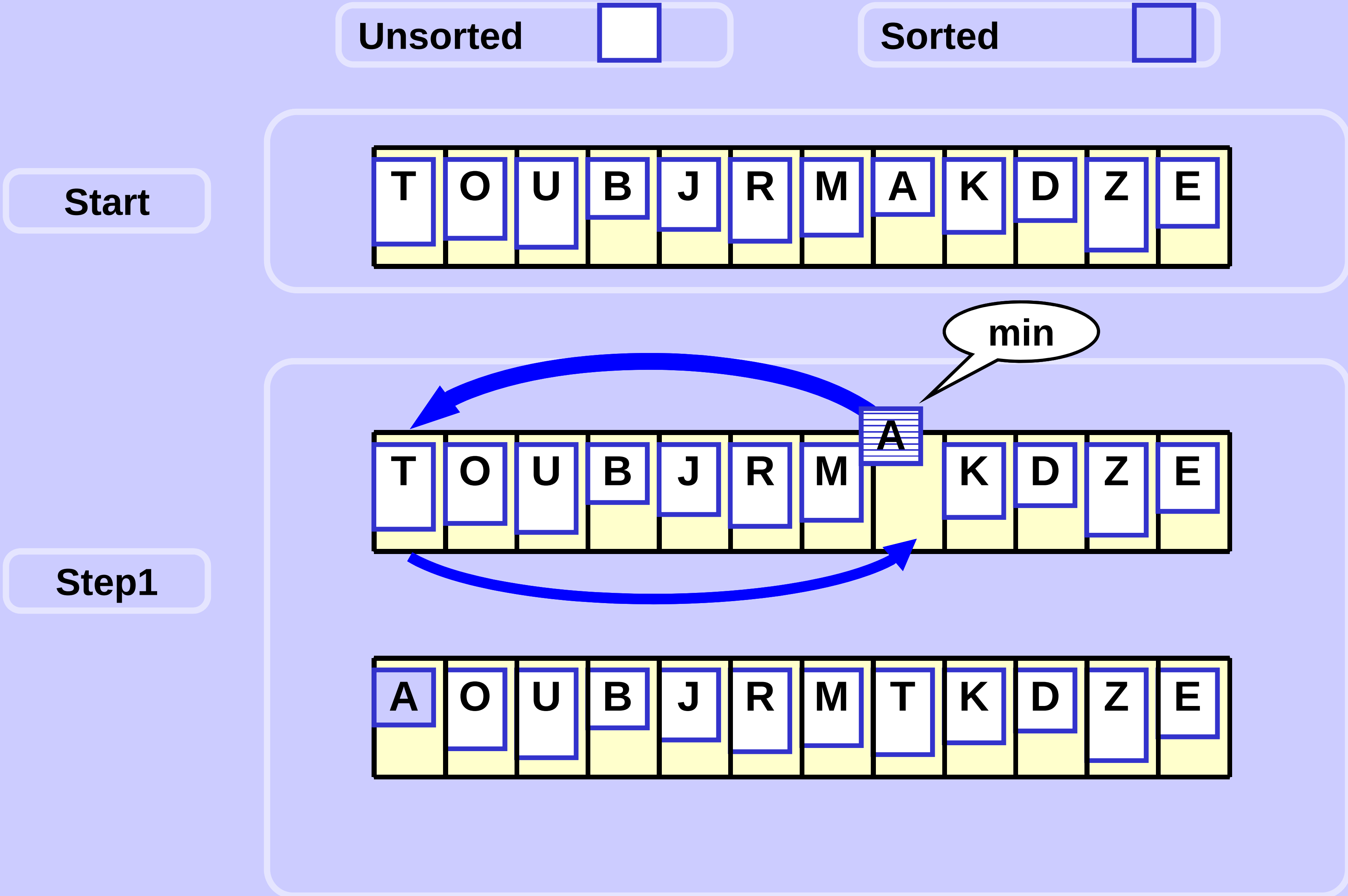
ALG 07

Selection sort (Select sort)
Insertion sort (Insert sort)
Bubble sort deprecated

Quicksort

Sort stability

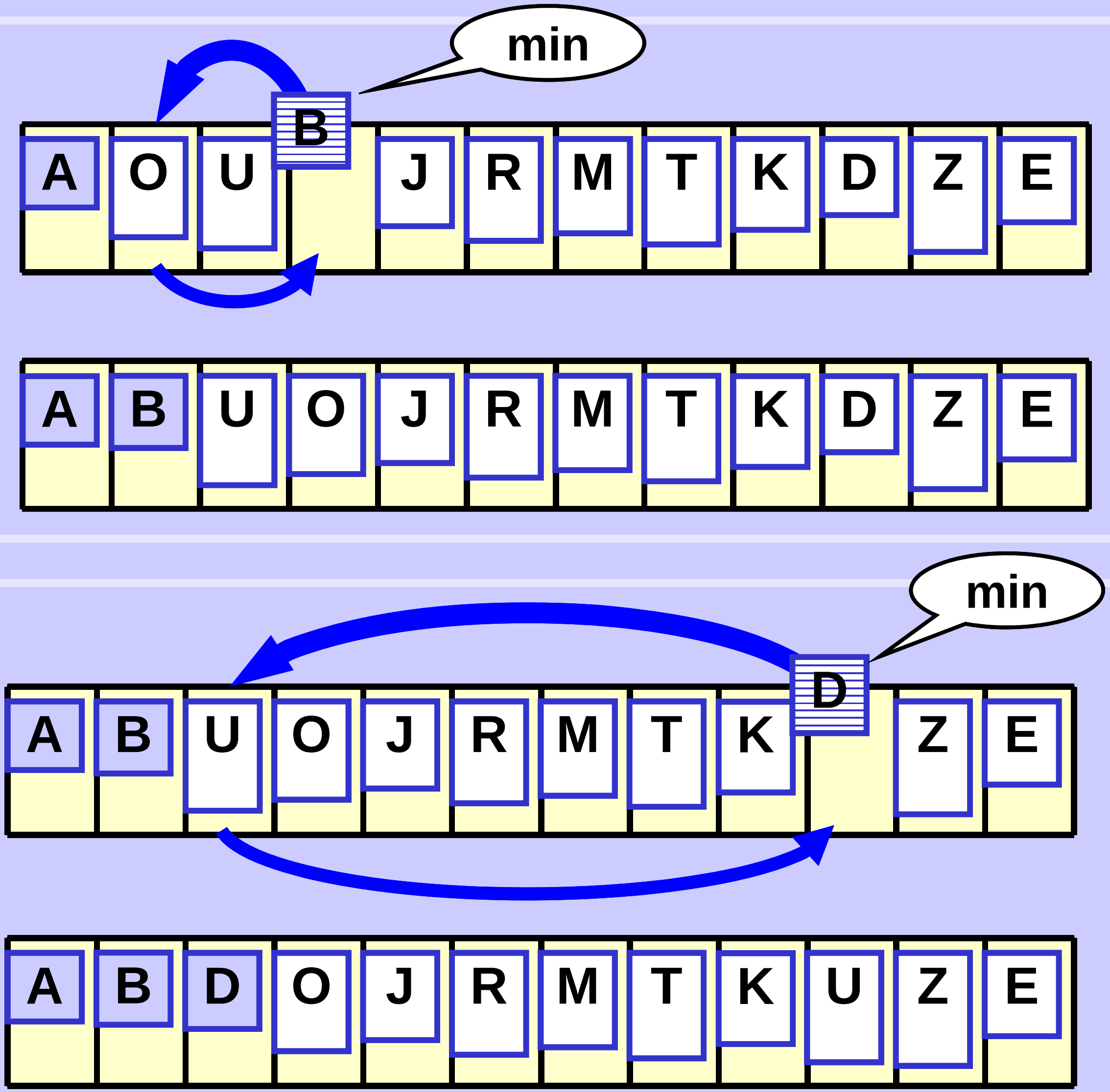
Selection sort



Selection sort

Step 2

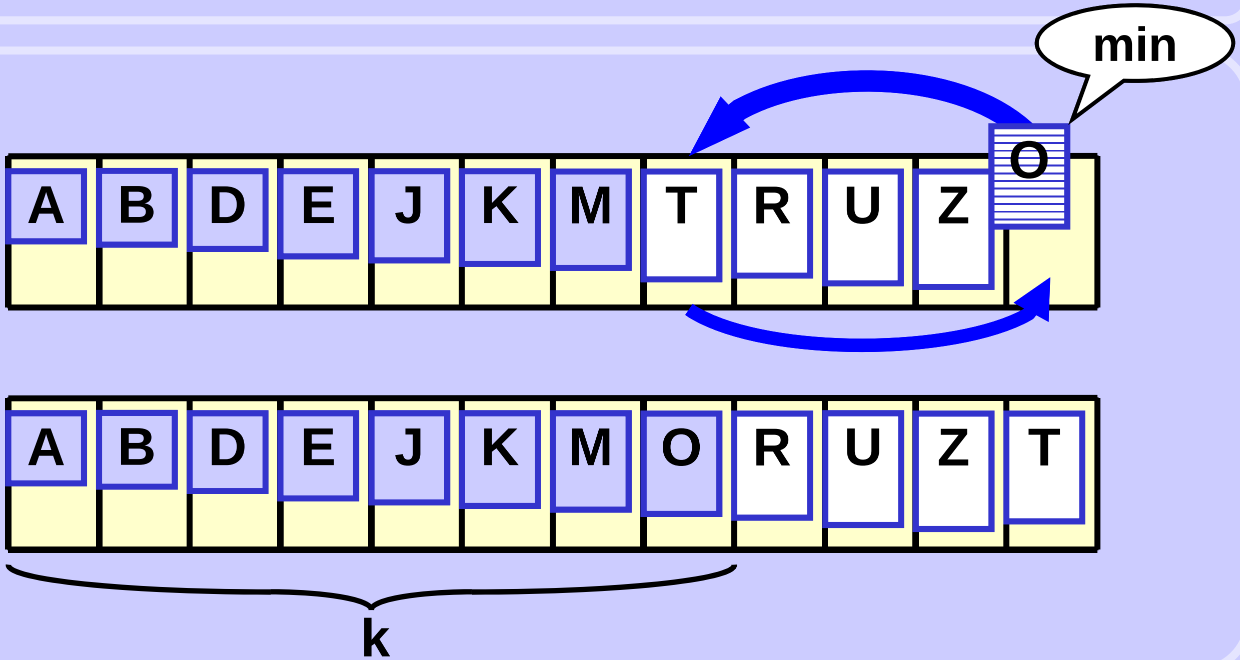
Step 3



Selection sort

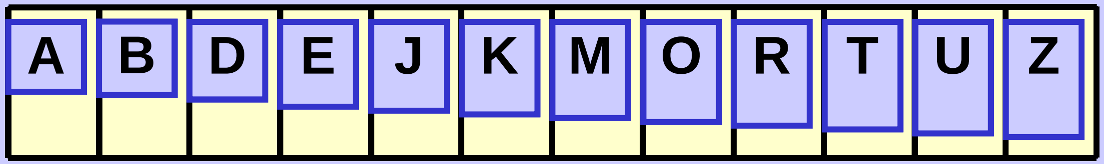
. . .

Step k



. . .

All sorted

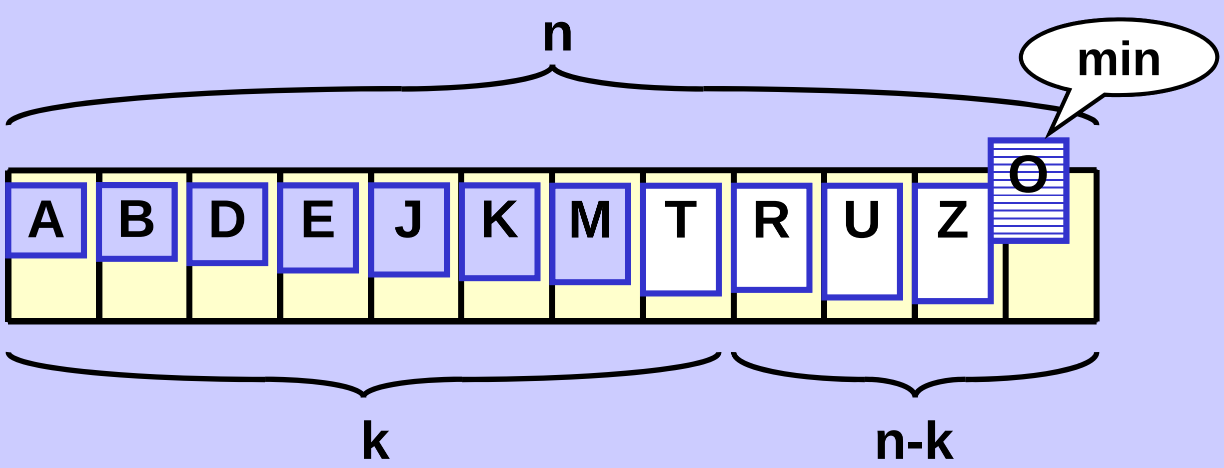


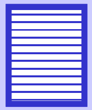
Selection sort

```
def selectSort( arr ):
    n = len( arr )
    for i in range( n-1 ):
        jmin = i
        # select minimum
        for j in range( i+1, n ):
            if arr[j] < arr[jmin]:
                jmin = j
        # put minimum to its place
        swap( arr, i, jmin )
```

Selection sort

Step k



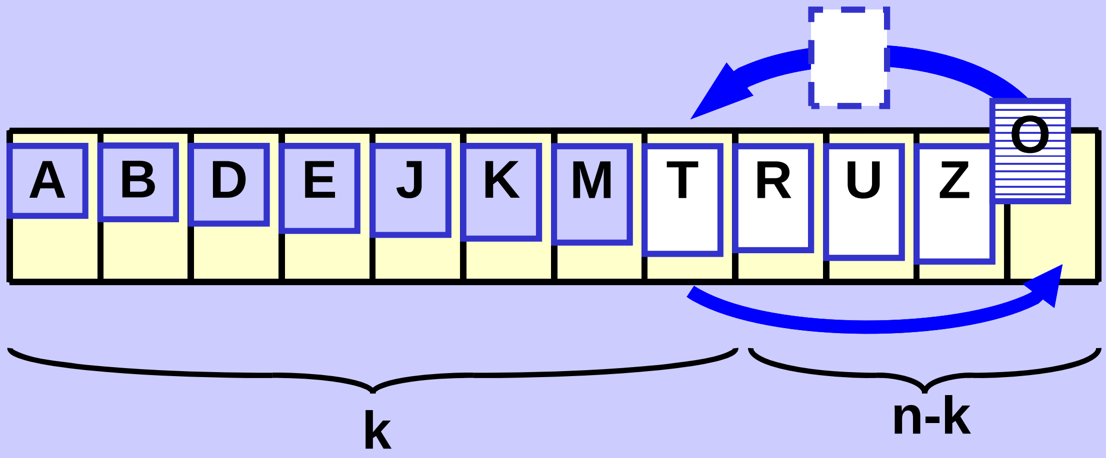
Select minimum  ……. **(n-k) tests**

Tests total

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

Selection sort

Step k



Moves **3**

Moves total

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

Selection sort

Resume

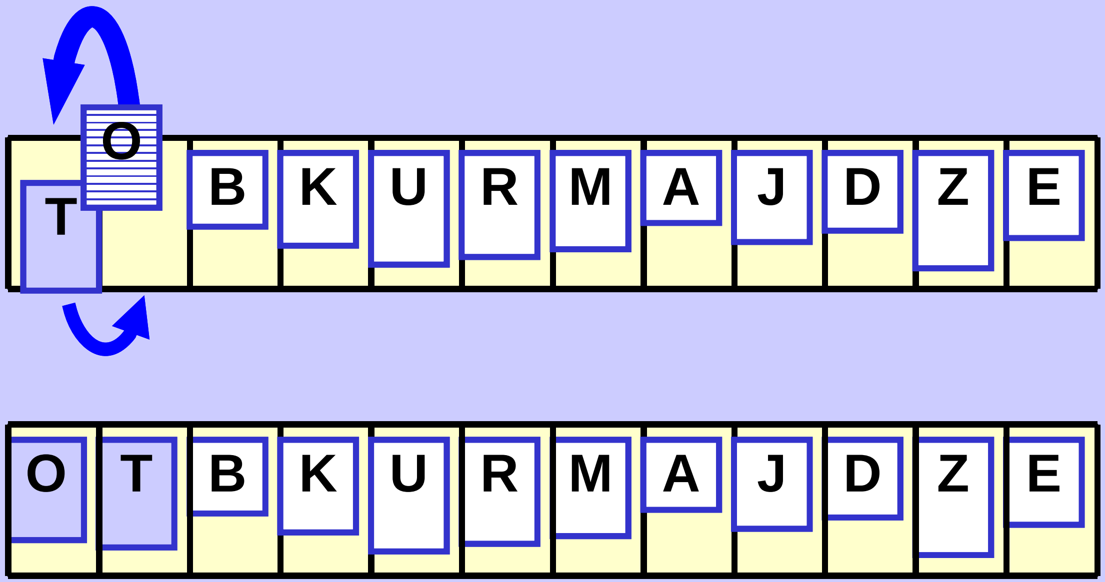
Tests total	$\frac{1}{2}(n^2 - n) \quad = \Theta(n^2)$
Moves total	$3(n-1) \quad = \Theta(n)$
Oerations total	$\frac{1}{2}(n^2 - n) \ + \ 3(n-1) \quad = \Theta(n^2)$

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

Insertion sort

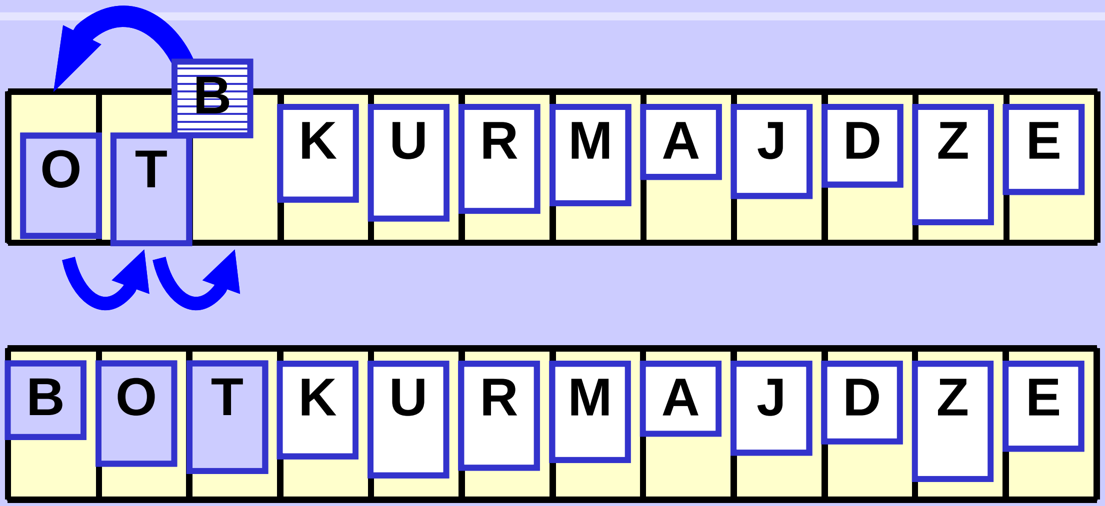
Start

Step 1

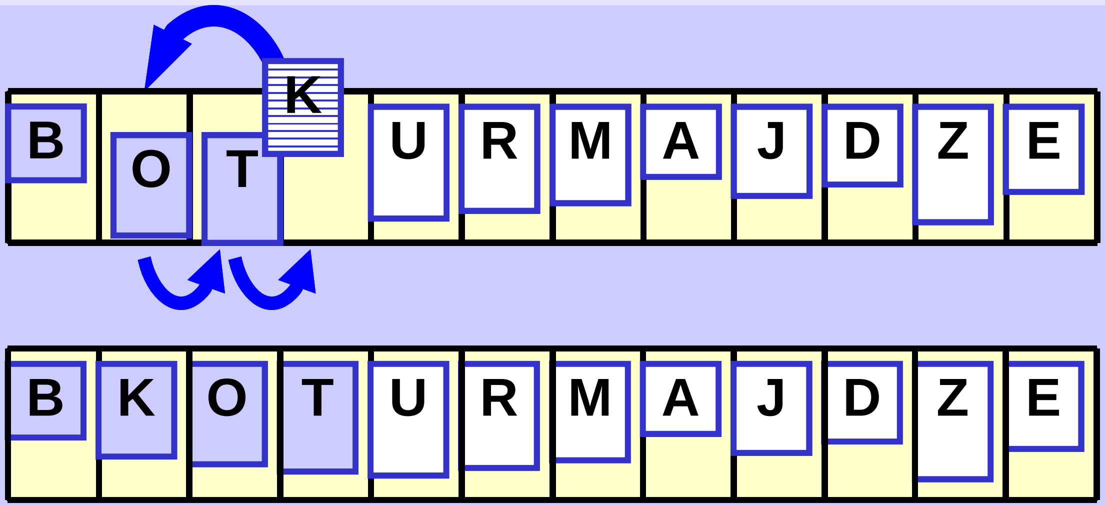


Insertion sort

Step 2



Step 3

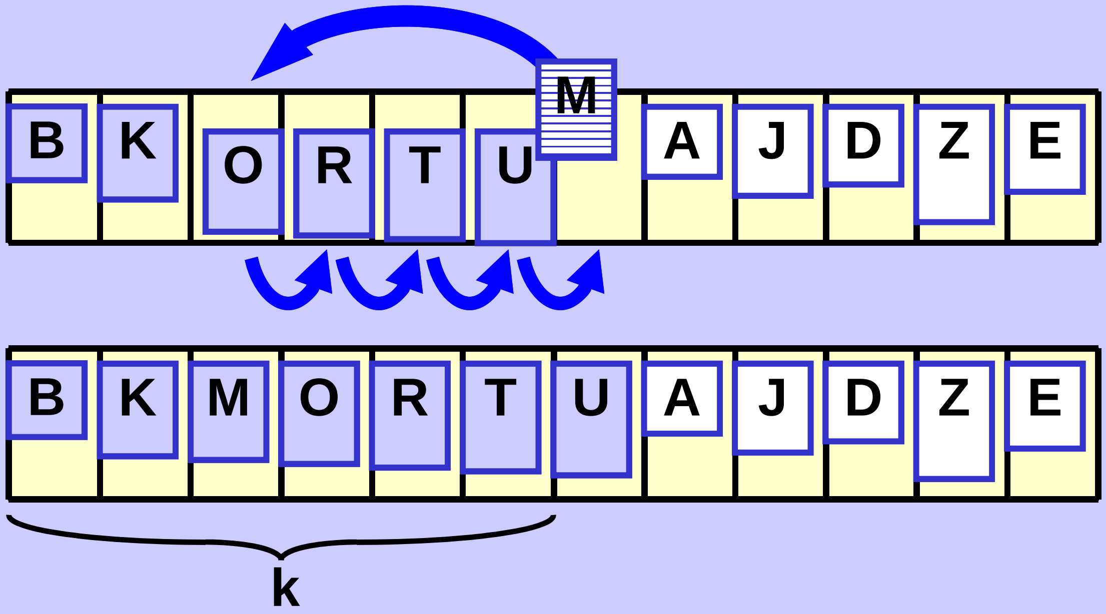


Insertion sort

. . .

• • •

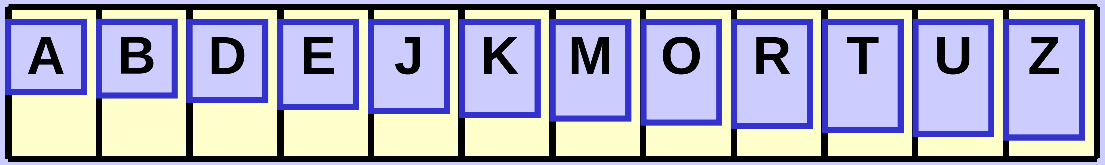
Step k



. . .

• • •

All sorted

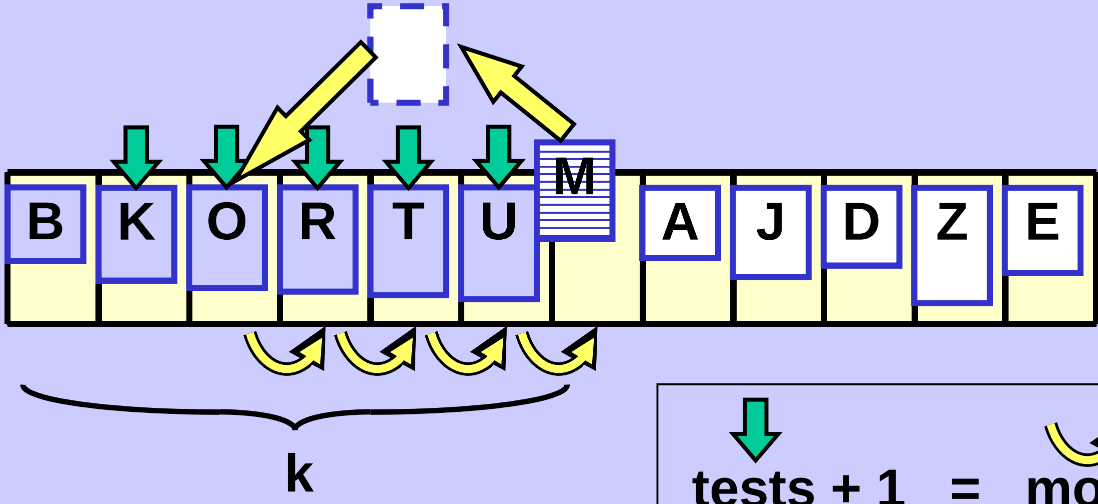


Insertion sort

```
def insertSort( arr ):
    n = len( arr )
    for i in range( 1, n ):
        # find & make place for arr[i]
        insVal = arr[i]
        j = i-1
        while j >= 0 and arr[j] > insVal:
            arr[j+1] = arr[j]
            j -= 1;
        # insert arr[i] to the correct place
        arr[j+1] = insVal
```

Insertion sort

Step k



tests + 1 = moves

tests …….	1	best case
	k	worst case
	(k+1)/2	“average” case

moves …….	2	best case
	k+1	worst case
	(k+3)/2	“average” case

Insertion sort

Resume

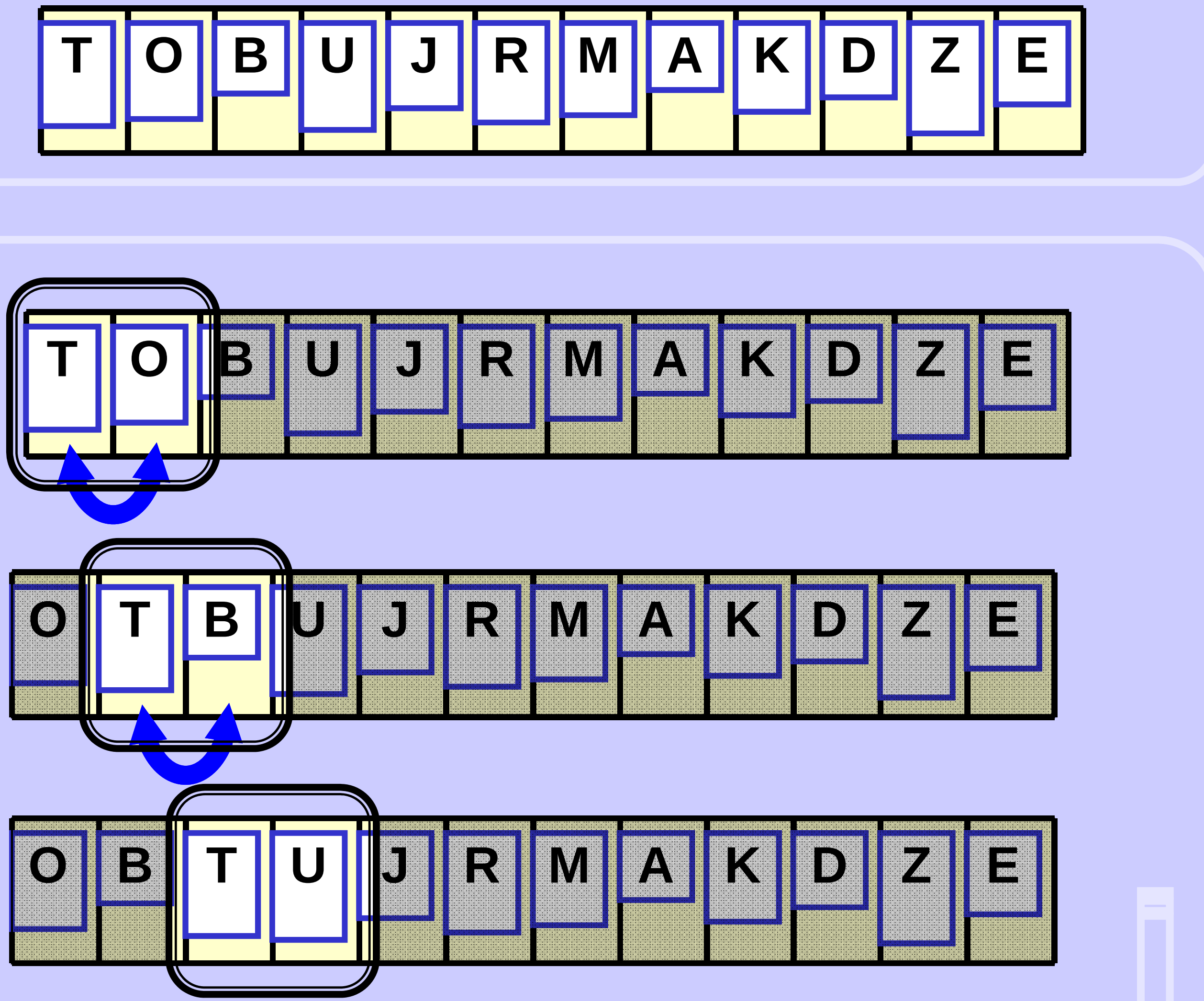
Tests total	$n - 1$	$= \Theta(n)$	best case
	$(n^2 - n)/2$	$= \Theta(n^2)$	worst case
	$(n^2 + n - 2)/4$	$= \Theta(n^2)$	"average" case
Moves total	$2n - 2$	$= \Theta(n)$	best case
	$(n^2 + n - 2)/2$	$= \Theta(n^2)$	worst case
	$(n^2 + 5n - 6)/4$	$= \Theta(n^2)$	"average" case

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

Bubble sort

Start

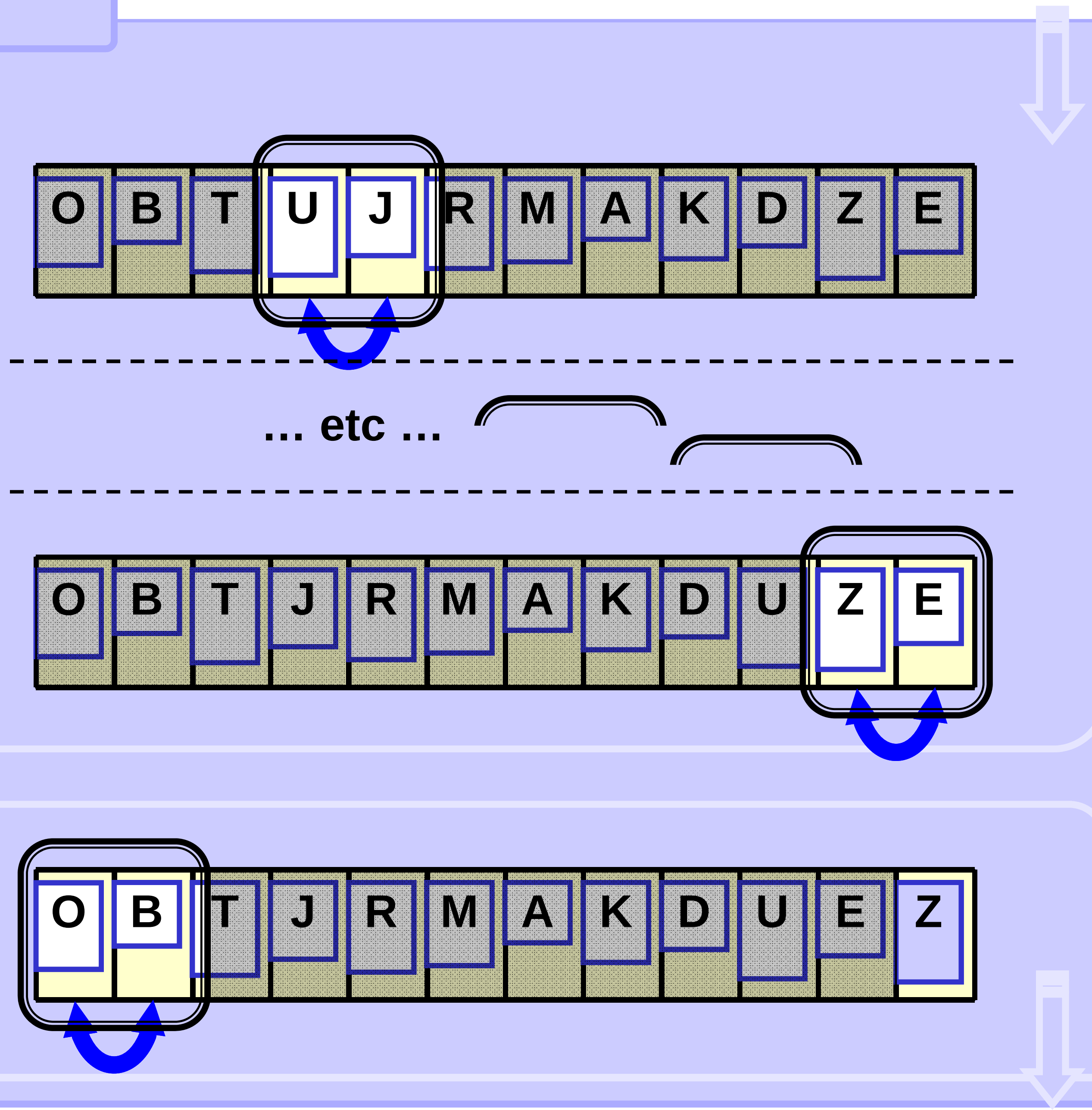
Phase 1



Bubble sort

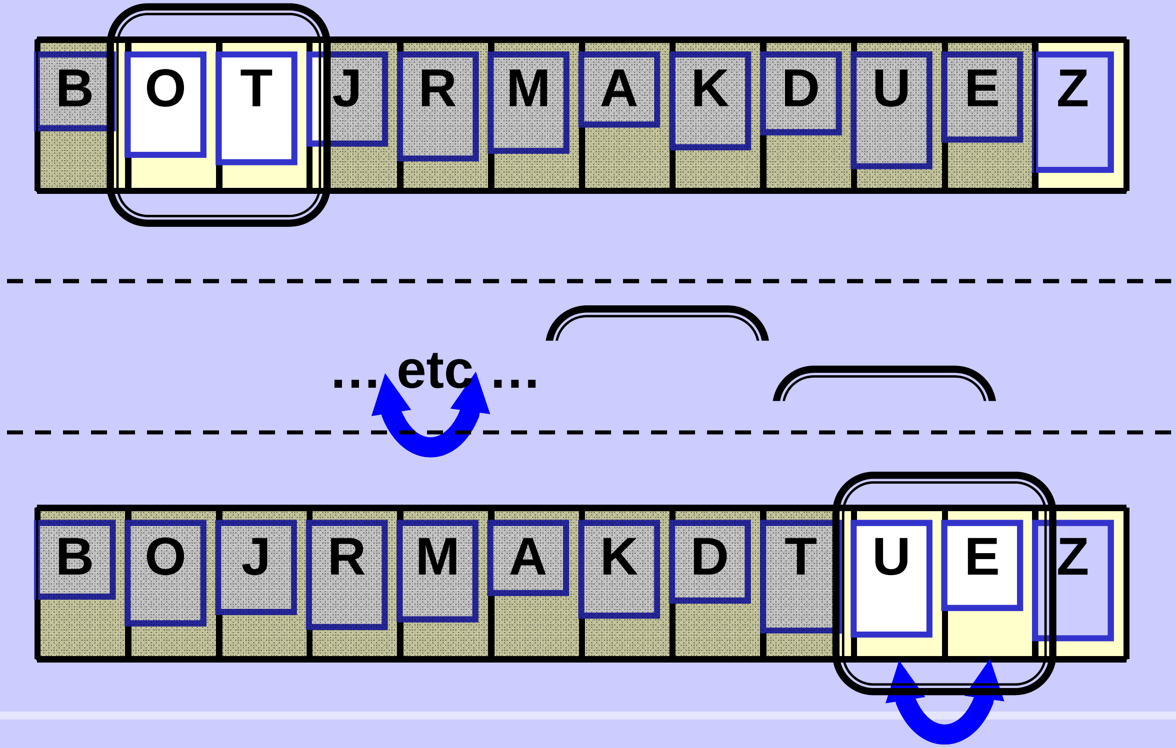
Phase 1

Phase 2

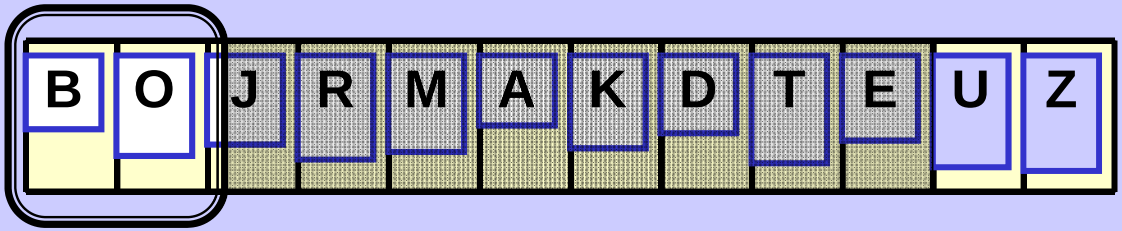


Bubble sort

Phase 2



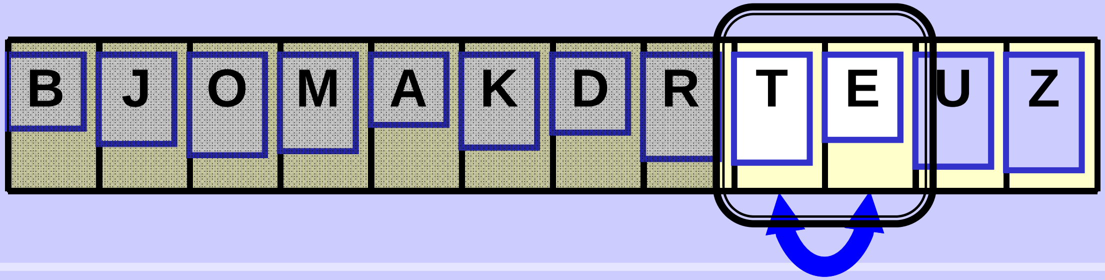
Phase 3



Bubble sort

Phase 3

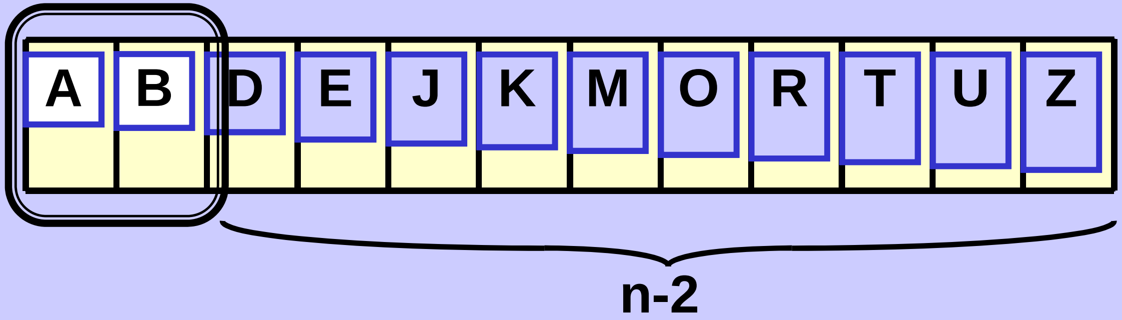
... etc ...



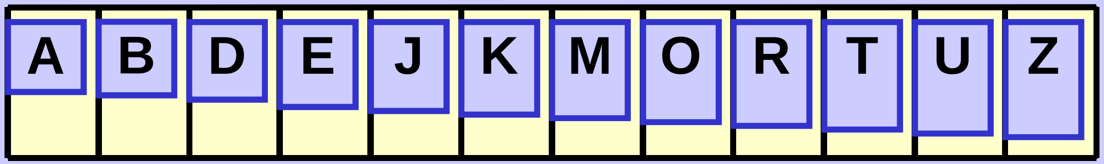
. . .

• • •

Phase n-1



All sorted



Bubble sort

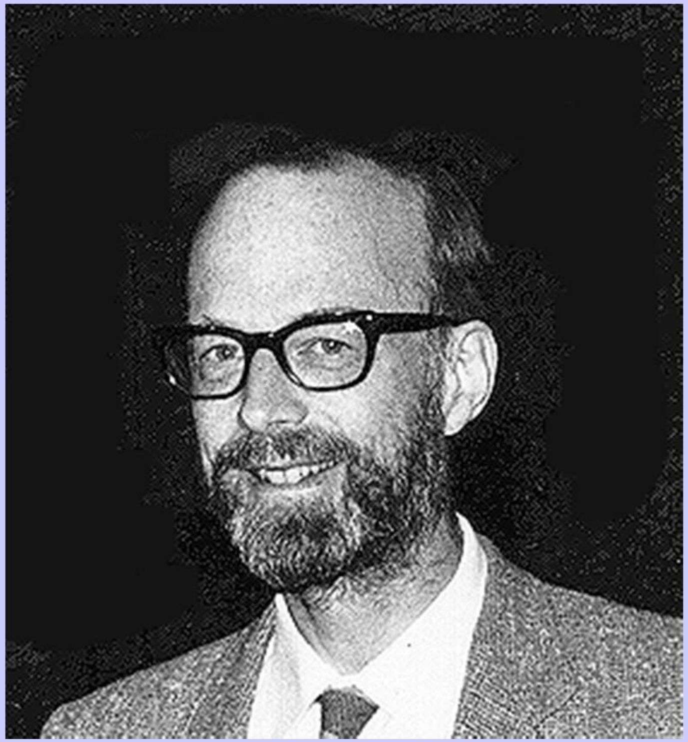
```
#decreasing lastPos
for lastPos in range( len(arr)-1, -1, -1 ):
   for j in range( lastPos ):
      if arr[j] > arr[j+1]: swap( arr, j, j+1 )
```

Resume

Tests total	$(n-1) + (n-2) + \ldots + 2 + 1 = \frac{1}{2}(n^2 - n) = \Theta(n^2)$
Moves total	$0 = \Theta(1)$ best case $\frac{1}{2}(n^2 - n) = \Theta(n^2)$ worst case

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

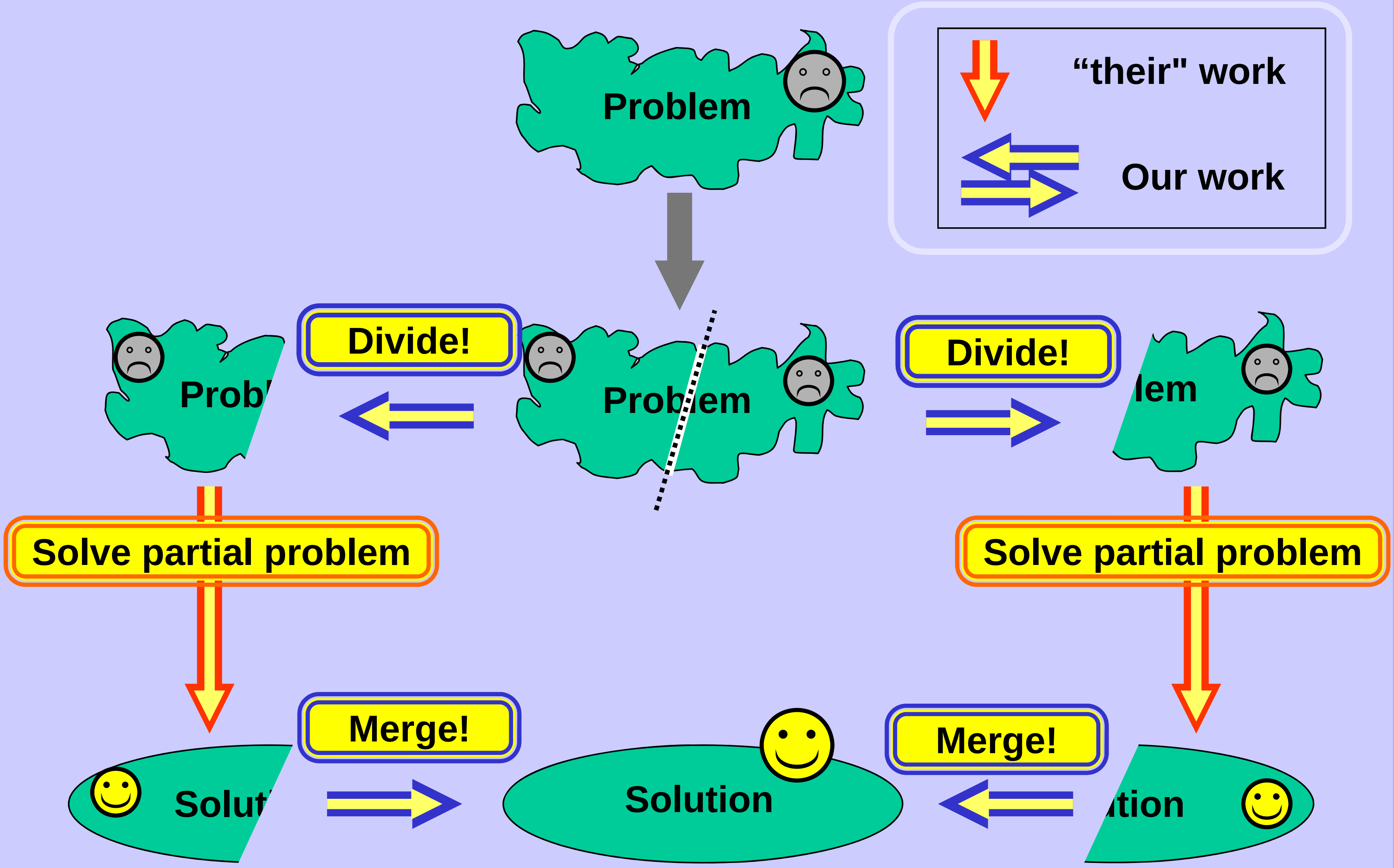
Quicksort



Sir Charles Antony Richard Hoare

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

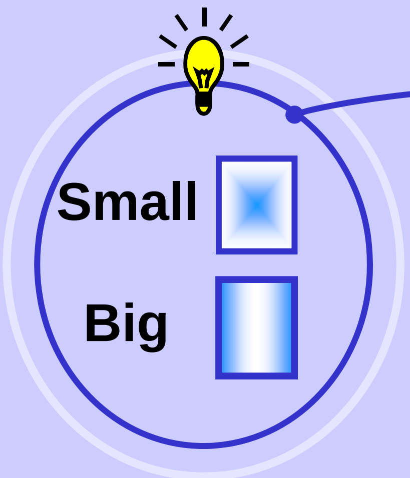
Divide and conquer! Divide et impera!

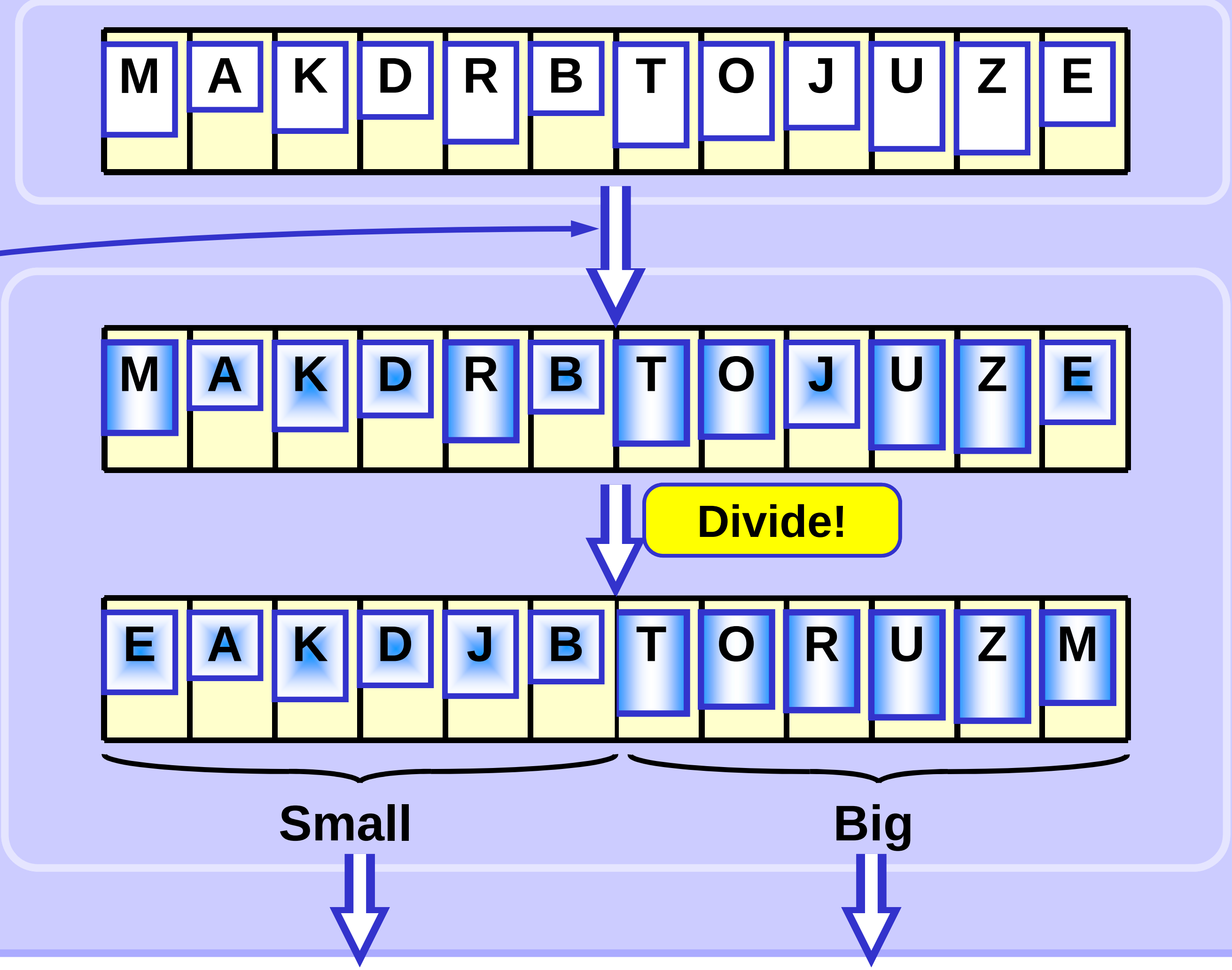


Quicksort

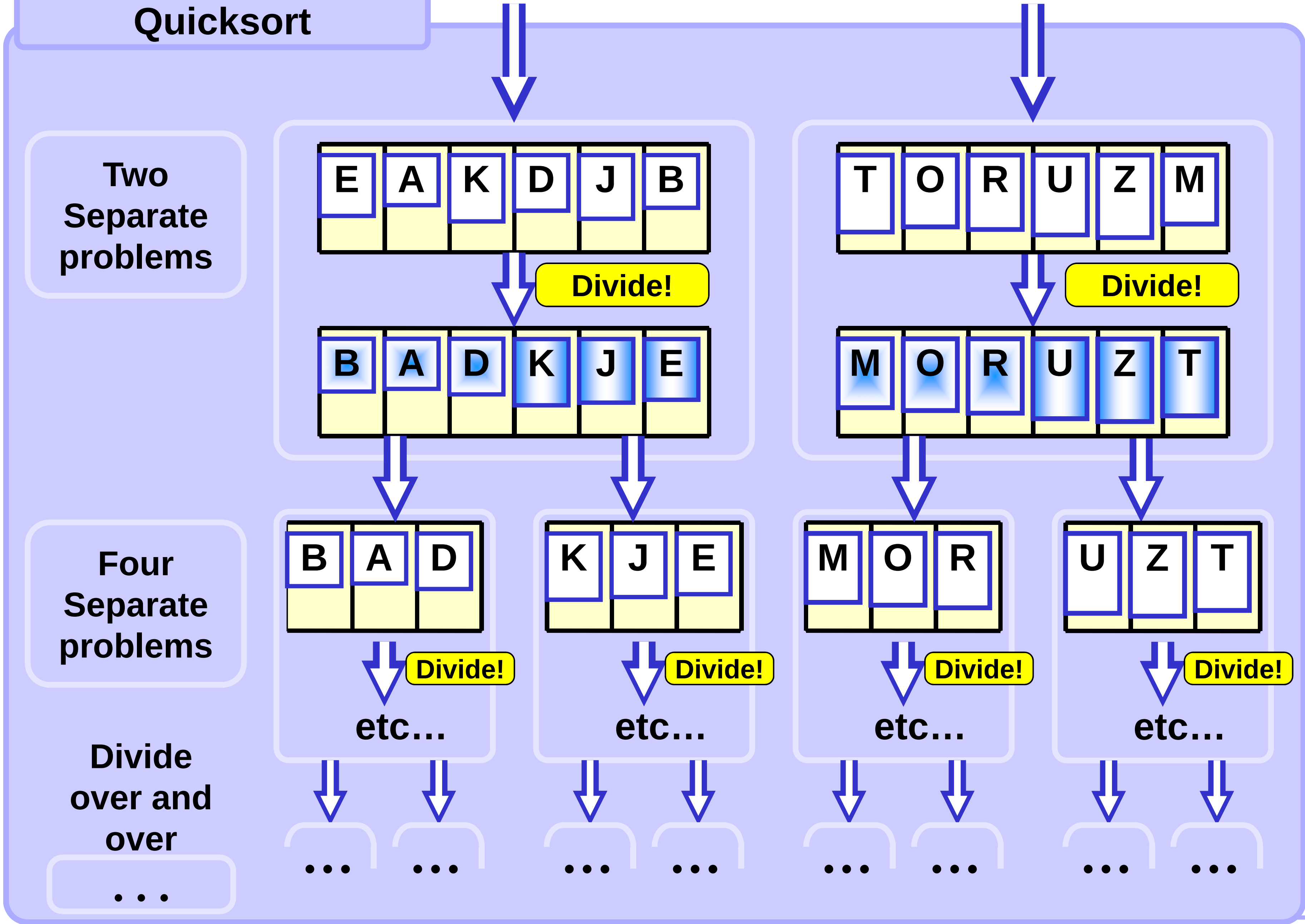
The idea — **Divide & Conquer!**

Start

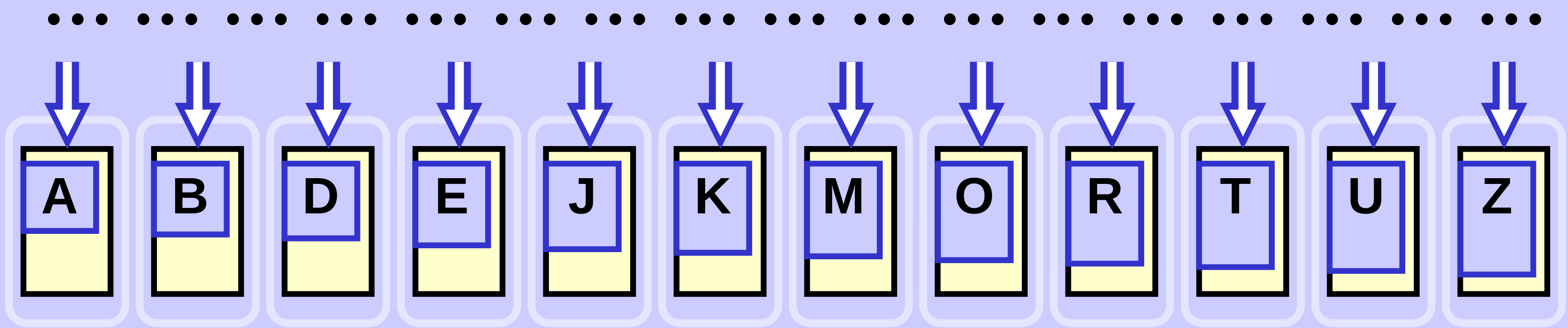




Quicksort


Two Separate problems
E A K D J B
T O R U Z M
Divide!
Divide!
B A D K J E
M O R U Z T
Four Separate problems
B A D
K J E
M O R
U Z T
Divide!
Divide!
Divide!
Divide!
etc…
etc…
etc…
etc…
Divide over and over
. . .

Quicksort

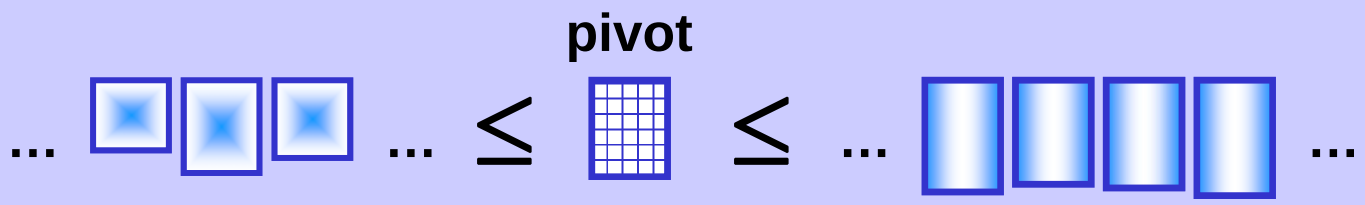


Conquered!

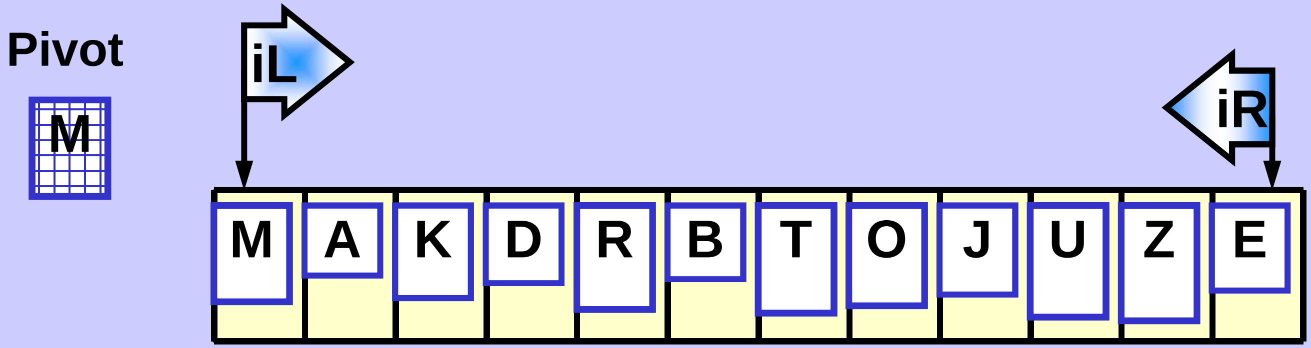
Quicksort

Dividing

Pivot



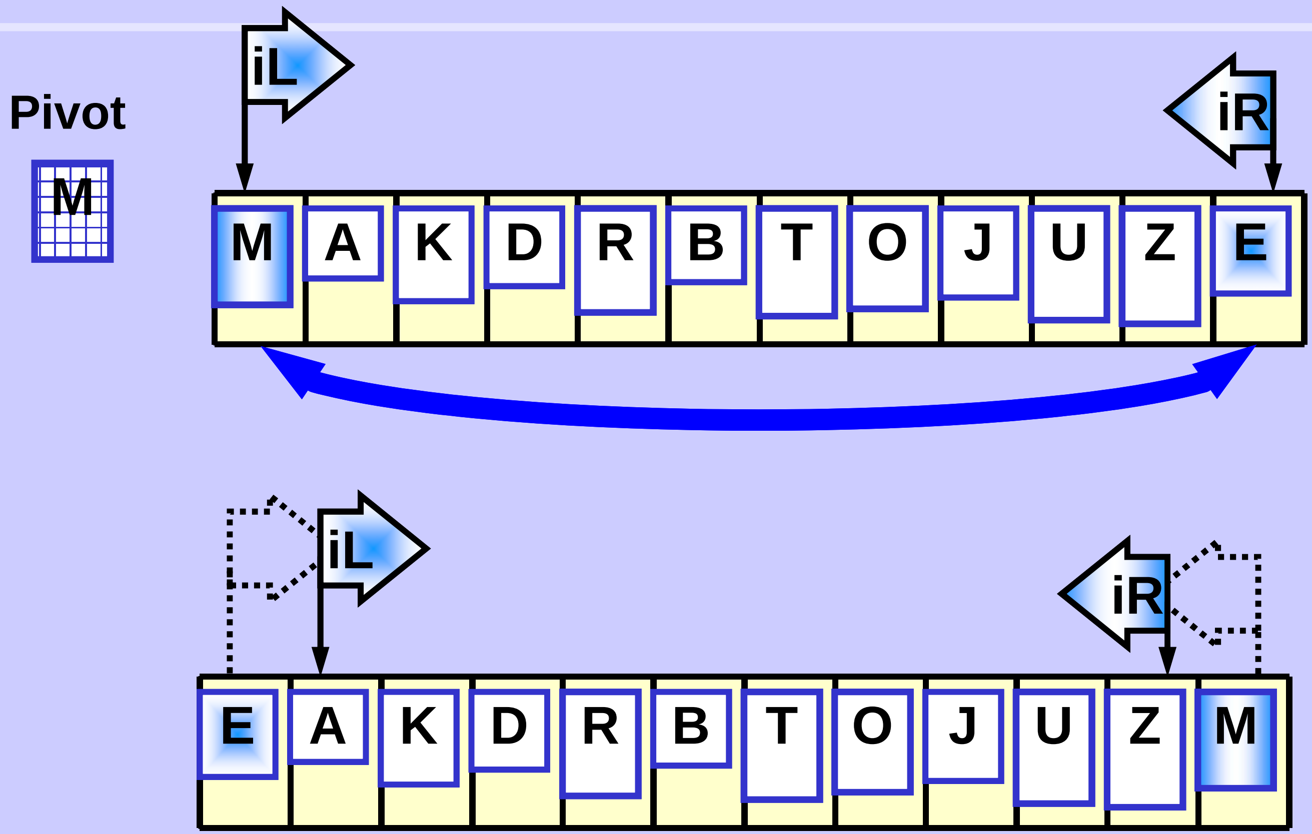
Init



Quicksort

Dividing

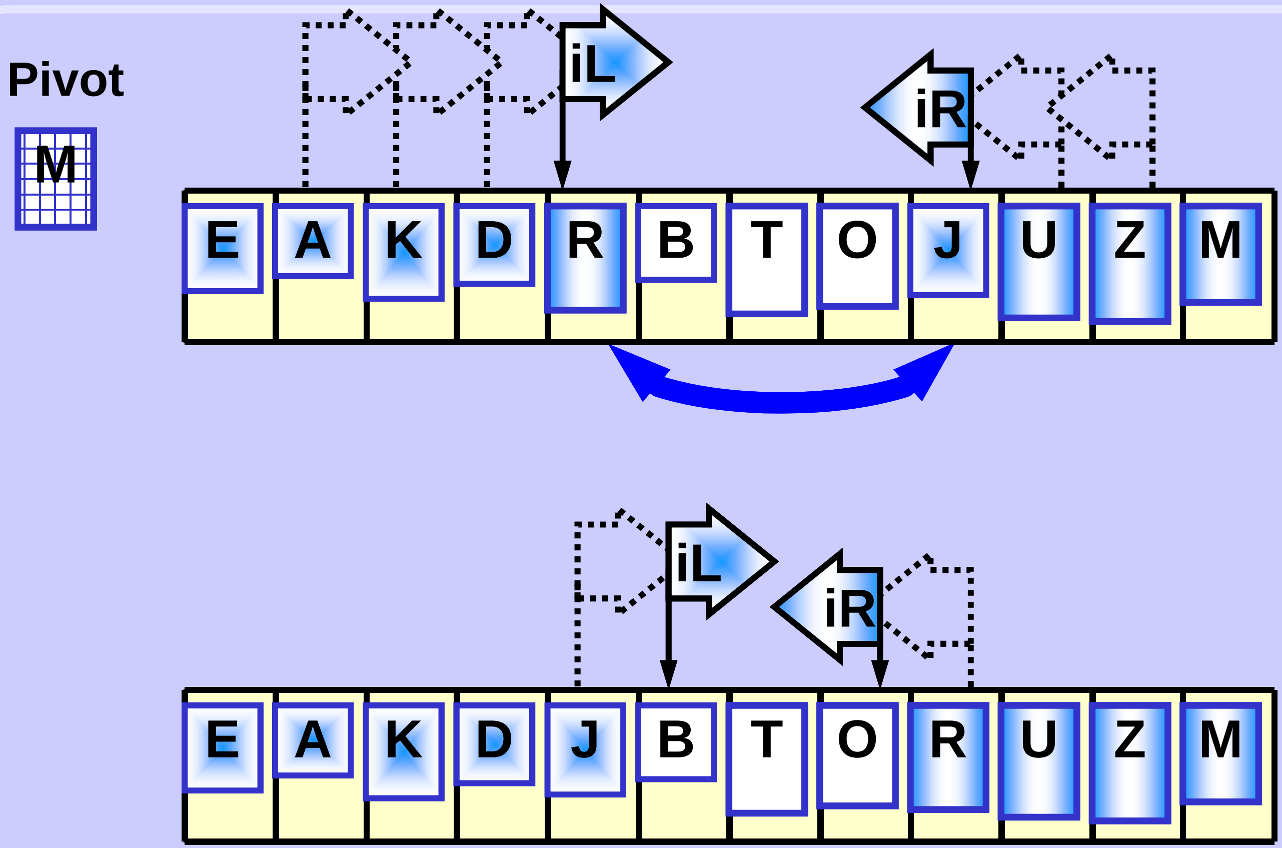
Step 1



Quicksort

Dividing

Step 2



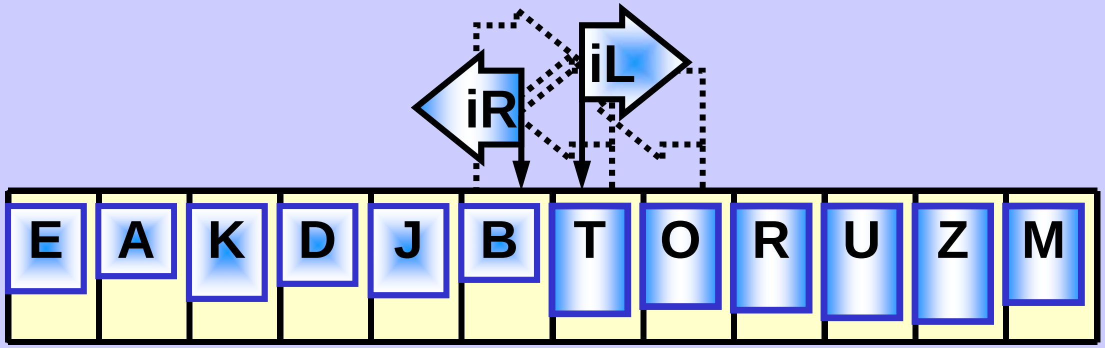
Quicksort

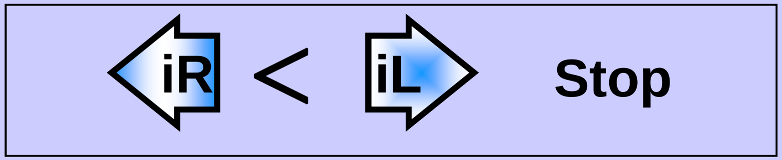
Dividing

Step 3

Pivot

M



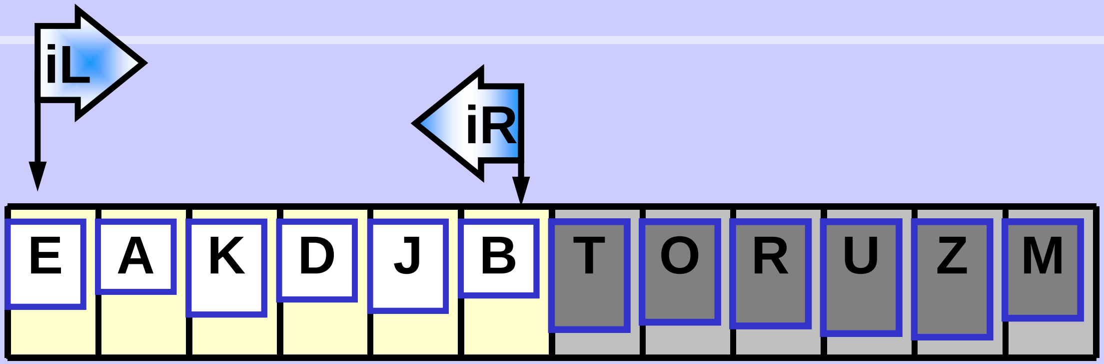


Divide!

Init

Pivot

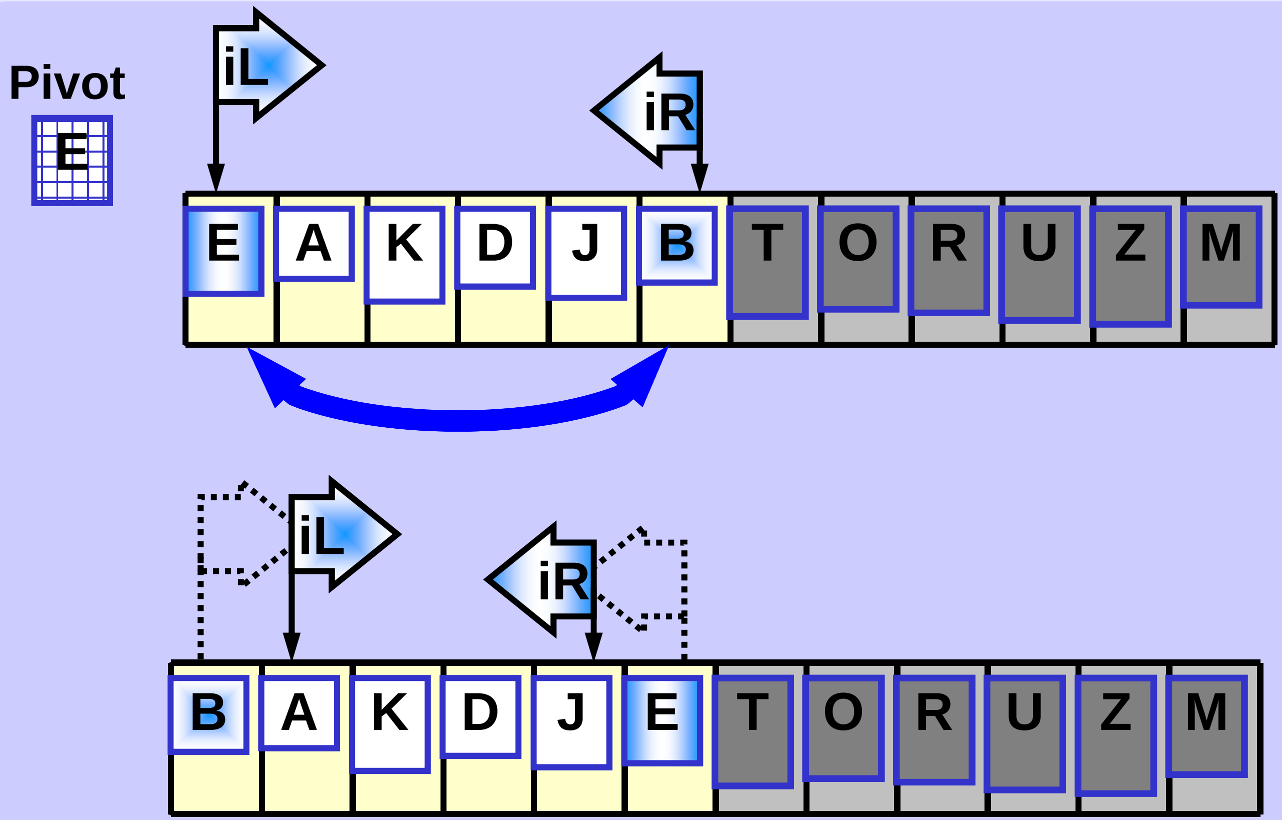
E



Quicksort

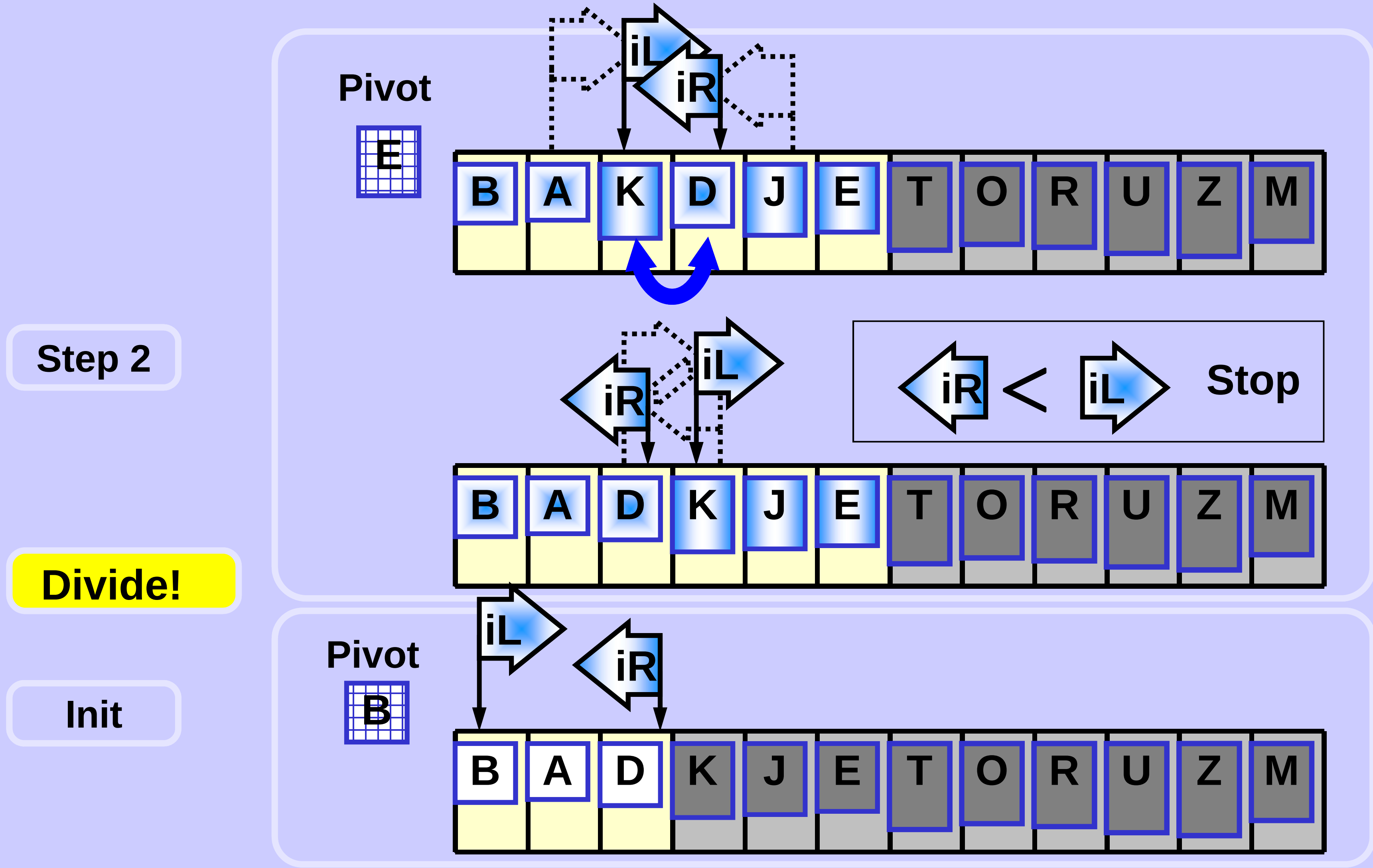
Dividing

Step 1



Quicksort

Dividing



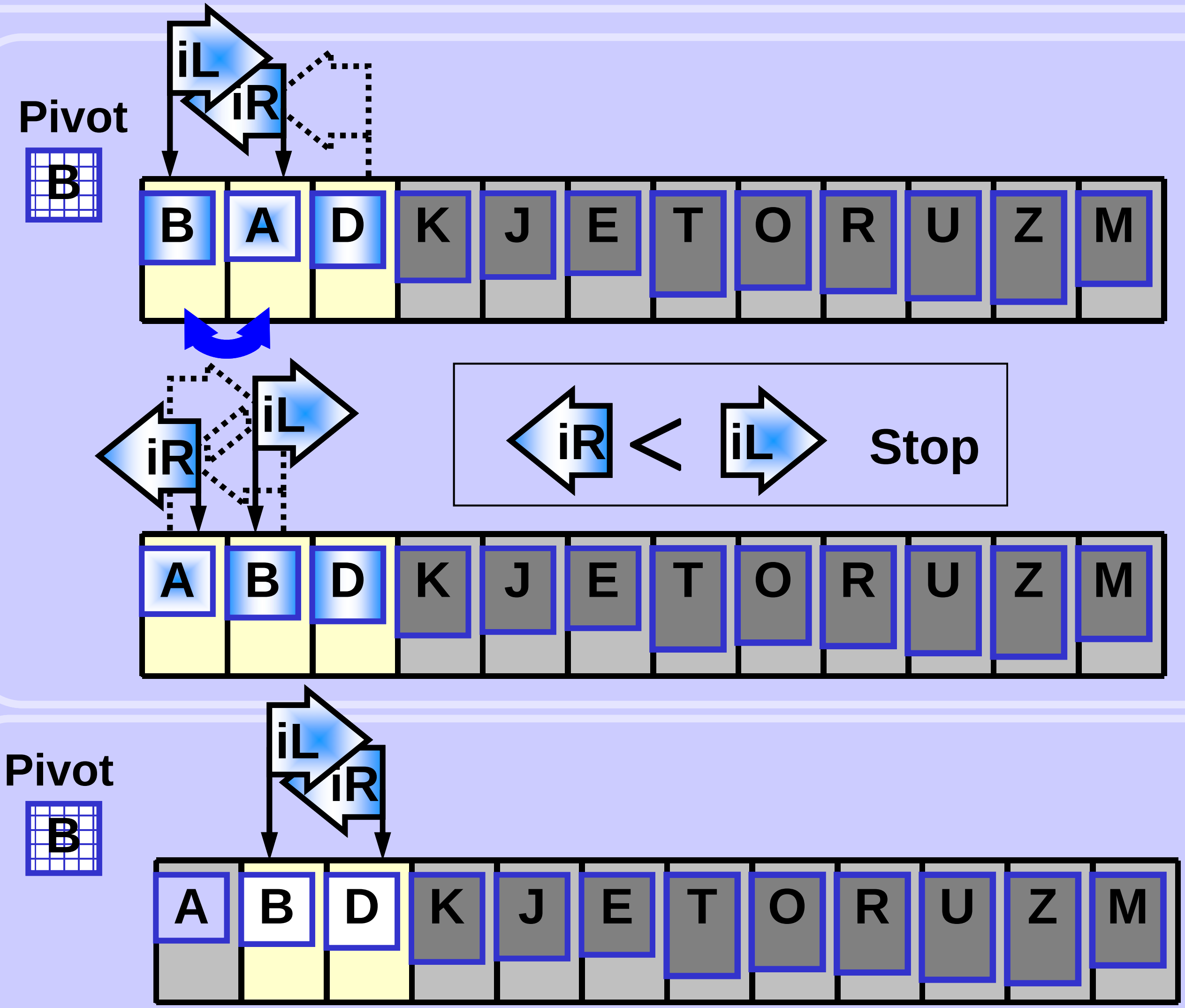
Quicksort

Dividing

Step 1

Divide!

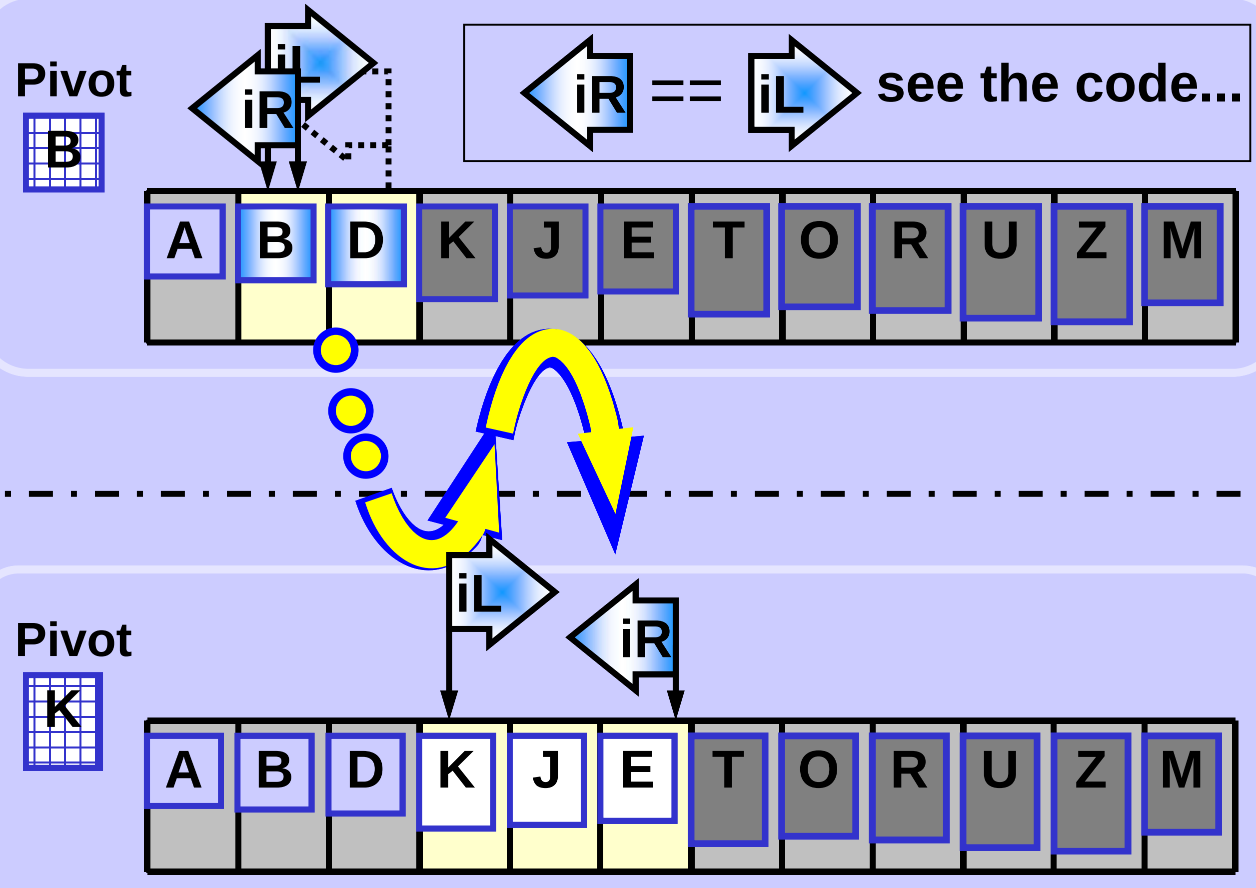
Init



Quicksort

Dividing

Step 1



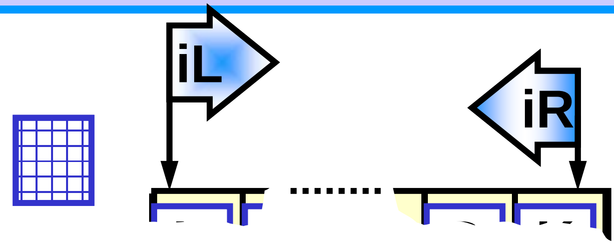
Next part

Init

etc...

etc…

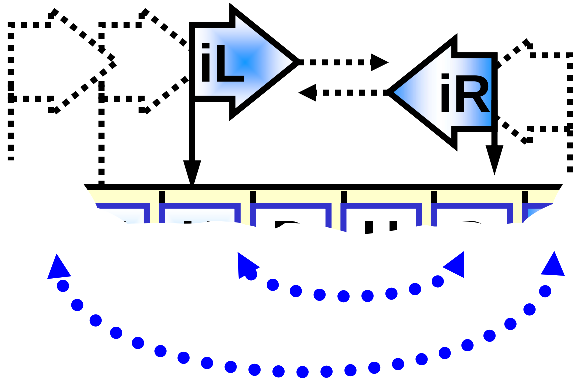
Quicksort




```
def qSort( a, low, high ):
 iL = low; iR = high;
 pivot = a[low]

 while True:
    if iL > iR: break
    while a[iL] < pivot: iL += 1
    while a[iR] > pivot: iR -= 1
     if iL < iR:
       swap(a, iL, iR)
       iL += 1; iR -= 1
     else:
       if iL == iR: iL += 1; iR -= 1

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



Divide!

Quicksort

Init: Left index is set to the first element of the current segment, right index is set to its last element, a pivot value is selected.

<u>Loop (dividing into ”small” and ”big”) :</u>

Left index moves to the right
and stops at element which value is greater or equal to the pivot.

Right index moves to the left
and stops at element which value is smaller or equal to the pivot.

If the left index is still to the left of the right index then
the corresponding elements are swapped
and both indices are moved by one position in their respective directions.

Else if the indices are equal then they are just moved by one
in their respective directions.

The loop stops when left index is to the right of the right one.

<u>The recursive calls follow</u> (processing “small" and „big" separately):

Processing segment <beginning, right index>
and the segment <left index, end>
if the segment length is greater than 1.

Quicksort

Asymptotic complexity

Total tests and moves

$\Theta(n \cdot \log_2(n))$	best case
$\Theta(n \cdot \log_2(n))$	expected case
$\Theta(n^2)$	worst case

Asymptotic complexity of Quicksort is $O(n^2)$ ...

... but! :

Expected complexity is $\Theta(n \cdot \log_2(n))$ (!!)

Quicksort

Comparing effectivity

N	N^2	$N \times \log_2(N)$	$\dfrac{N^2}{N \times \log_2(N)}$
1	1	0	
10	100	33.2	3.0
100	10 000	6 64.4	15.1
1 000	1 000 000	9 965.8	100.3
10 000	100 000 000	132 877.1	752.6
100 000	10 000 000 000	1 660 964.0	6 020.6
1 000 000	1 000 000 000 000	19 931 568.5	50 171.7
10 000 000	100 000 000 000 000	232 534 966.6	430 042.9

Tab. 1

Alternative Partition Method

Literature (e.g. [CLRS] and web as well, e.g. Wikipedia) sometimes mentiones the method implemented in the code below. The concept is straightforward, however, in practice it is slower, by factor 2 to 3.

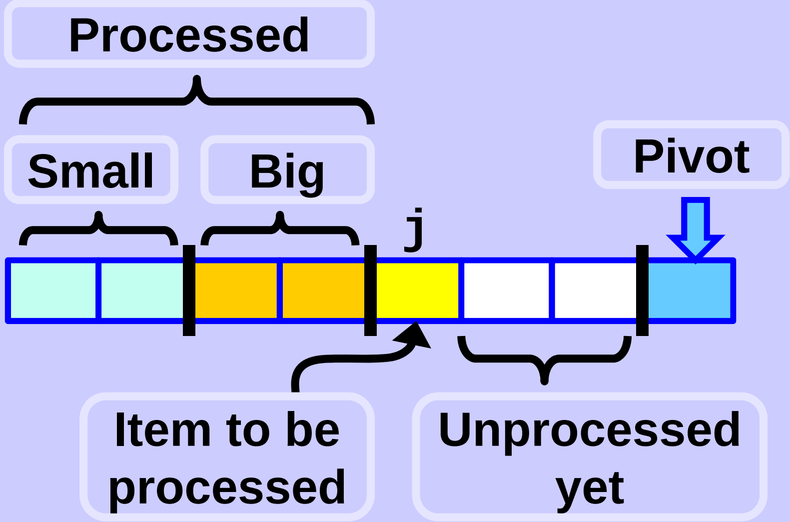
```
# returns index of the first element in the "big" part
# icluding the pivot
def partitionCLRS( a, iL, iR):
    pivot = a[iR]; iMid = iL-1
    for j in range(iL, iR):
        if a[j] <= pivot:
            iMid += 1
            swap(a, iMid, j)
    iMid += 1
    swap(a, iMid, R)
    return iMid
```

See the example on the next slide.

Each element smaller or equal to the pivot is swapped.
About half of all entries are expected to be swapped.
The classic variant performs much less swaps.

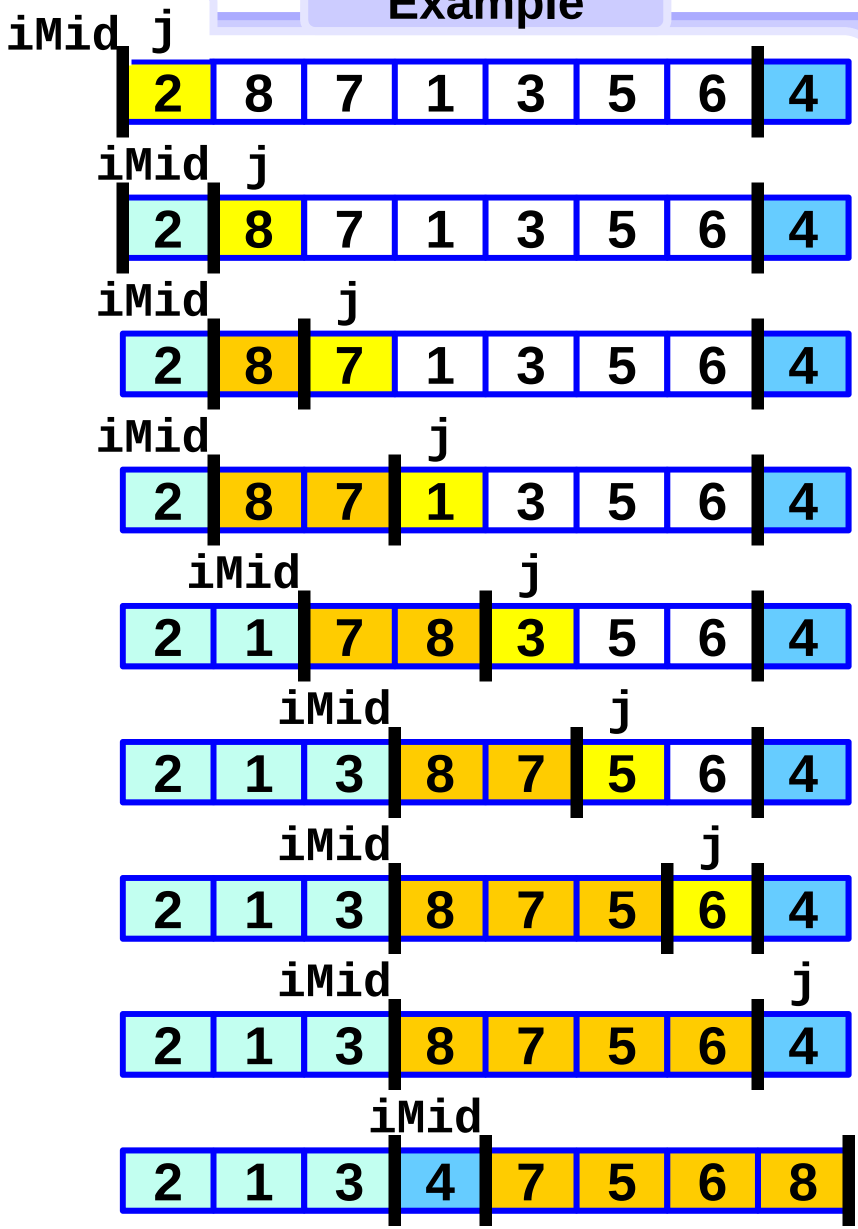
Alternative Partition Method

Process from left to right




```
pivot = a[iR]; iMid = iL-1
for j in range(iL, iR):
   if a[j] <= pivot:
      iMid += 1
      swap(a, iMid, j)
iMid += 1
swap(a, iMid, R)
return iMid
```

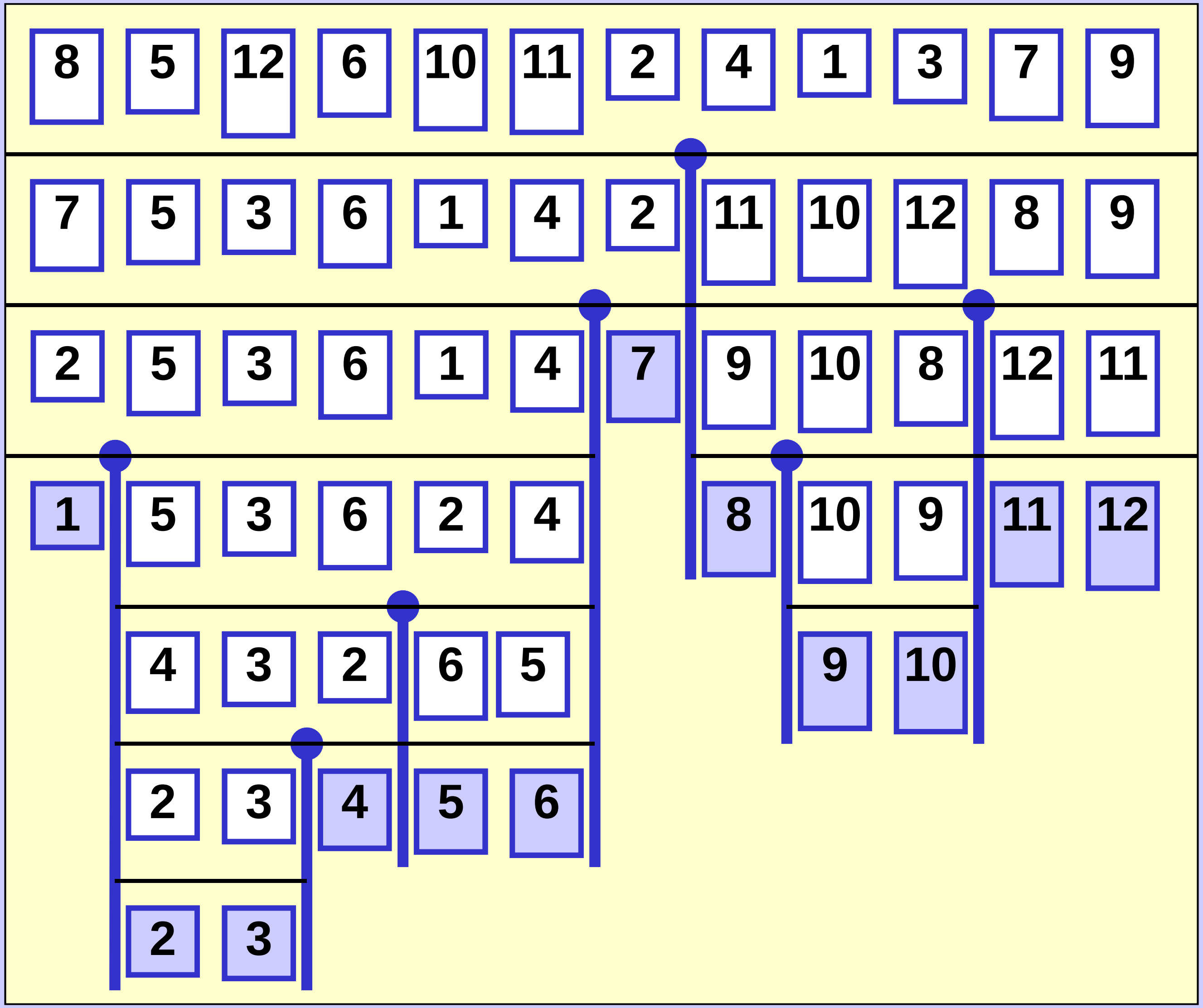
Example



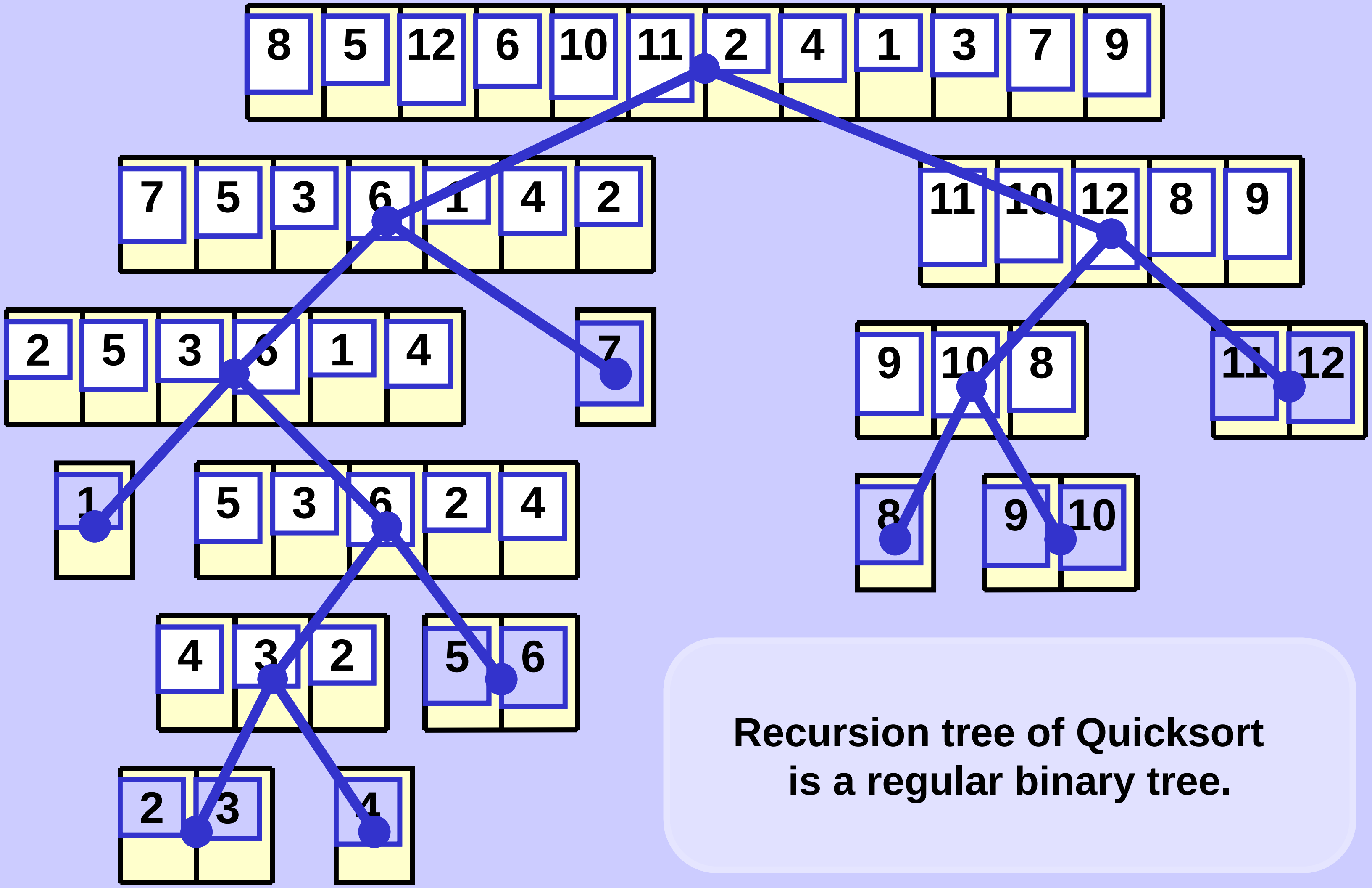
Quicksort

Example

pivot =
= first
in the
segment



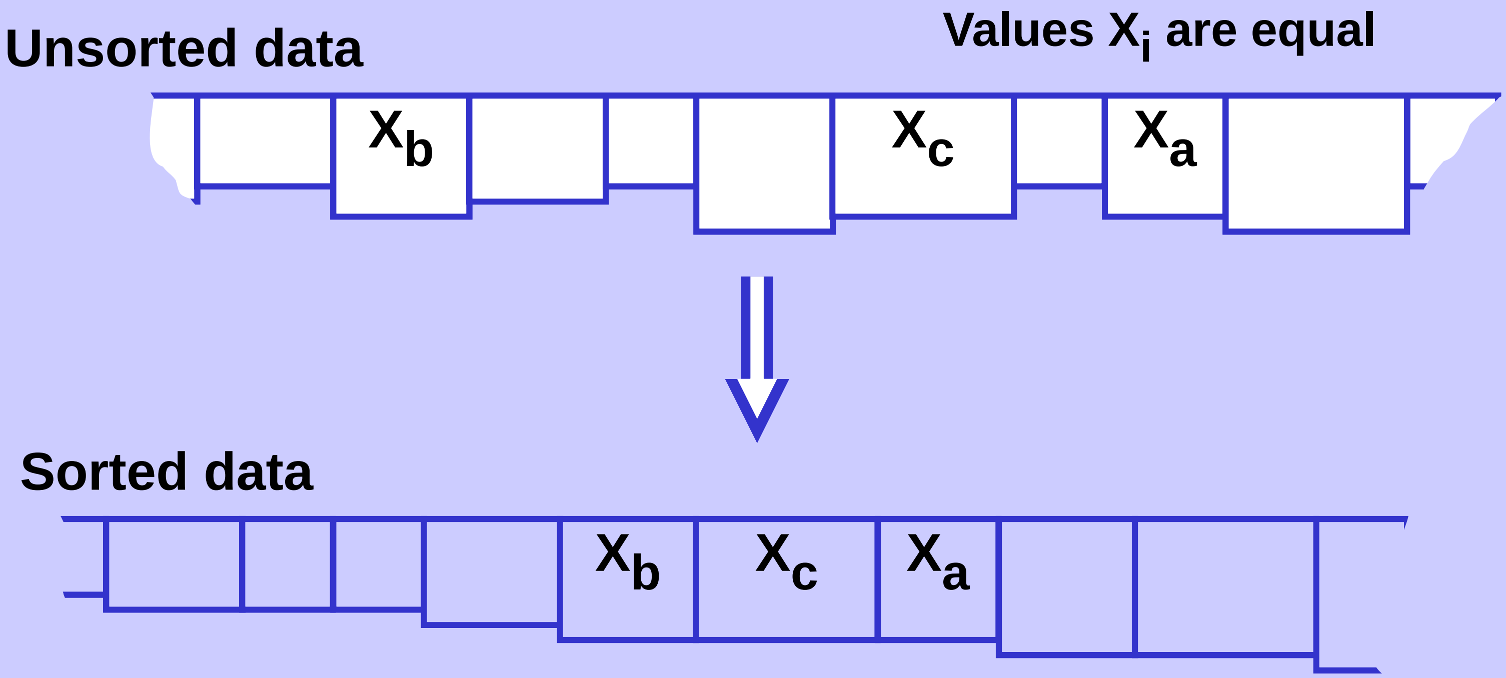
Quicksort



Recursion tree of Quicksort is a regular binary tree.

Stable sort

Stable sort does not change the order of elements with the same value.



Sort stability

$B_1\ D_1\ C_1\ A_1\ C_2\ B_2\ A_2\ D_2\ B_3\ A_3\ D_3\ C_3$

Insert
Bubble -- Stable implementation

$A_1\ A_2\ A_3\ B_1\ B_2\ B_3\ C_1\ C_2\ C_3\ D_1\ D_2\ D_3$

$B_1\ D_1\ C_1\ A_1\ C_2\ B_2\ A_2\ D_2\ B_3\ A_3\ D_3\ C_3$

Insert
Bubble -- Unstable implementation

Quicksort
Select sort Always unstable!!

$A_2\ A_1\ A_3\ B_2\ B_3\ B_1\ C_3\ C_1\ C_2\ D_3\ D_2\ D_1$

Stable sort

Record: | Name | Surname |

Input: List sorted only by names.

Andrew	Cook
Andrew	Amundsen
Andrew	Brown
Barbara	Cook
Barbara	Brown
Barbara	Amundsen
Charles	Amundsen
Charles	Cook
Charles	Brown

Stable sort

Sort records only by surnames

Output: List sorted by surnames and by names

Andrew	Amundsen
Barbara	Amundsen
Charles	Amundsen
Andrew	Brown
Barbara	Brown
Charles	Brown
Andrew	Cook
Barbara	Cook
Charles	Cook

The order of the records with the same name remains unchanged.

Unstable sort

Record: | Name | Surname |

Input: List sorted only by names.

Output: Original order of names is lost.

Andrew	Cook
Andrew	Amundsen
Andrew	Brown
Barbara	Cook
Barbara	Brown
Barbara	Amundsen
Charles	Amundsen
Charles	Cook
Charles	Brown

QuickSort

Sort records only by surnames

Sorted

Barbara	Amundsen
Andrew	Amundsen
Charles	Amundsen
Barbara	Brown
Charles	Brown
Andrew	Brown
Charles	Cook
Andrew	Cook
Barbara	Cook

The order of the records with the same name is changed.