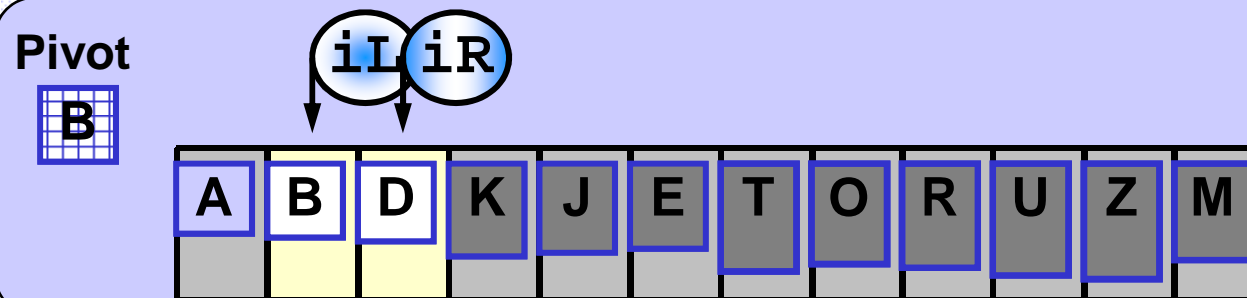
Partitioning

Step 1



Divide!

Init



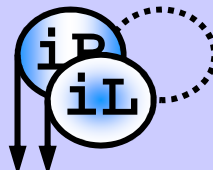
QuickSort

Partitioning

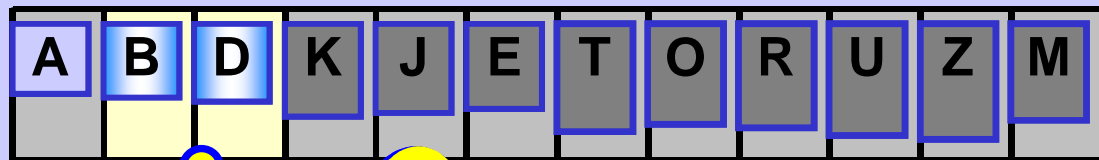
Step 1

Pivot

B



$iR == iL$



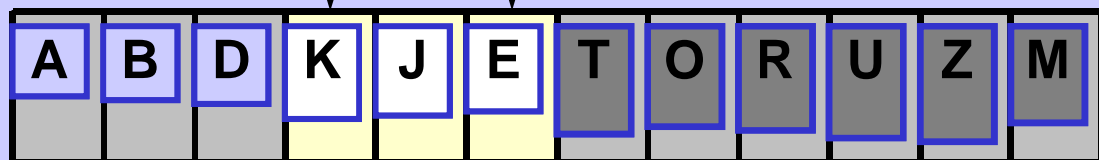
Next
Partition

Pivot

K

iL

iR



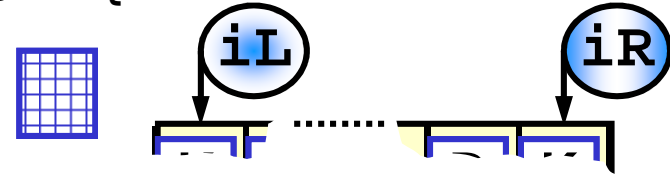
Init

etc...

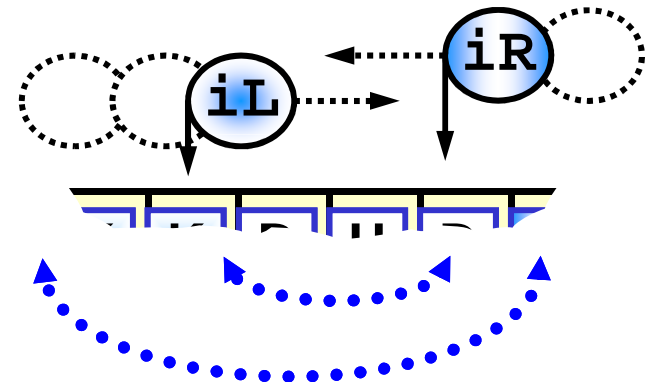
etc...

QuickSort

```
void qSort(Item a[], int low, int high) {
    int iL = low, iR = high;
    Item pivot = a[low];
```



```
do {
    while (a[iL] < pivot) iL++;
    while (a[iR] > pivot) iR--;
    if (iL < iR) {
        swap(a, iL, iR);
        iL++; iR--;
    }
    else {
        if (iL == iR) { iL++; iR--; }
    }
} while (iL <= iR);
```



```
if (low < iR) qSort(a, low, iR);
if (iL < high) qSort(a, iL, high);
}
```

Divide!

QuickSort

Levý index se nastaví na začátek zpracovávaného úseku pole, pravý na jeho konec, zvolí se pivot.

Cyklus:

Levý index se pohybuje doprava
a zastaví se na prvku větším nebo rovném pivotu.

Pravý index se pohybuje doleva
a zastaví se na prvku menším nebo rovném pivotu.

Pokud je levý index ještě před pravým,
příslušné prvky se prohodí,
a oba indexy se posunou o 1 ve svém směru.

Jinak pokud se indexy rovnají,
jen se oba posunou o 1 ve svém směru.

Cyklus se opakuje, dokud se indexy nepřekříží,
tj. pravý se dostane před levý.

Pak nastává rekurzivní volání
na úsek od začátku do pravého(!) indexu včetně
a na úsek od levého(!) indexu včetně až do konce,
má-li příslušný úsek délku větší než 1.

QuickSort

Asymptotic complexity

Tests and moves
total

$$\Theta(n \cdot \log_2(n))$$

best case

$$\Theta(n \cdot \log_2(n))$$

average case

$$\Theta(n^2)$$

worst case

Asymptotic worst case complexity of QuickSort is $O(n^2)$ (!!)

... but! :

“Average” complexity of QuickSort is $\Theta(n \cdot \log_2(n))$ (!!)

Sorting

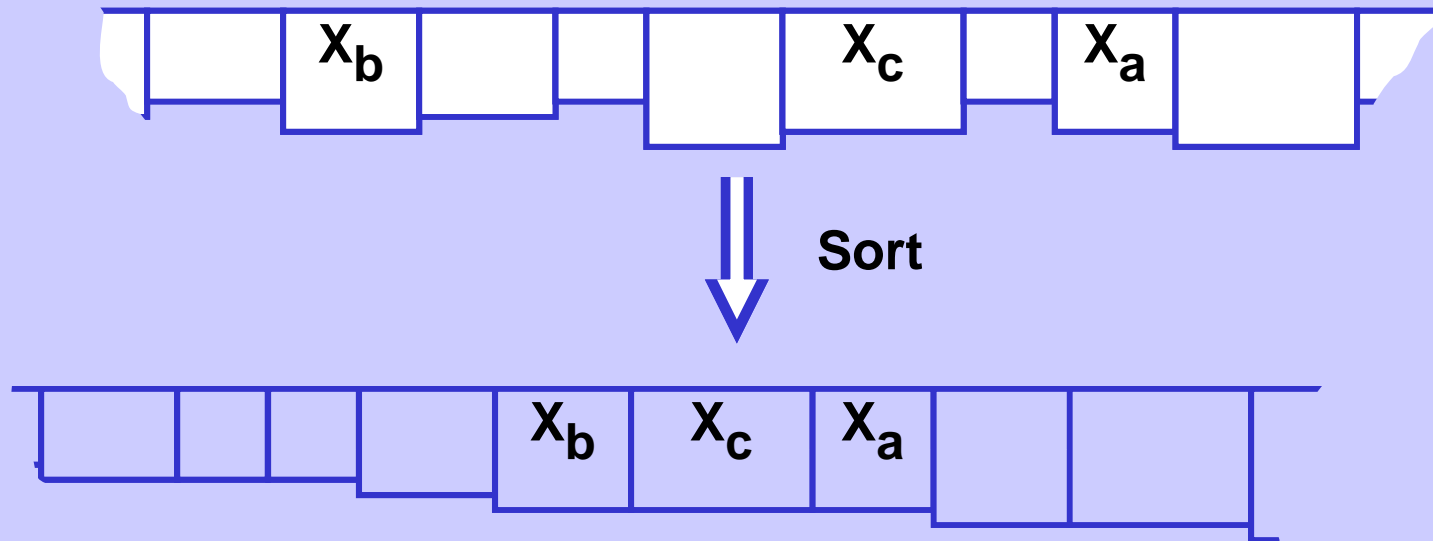


Effectivity comparison



N	N^2	$N \times \log_2(N)$	$\frac{N^2}{N \times \log_2(N)}$	deceleration (1~1sec)
1	1	0		
10	100	33.2	3.0	3 sec
100	10 000	6 64.4	15.1	15 sec
1 000	1 000 000	9 965.8	100.3	1.5 min
10 000	100 000 000	132 877.1	752.6	13 min
100 000	10 000 000 000	1 660 964.0	6 020.6	1.5 hrs
1 000 000	1 000 000 000 000	19 931 568.5	50 171.7	14 hrs
10 000 000	100 000 000 000 000	232 534 966.6	430 042.9	5 days

Sort stability



Stable sort does not change the **(relative)** order of elements with equal value.

Stabilní řazení nemění **(relativní)** pořadí prvků se stejnou hodnotou.

Sort stability

B₁ D₁ C₁ A₁ C₂ B₂ A₂ D₂ B₃ A₃ D₃ C₃

Select
Insert
Bubble

-- Stable
implementation



A₁ A₂ A₃ B₁ B₂ B₃ C₁ C₂ C₃ D₁ D₂ D₃

B₁ D₁ C₁ A₁ C₂ B₂ A₂ D₂ B₃ A₃ D₃ C₃

Select
Insert
Bubble

-- Unstable
implementation



QuickSort -- Always!!

A₂ A₁ A₃ B₂ B₃ B₁ C₃ C₁ C₂ D₃ D₂ D₁

Sorting

**Různé algoritmy
mají
různou složitost**

Sorting

**The complexity
of different algorithms
varies**