CURS 11 SDA: TEHNICI DE DEZVOLTARE A ALGORITMILOR (IV) DIVIDE AND CONQUER

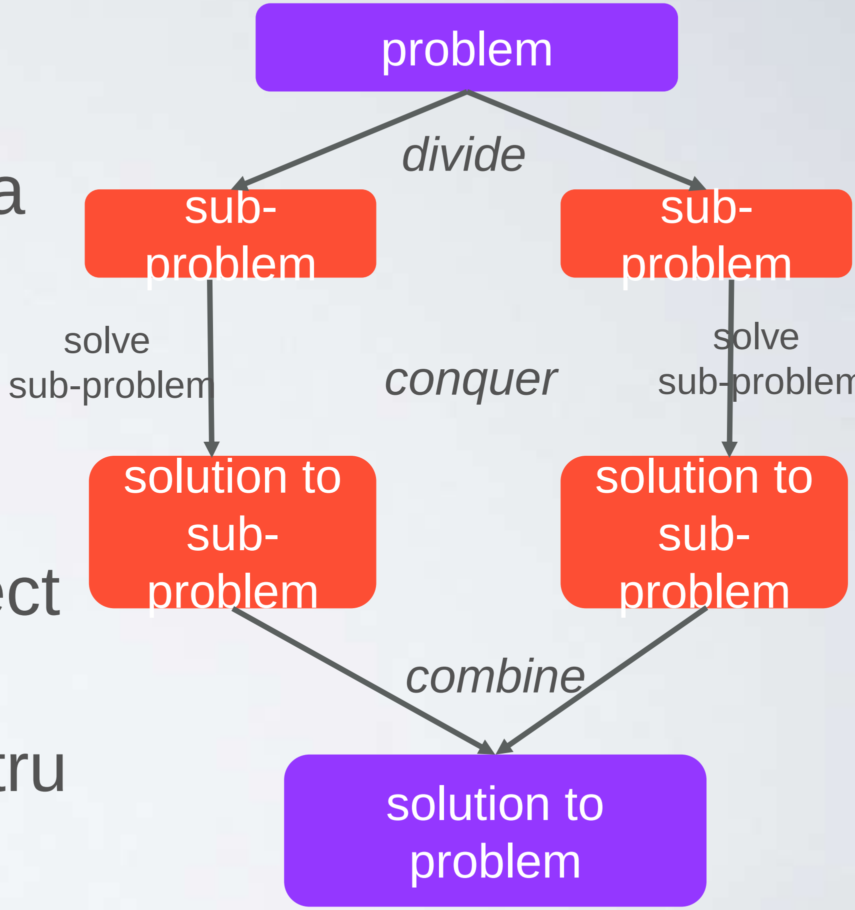
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

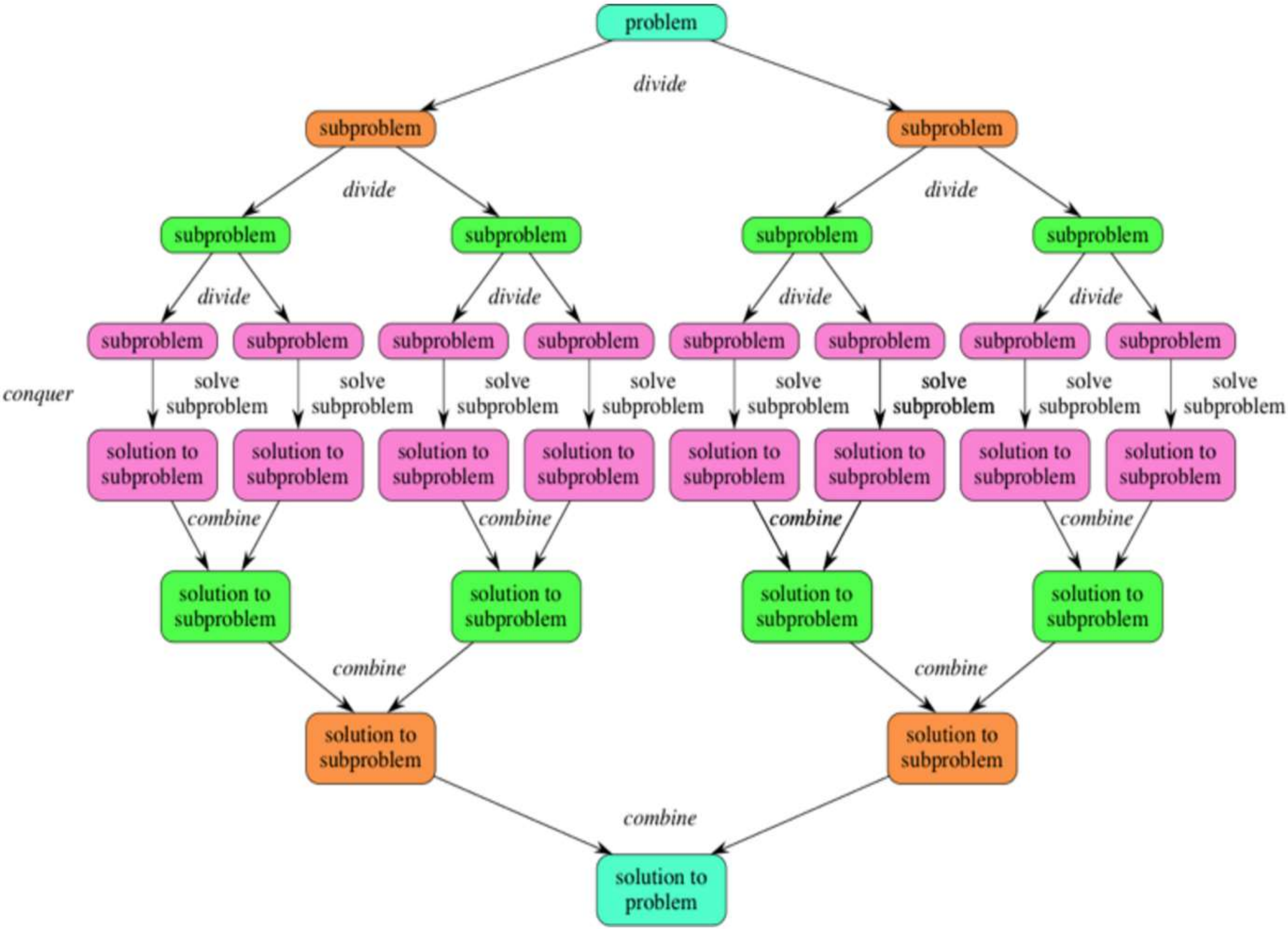
DIVIDE AND CONQUER

- Se imparte (*divide*) problema in una sau mai multe sub-probleme similare cu problema initiala
- Se rezolva fiecare sub-problema independent (*conquer*)
- Solutiile sub-problemelor se combina pentru a obtine solutia la problema initiala
  - se implementeaza de regula folosind recursivitatea
  - In general, sub-problemele generate la fiecare impartire sunt independente (en. non-overlapping)

DIVIDE AND CONQUER: PASI

- **Divide**: daca dimensiunea problemei de intrare este prea mare pentru a rezolva problema direct, o impartim in 2 sau mai multe sub-probleme disjuncte
- **Conquer**: se folosesc apeluri recursive pentru a rezolva sub-problemele, sau se rezolva direct
- **Combine**: se iau solutiile sub-problemelor si se combina pentru a obtine solutia la problema originala





https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/divide-and-

SUPLIMENT: TEOREMA LUI MASTER

- Pt. estimarea complexitatii in timp a alg. recursivi
- Recurente de forma:
  - $T(n) = aT(n/b) + n^c$, a>=1, b>1
    - daca $a < b^c$: $O(n^c)$
    - daca $a = b^c$: $O(n^c log_b n)$
    - daca $a > b^c$: $O(n^{log_b a})$
- Alte posibilitati: metoda substitutiei (inductie), metoda arborelui de recursivitate (pt a “ghici” complexitatea)
  - vezi Cormen, cap. 4

RIDICAREA LA PUTERE

Problema: Calculati a^n, unde $n \in N$

```
SlowPower(a,n)
1. x = a
2. for i = 2 to n
3.     x = x*a
4. return x
```

```
FastPower(a,n)
1. if n=1
2.     then return a
3. else
4.     x=FastPower(a,⌊n/2⌋)
5.     if n is even
6.         then return x*x
7.         else return x*x*a
```

- Comparati cei doi algoritmi:
  - cati pasi ia fiecare? (timp)
  - cata memorie?

RIDICAREA LA PUTERE

<u>Problema</u>: Calculati a^n, unde $n \in N$

```
SlowPower(a,n)
x = a
for i = 2 to n
    x = x*a
return x
```

Θ (n)

```
FastPower(a,n)
if n=1
    then return a
else
    x=FastPower(a,⌊n/2⌋)
    if n is even
        then return x*x
        else return x*x*a
```

$T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\lg n)$

- Comparati cei doi algoritmi:
  - cati pasi ia fiecare? (timp)
  - cata memorie?

CAUTARE

- Problema: se da vectorul A[1…n] ordonat crescator; se cere gasirea elementului q in A. Daca q nu este in A, sa se returneze pozitia unde q ar putea fi inserat
- Solutie naiva: cautare secventiala

```
LinearSearch(A[1…n], q)

for i = 1 to n do
    if A [i] ≥ q then
        return index i
return n + 1
```

- Timp: Θ(r), unde r este indexul returnat. In cazul defavorabil: O(n), respectiv O(1) in cazul favorabil

	1	2	3	4	5	6	7	8
A	2	3	5	5	7	9	9	12

Ce returneaza LinearSearch(A, 5)? Dar LinearSearch(A, 8)? Dar LinearSearch(A,13)?

LinearSearch(A, 5)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

	1	2	3	4	5	6	7	8	
A	**2**	3	5	5	7	9	9	12	q=5
	i								

	1	2	3	4	5	6	7	8	
A	2	**3**	5	5	7	9	9	12	q=5
		i							

	1	2	3	4	5	6	7	8	
A	2	3	**5**	5	7	9	9	12	q=5
			i						

	1	2	3	4	5	6	7	8	
A	2	3	**5**	5	7	9	9	12	q=5

… va returna valoarea 3

LinearSearch(A, 8)?

LinearSearch(A, 13)?

CAUTARE BINARA

- Varianta recursiva

```
BinarySearch(A, low, high, q)
    if low = high
       then return low  //index
    mid = ⌊(low + high)/2 ⌋
    if q ≤ A[mid]
        then return BinarySearch(A,low,mid,q)
    else return BinarySearch(A,mid+1,high,q)
```

- $T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\log n)$

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
A	2	3	5	5	7	9	9	12

Ce returneaza BinarySearch(A,1,8,5)?
Dar BinarySearch(A,1,8,8)?
Dar BinarySearch(A,1,8,13)?

BinarySearch(A,1,8,5)

```
BinarySearch(A, low, high, q)
    if low = high
       then return low  //index
    mid = ⌊(low + high)/2 ⌋
    if q ≤ A[mid]
        then return BinarySearch(A,low,mid,q)
    else return BinarySearch(A,mid+1,high,q)
```

	low							*high*
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
A	2	3	5	5	7	9	9	12

q=5

BinarySearch(A,1,8,5)

```
BinarySearch(A, low, high, q)
if low = high
  then return low  //index
mid = ⌊(low + high)/2 ⌋
if q ≤ A[mid]
   then return BinarySearch(A,low,mid,q)
else return BinarySearch(A,mid+1,high,q)
```

	low			mid				high	
	1	2	3	4	5	6	7	8	
A	2	3	5	5	7	9	9	12	q=5

BinarySearch(A,1,8,5)

BinarySearch(A,1,4,5)

```
BinarySearch(A, low, high, q)
    if low = high
      then return low  //index
    mid = ⌊(low + high)/2 ⌋
    if q ≤ A[mid]
       then return BinarySearch(A,low,mid,q)
    else return BinarySearch(A,mid+1,high,q)
```

	low			high				
	1	2	3	4	5	6	7	8
A	2	3	5	5	7	9	9	12

q=5

BinarySearch(A,1,8,5)

BinarySearch(A,1,4,5)

```
BinarySearch(A, low, high, q)
1.   if low = high
2.     then return low  //index
3.   mid = ⌊(low + high)/2 ⌋
4.   if q ≤ A[mid]
5.      then return BinarySearch(A,low,mid,q)
6.   else return BinarySearch(A,mid+1,high,q)
```

	low	mid		high				
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
A	2	3	5	5	7	9	9	12

q=5

BinarySearch(A,1,8,5)

BinarySearch(A,1,4,5)

BinarySearch(A,3,4,5)

```
BinarySearch(A, low, high, q)
1.  if low = high
2.     then return low  //index
3.  mid = ⌊(low + high)/2 ⌋
4.  if q ≤ A[mid]
5.      then return BinarySearch(A,low,mid,q)
6.  else return BinarySearch(A,mid+1,high,q)
```

	low	*high*					
1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
2	3	5	5	7	9	9	12

A (q=5)

BinarySearch(A,1,8,5)

BinarySearch(A,1,4,5)

BinarySearch(A,3,4,5)

```
BinarySearch(A, low, high, q)
    if low = high
       then return low   //index
    mid = ⌊(low + high)/2 ⌋
    if q ≤ A[mid]
        then return BinarySearch(A,low,mid,q)
    else return BinarySearch(A,mid+1,high,q)
```

			low	mid	high				
		1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
A		2	3	5	5	7	9	9	12

q=5

BinarySearch(A,1,8,5)

BinarySearch(A,1,4,5)

BinarySearch(A,3,4,5)

BinarySearch(A,3,3,5)

```
BinarySearch(A, low, high, q)
    if low = high
      then return low  //index
    mid = ⌊(low + high)/2 ⌋
    if q ≤ A[mid]
       then return BinarySearch(A,low,mid,q)
    else return BinarySearch(A,mid+1,high,q)
```

low high

	1	2	3	4	5	6	7	8	
A	2	3	5	5	7	9	9	12	q=5

3

BinarySearch(A,1,8,5) 3

BinarySearch(A,1,4,5)

BinarySearch(A,3,4,5) 3

BinarySearch(A,3,3,5) 3

```
BinarySearch(A, low, high, q)
  if low = high
     then return low  //index
  mid = ⌊(low + high)/2 ⌋
  if q ≤ A[mid]
      then return BinarySearch(A,low,mid,q)
  else return BinarySearch(A,mid+1,high,q)
```

low high

	1	2	**3**	4	5	6	7	8	
A	2	3	5	5	7	9	9	12	q=5

BinarySearch(A,1,8,8)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

```
BinarySearch(A, low, high, q)
    if low = high
      then return low  //index
    mid = ⌊(low + high)/2 ⌋
    if q ≤ A[mid]
       then return BinarySearch(A,low,mid,q)
    else return BinarySearch(A,mid+1,high,q)
```

	low							high	
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	
A	2	3	5	5	7	9	9	12	q=8

BinarySearch(A,1, 8, 8)

```
BinarySearch(A, low, high, q)
    if low = high
      then return low   //index
    mid = ⌊(low + high)/2 ⌋
    if q ≤ A[mid]
       then return BinarySearch(A,low,mid,q)
    else return BinarySearch(A,mid+1,high,q)
```

	low			mid				high	
	1	2	3	4	5	6	7	8	
A	2	3	5	5	7	9	9	12	q=8

BinarySearch(A,1, 8, 8)

BinarySearch(A,5,8,8)

```
BinarySearch(A, low, high, q)
    if low = high
      then return low  //index
    mid = ⌊(low + high)/2 ⌋
    if q ≤ A[mid]
       then return BinarySearch(A,low,mid,q)
    else return BinarySearch(A,mid+1,high,q)
```

					low			*high*
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
A	2	3	5	5	7	9	9	12

q=8

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

BinarySearch(A,1, 8, 8)

BinarySearch(A,5,8,8)

```
BinarySearch(A, low, high, q)
if low = high
   then return low  //index
mid = ⌊(low + high)/2 ⌋
if q ≤ A[mid]
    then return BinarySearch(A,low,mid,q)
else return BinarySearch(A,mid+1,high,q)
```

					low	mid		high
	1	2	3	4	5	6	7	8
A	2	3	5	5	7	9	9	12

q=8

BinarySearch(A,1, 8, 8)

BinarySearch(A,5,8,8)

BinarySearch(A,5,6,8)

```
BinarySearch(A, low, high, q)
   if low = high
     then return low  //index
   mid = ⌊(low + high)/2 ⌋
   if q ≤ A[mid]
      then return BinarySearch(A,low,mid,q)
   else return BinarySearch(A,mid+1,high,q)
```

				low	high		
1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
2	3	5	5	7	9	9	12

A

q=8

BinarySearch(A,1, 8, 8)

BinarySearch(A,5,8,8)

BinarySearch(A,5,6,8)

```
BinarySearch(A, low, high, q)
    if low = high
      then return low  //index
    mid = ⌊(low + high)/2 ⌋
    if q ≤ A[mid]
       then return BinarySearch(A,low,mid,q)
    else return BinarySearch(A,mid+1,high,q)
```

					lowmid	high		
	1	2	3	4	5	6	7	8
A	2	3	5	5	7	9	9	12

q=8

BinarySearch(A,1, 8, 8)
  BinarySearch(A,5,8,8)
    BinarySearch(A,5,6,8)
      BinarySearch(A,6,6,8)

```
BinarySearch(A, low, high, q)
    if low = high
      then return low  //index
    mid = ⌊(low + high)/2 ⌋
    if q ≤ A[mid]
       then return BinarySearch(A,low,mid,q)
    else return BinarySearch(A,mid+1,high,q)
```

						low high			
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	
A	2	3	5	5	7	9	9	12	q=8

6

BinarySearch(A,1, 8, 8) 6

BinarySearch(A,5,8,8) 6

BinarySearch(A,5,6,8) 6

BinarySearch(A,6,6,8) 6

```
BinarySearch(A, low, high, q)
    if low = high
       then return low  //index
    mid = ⌊(low + high)/2 ⌋
    if q ≤ A[mid]
        then return BinarySearch(A,low,mid,q)
    else return BinarySearch(A,mid+1,high,q)
```

					low *high*		
1	*2*	*3*	*4*	*5*	***6***	*7*	*8*
2	3	5	5	7	9	9	12

A

q=8

Tema: BinarySearch(A,1,8,13)?

CAUTARE BINARA

- Varianta iterativa

```
BinarySearch(A, q)
if q > A [n]
     then return n + 1
i = 1
j = n
while i < j do
    k = (i + j)/2
    if q ≤ A [k]
        then j = k
        else i = k + 1
return i //the index
```

- $T(n) = \Theta(\log n)$

Tema: Ce returneaza BinarySearch(A,5)?
Dar BinarySearch(A,8)?
Dar BinarySearch(A,13)?

SORTARE CU DIVIDE AND CONQUER

- Problema: se da vectorul A[1…n]; se cere sa se gaseasca o permutare a elementelor lui A, a.i. $\mathrm{A}[1] \leq A[2] \leq \cdots \leq A[n]$
- 2 algoritmi "*buni*" de sortare utilizeaza tehnica D&C
  - **MergeSort (Interclasare)**
    - Sorteaza jumatatea din stanga a sirului (recursiv)
    - Sorteaza jumatatea din dreapta a sirului (recursiv)
    - Interclaseaza (en. *merge*) cele 2 parti sortate, pentru a obtine sirul sortat
  - **Quicksort (Rapida)**
    - Alege un element din sir ca *pivot*
    - Imparte elementele sirului in 2 partitii: elemente ≤ pivot, si elemente > pivot, pivotul se pozitioneaza intre cele 2 partitii
    - Sorteaza partitia elementelor ≤ pivot
    - Sorteaza partitia elementelor > pivot

  (sirul rezultat este gata sortat)

SORTAREA PRIN INTERCLASARE (MERGESORT)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
A	9	3	12	5	7	2	9	5

- Pentru a sorta vectorul A, intre indecsii *low* si *high*
  - Daca am ajuns la sir de 1 element, e gata sortat, stop
  - Altfel:
    - Sorteaza (recursiv) sirul de la *low* la *(low + high)/2*
    - Sorteaza (recursiv) sirul de la *(low + high)/2* la *high*
    - Interclaseaza cele 2 jumatati de sir rezultate
  - Operatia de interclasare primeste 2 parti sortate si genereaza sirul sortat cu toate elementele
    - O(n)
    - …dar necesita memorie auxiliara

EXEMPLU: INTERCLASARE

Incepem cu …

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
9	3	12	5	7	2	9	5

EXEMPLU: INTERCLASARE

Incepem cu …

1	2	3	4	5	6	7	8
9	3	12	5	7	2	9	5

Se vor lansa apelurile recursive:

```
MergeSort(A, 1, 4)
MergeSort(A, 5, 8)
```

EXEMPLU: INTERCLASARE

Incepem cu …

1	2	3	4	5	6	7	8	
9	3	12	5	7	2	9	5	A

Se vor lansa apelurile recursive:

```
MergeSort(A, 1, 4)
MergeSort(A, 5, 8)
```

dupa ce vor returna, sirul arata asa:

1	2	3	4	5	6	7	8	
3	5	9	12	2	5	7	9	A

EXEMPLU: INTERCLASARE

Initial:

1	2	3	4	5	6	7	8	
9	3	12	5	7	2	9	5	A

Dupa apeluri:

1	2	3	4	5	6	7	8	
3	5	9	12	2	5	7	9	A

i *j*

Merge (Interclaseaza):

1	2	3	4	5	6	7	8	
								Aux

k

(dupa interclasare, rezultatul se va copia inapoi in sirul initial):

EXEMPLU: INTERCLASARE

Initial:

1	2	3	4	5	6	7	8	
9	3	12	5	7	2	9	5	A

Dupa apeluri:

1	2	3	4	5	6	7	8	
3	5	9	12	2	5	7	9	A

i

j

Merge (Interclaseaza):

1	2	3	4	5	6	7	8	
2								Aux

k

(dupa interclasare, rezultatul se va copia inapoi in sirul initial):

EXEMPLU: INTERCLASARE

Initial:

1	2	3	4	5	6	7	8	
9	3	12	5	7	2	9	5	A

Dupa apeluri:

1	2	3	4	5	6	7	8	
3	5	9	12	2	5	7	9	A

i *j*

Merge (Interclaseaza):

1	2	3	4	5	6	7	8	
2	3							Aux

k

(dupa interclasare, rezultatul se va copia inapoi in sirul initial):

EXEMPLU: INTERCLASARE

Initial:

	1	2	3	4	5	6	7	8	
	9	3	12	5	7	2	9	5	A

Dupa apeluri:

	1	2	3	4	5	6	7	8	
	3	5	9	12	2	5	7	9	A

i *j*

Merge (Interclaseaza):

	1	2	3	4	5	6	7	8	
	2	3	5						Aux

k

(dupa interclasare, rezultatul se va copia inapoi in sirul initial):

EXEMPLU: INTERCLASARE

Initial:

	1	2	3	4	5	6	7	8
A	9	3	12	5	7	2	9	5

Dupa apeluri:

	1	2	3	4	5	6	7	8
A	3	5	9	12	2	5	7	9

i *j*

Merge (Interclaseaza):

	1	2	3	4	5	6	7	8
Aux	2	3	5	5				

k

(dupa interclasare, rezultatul se va copia inapoi in sirul initial):

EXEMPLU: INTERCLASARE

Initial:

	1	2	3	4	5	6	7	8	
	9	3	12	5	7	2	9	5	A

Dupa apeluri:

	1	2	3	4	5	6	7	8	
	3	5	9	12	2	5	7	9	A

i *j*

Merge (Interclaseaza):

	1	2	3	4	5	6	7	8	
	2	3	5	5	7				Aux

k

(dupa interclasare, rezultatul se va copia inapoi in sirul initial):

EXEMPLU: INTERCLASARE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Initial: (A)

1	2	3	4	5	6	7	8
9	3	12	5	7	2	9	5

Dupa apeluri: (A)

1	2	3	4	5	6	7	8
3	5	9	12	2	5	7	9

i *j*

Merge (Interclaseaza): (Aux)

1	2	3	4	5	6	7	8
2	3	5	5	7	9		

k

(dupa interclasare, rezultatul se va copia inapoi in sirul initial):

EXEMPLU: INTERCLASARE

Initial:

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	
	9	3	12	5	7	2	9	5	A

Dupa apeluri:

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	
	3	5	9	12	2	5	7	9	A
				i				*j*	

Merge (Interclaseaza):

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	
	2	3	5	5	7	9	9		Aux
								k	

(dupa interclasare, rezultatul se va copia inapoi in sirul initial):

EXEMPLU: INTERCLASARE

Initial:

1	2	3	4	5	6	7	8
9	3	12	5	7	2	9	5

A

Dupa apeluri:

1	2	3	4	5	6	7	8
3	5	9	12	2	5	7	9

A

i *j*

Merge (Interclaseaza):

1	2	3	4	5	6	7	8
2	3	5	5	7	9	9	12

Aux

k

(dupa interclasare, rezultatul se va copia inapoi in sirul initial):

EXEMPLU: INTERCLASARE

Initial:

1	2	3	4	5	6	7	8	
9	3	12	5	7	2	9	5	A

Dupa apeluri:

1	2	3	4	5	6	7	8	
3	5	9	12	2	5	7	9	A

i *j*

Merge (Interclaseaza):

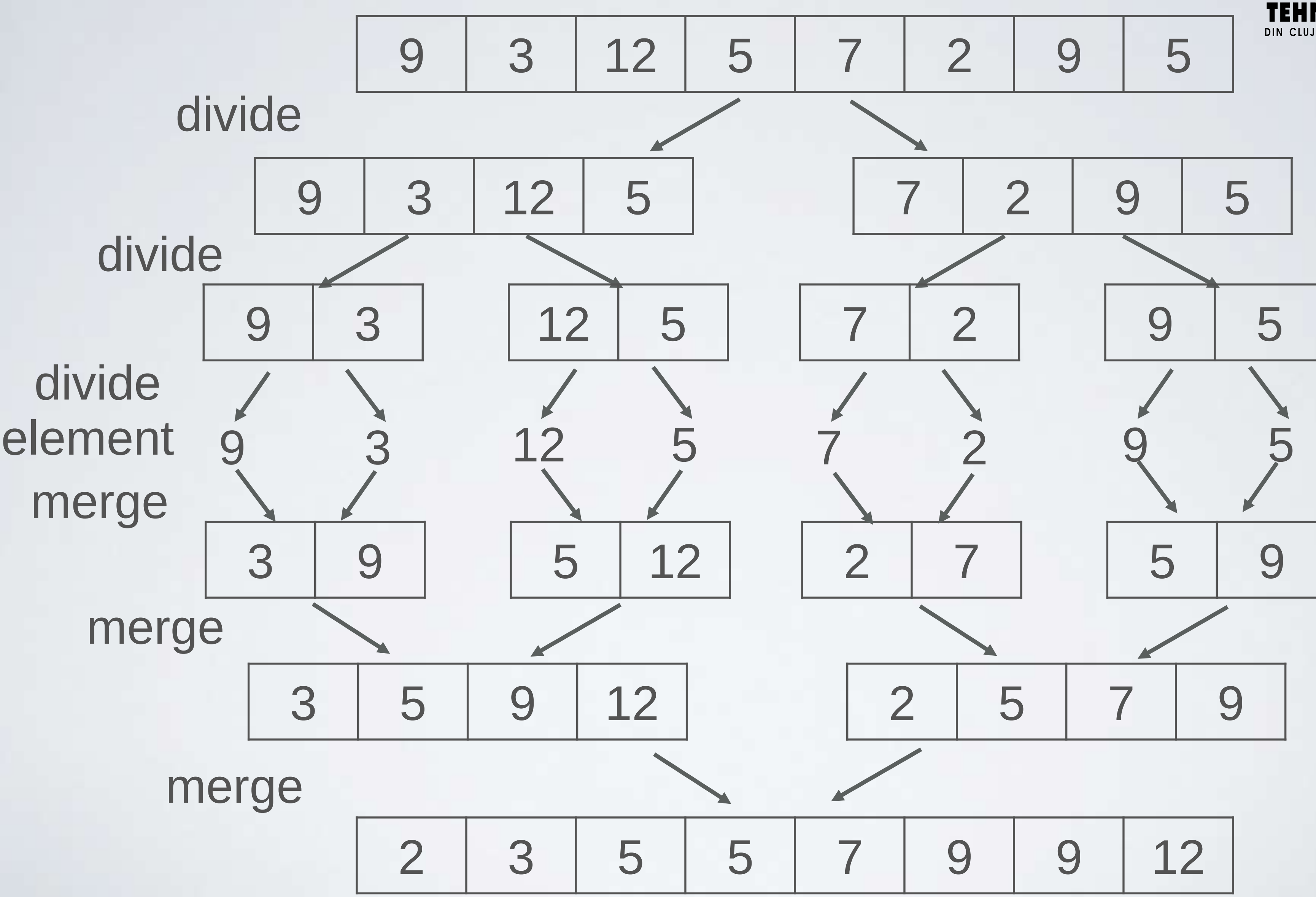
1	2	3	4	5	6	7	8	
2	3	5	5	7	9	9	12	Aux

(dupa interclasare, rezultatul se va copia inapoi in sirul initial):

k

1	2	3	4	5	6	7	8	
2	3	5	5	7	9	9	12	A

EXEMPLU: APELURILE RECURSIVE



MERGESORT: PSEUDOCOD

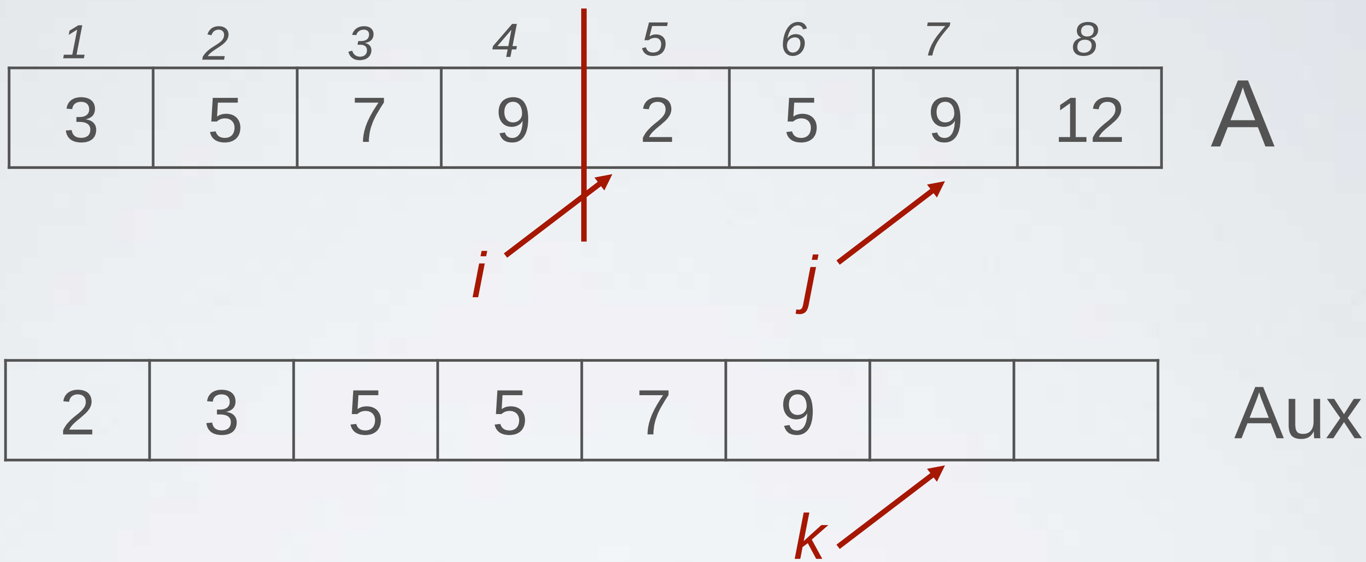
```
MERGE-SORT(A, low, high)
if low < high
    then mid = ⌊(low + high)/2 ⌋
    MERGE-SORT(A, low, mid)
    MERGE-SORT(A, mid+1, high)
    MERGE(A, low, mid, high)

MERGE(A, low, mid, high)
Let Aux[low...high] be a new array //auxiliary memory
i = low
k = low
j = mid+1
while i ≤ mid and j ≤high // now merge ...
    if A[i] ≤ A[j]
        then Aux[k++] = A[i]
        i = i + 1
    else Aux[k++] = A[j]
        j = j+1
while i ≤ mid //copy remaining elems from 1st part into Aux
    Aux[k++] = A[i++]
while j ≤high //copy remaining elems from 2nd part into Aux
    Aux[k++] = A[j++]
for kk=low to high //copy Aux back into A
    A[kk] = Aux[kk]
```

MERGESORT: DETALII LEGATE DE IMPLEMENTARE EFICIENTA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

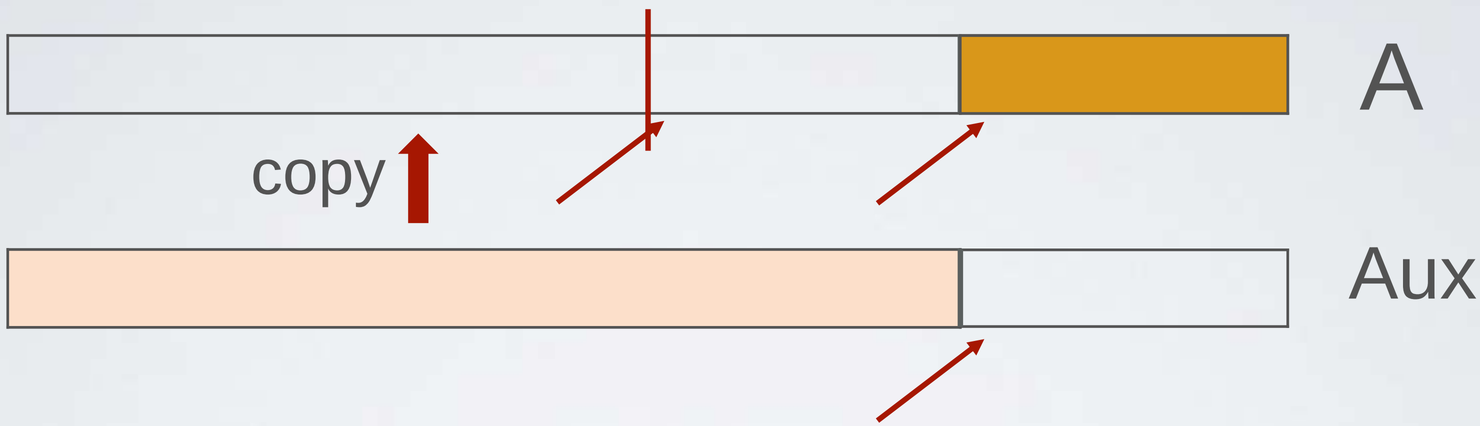
- Daca, in ultimii pasi de interclasare, vectorul nostru arata asa:



- nu este eficient sa copiem ultimele 2 elemente din A in Aux, doar ca apoi sa le copiem inapoi in A

MERGESORT: DETALII LEGATE DE IMPLEMENTARE EFICIENTA

Daca partea stanga termina, stop si copiaza inapoi:



Daca partea dreapta termina, copiaza restul din stanga in zona corespunzatoare, si copiaza inapoi:

MERGESORT: DETALII LEGATE DE IMPLEMENTARE EFICIENTA

- Utilizarea sirului <u>Aux</u> – strategii:
  - Simpla/ineficienta:
    - Sir nou, de dimensiune *high-low,* pentru fiecare apel de interclasare (i.e. aloci sir nou la fiecare apel de interclasare)
  - Mai bine:
    - Sir nou, de dimensiune, *n*, pentru fiecare stadiu de interclasare (i.e. utilizezi un singur sir, de dimensiune *n*, pentru toate interclasarile de la un nivel)
  - Si mai bine:
    - Reutilizezi *acelasi* sir de dimensiune *n*, la fiecare stadiu de interclasare
  - Cel mai bine (dar putina bataie de cap la implementare)*:*
    - Nu copia inapoi la stadiile pare de interclasare (al 2-lea, al 4-lea, etc), ci utilizeaza Aux ca si A, si A ca si Aux (i.e. interschimba “intelesul” celor 2 siruri)

MERGESORT SI LISTE INLANTUITE

- Ce se intampla daca secventa de sortat este lista inlantuita, nu sir?
- Am putea sa …
  - Convertim la sir: O(n)
  - Sortam
  - Convertim inapoi la lista: O(n)
- … SAU – MergeSort poate functiona foarte bine si pe structuri inlantuite, nu doar pe cele secventiale – intrucat nu se bazeaza pe acces indexat! (ca si Quicksort – dupa cum vom vedea, sau HeapSort – dupa cum veti vedea in anul al II-lea)
- Tocmai de aceea, MergeSort este utilizat pentru sortare externa
  - Interclasarea liniara minimizeaza numarul de accese la disc
  - Se poate paraleliza relativ usor (pt date masive)

MERGESORT: ANALIZA

- T(n) = $\begin{cases} c_1, n = 1 \\ 2T\left(\frac{n}{2}\right) + c_2 n \end{cases}$
- T(n) = $2T\left(\frac{n}{2}\right) + n =$

$$= 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n = 4T\left(\frac{n}{4}\right) + 2n =$$

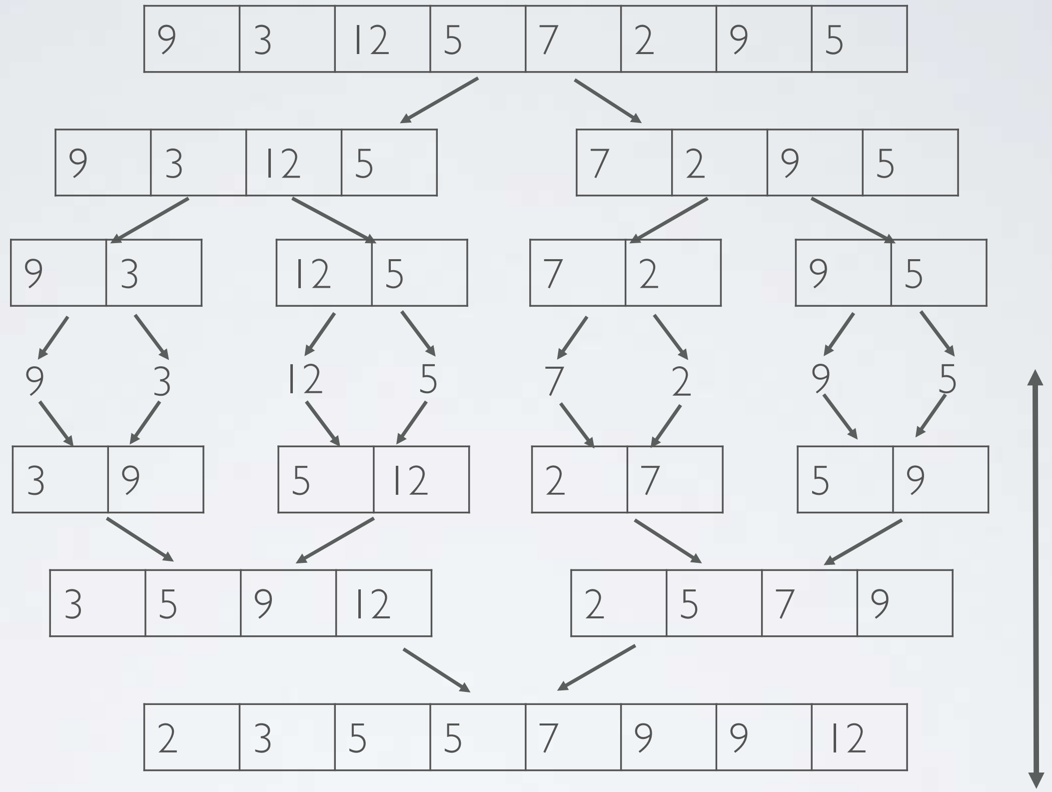
$$= 4\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n = 8T\left(\frac{n}{8}\right) + 3n =$$

$$= \ldots. =$$

$$= 2^k T\left(\frac{n}{2^k}\right) + kn$$

- $\frac{n}{2^k} = 1, deci\ k = \mathrm{logn}$
- Deci: *T(n) = n+nlogn = nlogn*

MERGESORT: ANALIZA INTUITIVA



Efort total nivel: *n*

Efort total nivel: *n*

Efort total nivel: *n*

H = log n

- Arborele de recursie are inaltime: *log n*
- La fiecare nivel al arborelui, efortul total este: *n* (pentru toate interclasarile, pe revenire)

SORTAREA RAPIDA(QUICKSORT)

	1	2	3	4	5	6	7	8
A	9	3	12	5	7	2	9	5

- Pentru a sorta vectorul A, intre indecsii *low* si *high*
  - Daca am ajuns la sir de 0 elemente, e gata sortat, stop
  - Altfel:
    - Alege un element ca si pivot (vom exemplifica varianta cu *ultimul* element ca pivot, dar mai sunt si alte variante de selectie)
    - Partitioneaza elementele sirului in 2 partitii: elemente ≤ pivot, si elemente > pivot, pivotul se pozitioneaza intre cele 2 partitii
    - Sorteaza partitia elementelor ≤ pivot
    - Sorteaza partitia elementelor > pivot

EXEMPLU: PARTITIONARE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Incepem cu … A

1	2	3	4	5	6	7	8
9	3	12	5	7	2	9	5

Se va apela: `partition(A,1,8)`

pivot = A[high] A

1	2	3	4	5	6	7	8
9	3	12	5	7	2	9	**5**

In partitionare:

- Sirul elementelor ≤ pivot va incepe la *low-1*initial
- Se parcurge sirul de la *low* la *high -1,* si:
- daca element_curent ≤ pivot, crestem sirul elementelor mai mici, si interschimbam element_curent ***cu noua ultima pozitie a elementelor mai mici decat pivotul***

EXEMPLU: PARTITIONARE


```
partition(A,1,8)
```

	1	2	3	4	5	6	7	8
A	9	3	12	5	7	2	9	**5**

sm *crt*

EXEMPLU: PARTITIONARE

`partition(A,1,8)`

	1	2	3	4	5	6	7	8
A	9	3	12	5	7	2	9	**5**

sm *crt*

	1	2	3	4	5	6	7	8
A	9	3	12	5	7	2	9	**5**

sm *crt*

	1	2	3	4	5	6	7	8
A	**3**	**9**	12	5	7	2	9	**5**

sm *crt*

EXEMPLU: PARTITIONARE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

`partition(A,1,8)`

	1	2	3	4	5	6	7	8
A	3	9	12	5	7	2	9	**5**

sm *crt*

EXEMPLU: PARTITIONARE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

`partition(A,1,8)`

	1	2	3	4	5	6	7	8
A	3	9	12	5	7	2	9	**5**
	sm			crt				

	1	2	3	4	5	6	7	8
A	3	9	12	5	7	2	9	**5**
		sm		crt				

	1	2	3	4	5	6	7	8
A	3	**5**	12	**9**	7	2	9	**5**
		sm		crt				

EXEMPLU: PARTITIONARE

`partition(A,1,8)`

	1	2	3	4	5	6	7	8
A	3	5	12	9	7	2	9	**5**

sm *crt*

EXEMPLU: PARTITIONARE

`partition(A,1,8)`

	1	2	3	4	5	6	7	8
A	3	5	12	9	7	2	9	**5**
		sm				crt		

	1	2	3	4	5	6	7	8
A	3	5	12	9	7	2	9	**5**
			sm			crt		

	1	2	3	4	5	6	7	8
A	3	5	2	9	7	12	9	**5**
			sm			crt		

EXEMPLU: PARTITIONARE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

`partition(A,1,8)`

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
A	3	5	2	9	7	12	9	**5**
			sm				*crt*	

EXEMPLU: PARTITIONARE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

`partition(A,1,8)`

	1	2	3	4	5	6	7	8
A	3	5	2	9	7	12	9	**5**

sm *crt*

	1	2	3	4	5	6	7	8
A	3	5	2	**5**	7	12	9	**9**

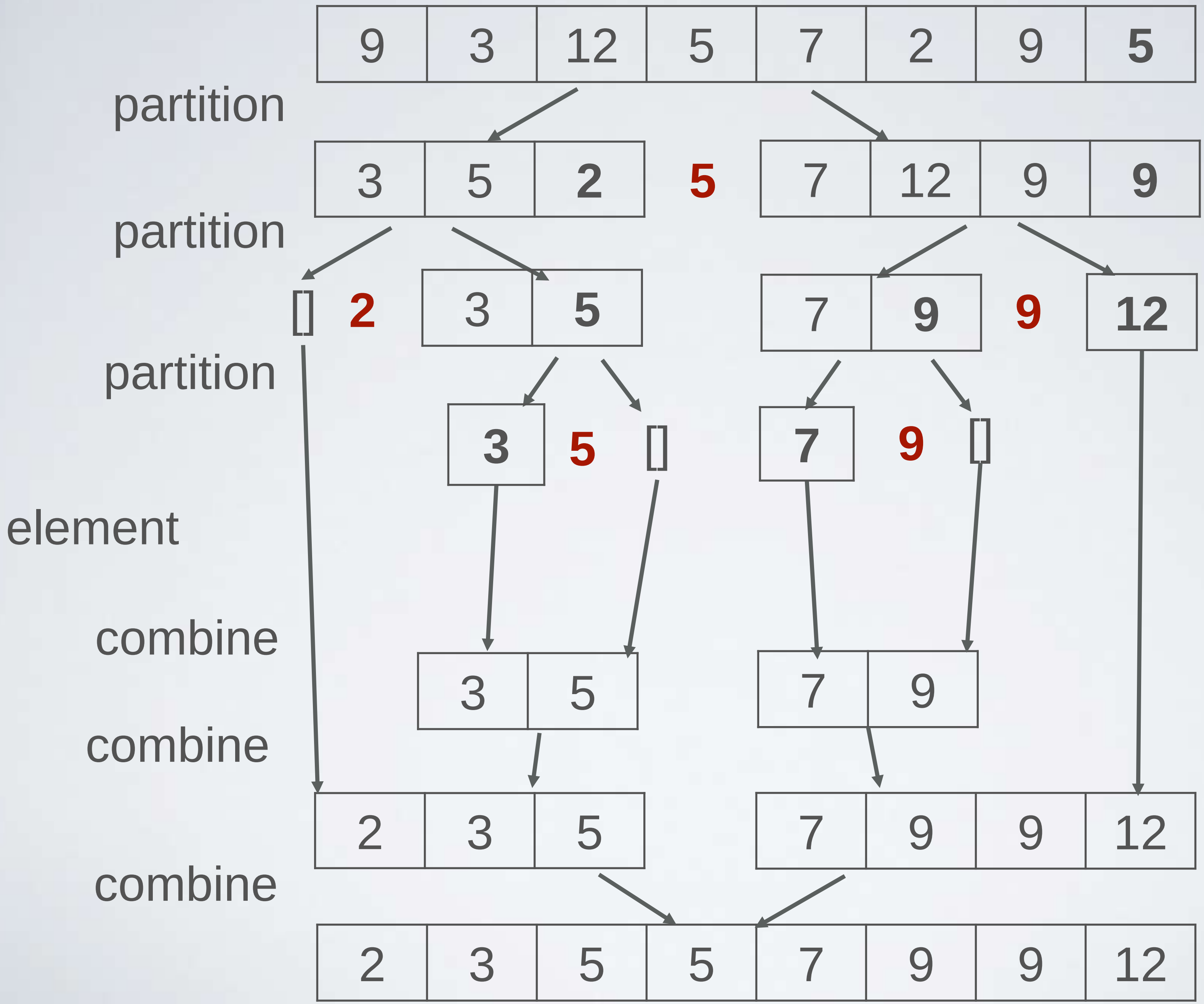
sm *crt*

`partition(A,1,8)` va returna valoarea 4 (indexul/pozitia pivotul dupa partitionare)

QUICKSORT: PSEUDOCOD


```
QUICK-SORT(A, low, high)
   if low < high
      then q = PARTITION(A, low, high)
      QUICK-SORT(A, low, q-1)
      QUICK-SORT(A, q+1, high)
```

```
PARTITION(A, low, high)
   pivot = A[high]
   sm = low - 1
   for crt = low to high-1
       if A[crt] ≤ pivot
       then  sm = sm + 1
             swap(A[crt], A[sm])
   swap (A[sm+1], A[high])
   return sm+1
```



UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

QUICKSORT: ANALIZA

- Spre deosebire de MergeSort, arborele de recursie la QuickSort poate avea inaltime variabila
  - *De ce?*
  - $H_{min}=?$, $H_{max}=?$
- Caz mediu:
  - Pp. arbore de recursie aproximativ echilibrat, H ~ *logn*: *O(nlogn)*
- Caz defavorabil (cand?)
  - Arbore de recursie degenerat, H ~ *n*: $O(n^2)$
- In practica
  - Quicksort are constanta multiplicativa f. mica
  - Foarte eficient in cazul mediu
  - Cazul defavorabil se poate evita (probabilitate extrem de mica de a ajunge in el) relativ usor
    - Selectia randomizata a pivotului

CONSTRUIRE ABC PERFECT ECHILIBRAT

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Problema: Se da un sir de intregi, sortat; se cere sa se construiasca un ABC perfect echilibrat (diferenta intre nr. de noduri dintre subarborele stang si drept este cel mult `1`, la orice nod din arbore)

```
BuidPBT(A, low, high)
if low > high
    then return NIL //empty node
mid = ⌊(low + high)/2 ⌋
node = createNewNode(mid)
node.left = BuidPBT(A,low,mid-1)
node.right = BuidPBT(A,mid+1, high)
return node
```

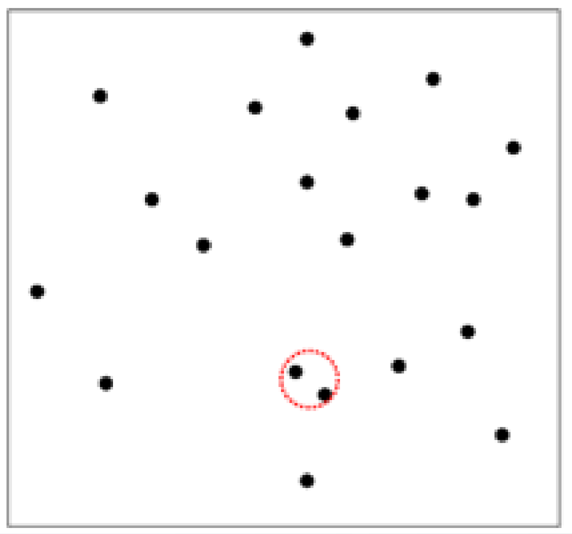
Complexitate?

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
A	2	3	5	5	7	9	9	12

Ce arbore va rezulta in urma apelului BuildPBT(A,1,8)?

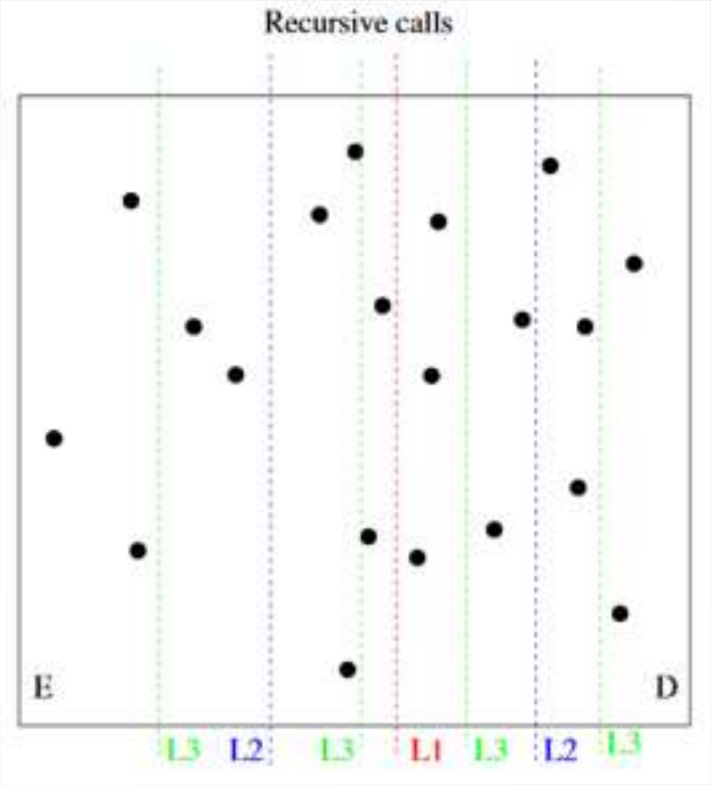
CEA MAI APROPIATA PERECHE DE PUNCTE IN 2D

- Problema: se da o multime de puncte in plan; gasiti perechea de puncte cu cea mai mica distanta Euclideana
- Presupunere: Nu exista 2 puncte cu aceeasi valoare pt. coordonata *x*
- Algoritmul de forta bruta: Calculeaza distanta intre oricare pereche (i, j) de puncte si o compara cu celelalte: $O(n^2)$
- 1D: sortare dupa coordinate O(n lg n)
- Metoda sortarii nu generalizeaza in spatiu multi-dimensional (e.g. 2D). De ce?



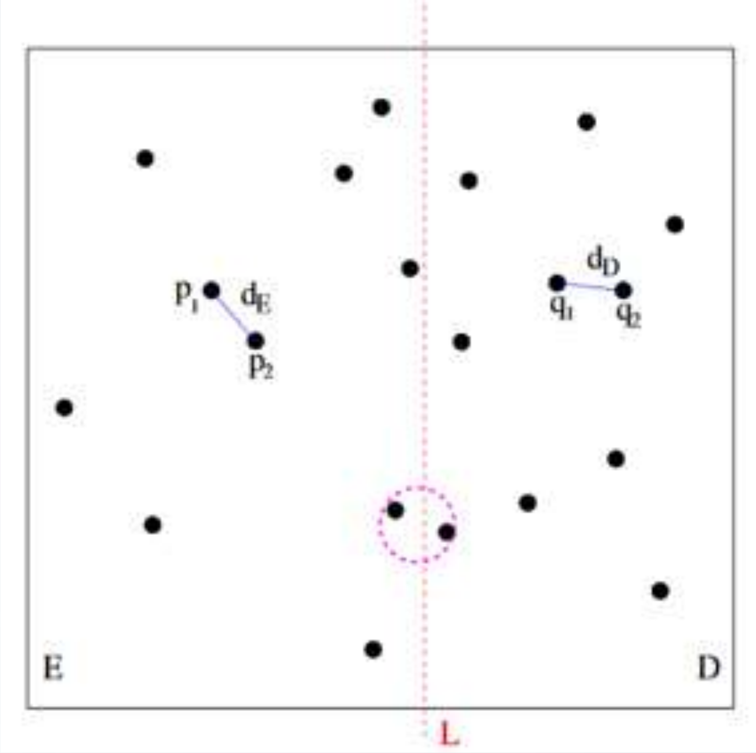
CEA MAI APROPIATA PERECHE DE PUNCTE IN 2D

- <u>Divide</u>: se imparte planul printr-o dreapta L, in 2 jumatati - E si D - avand un numar aproximativ egal de pcte (+/- 1)
- <u>Conquer</u>: recursiv se gaseste distanta minima dintre puncte aflate in fiecare partitie
- <u>Combine</u>: se iau in considerare perechi de puncte (p, q) de pe frontiera, i.e. cu p ∈ E, si q ∈ D



CEA MAI APROPIATA PERECHE DE PUNCTE IN 2D

1. <u>Divide</u>:
   1. se sorteaza cele n puncte dupa coord x
   2. se alege L, a.i. $\left\lceil \frac{n}{2} \right\rceil$ sa ajunga in stanga (E), respectiv $\left\lfloor \frac{n}{2} \right\rfloor$ sa ajunga in dreapta (D)
2. <u>Conquer</u>: returneaza
   d = min{dE , dD }
3. <u>Combine</u>: s-ar putea sa existe 2 pcte, unul in E, celalalt in D, care sunt mai apropiate decat distanta *d* (puncte de langa frontiera L)

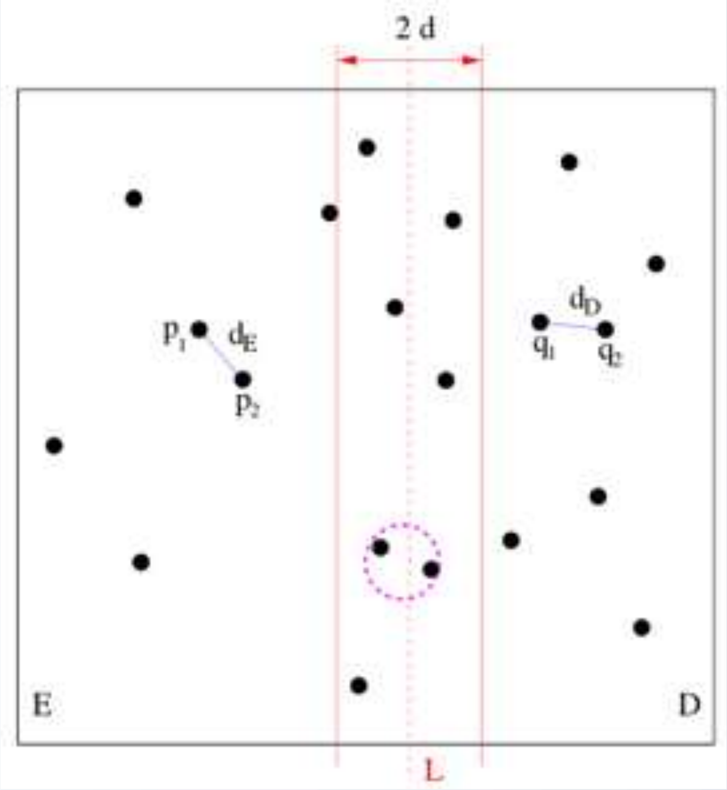


CEA MAI APROPIATA PERECHE DE PUNCTE IN 2D

3. Combine:

- Se ia o banda verticala de latime 2d in jurul lui L
- Orice $q_i \in E$ si $q_j \in D$ a.i. $d(q_i, q_j) \le d$ trebuie sa se gaseasca in aceasta banda (de ce?)
- S-ar putea sa gasim mai multe pcte in interiorul benzii – fie *m* numarul de puncte din int. benzii
- Pt a gasi perechea cu cele mai apropiate puncte: sortam crescator punctele dupa coordonata y: $Y = \{y_1, y_2, \ldots y_m\}$, si:

```
minD = d
for i=1 to m
   j=i+1
   while yj-yi < d
      minD = min(minD, dist(qi, qj))
      j=j+1
```



Corect? Complexitatea pasului de Combine?

CEA MAI APROPIATA PERECHE DE PUNCTE IN 2D: PSEUDOCOD

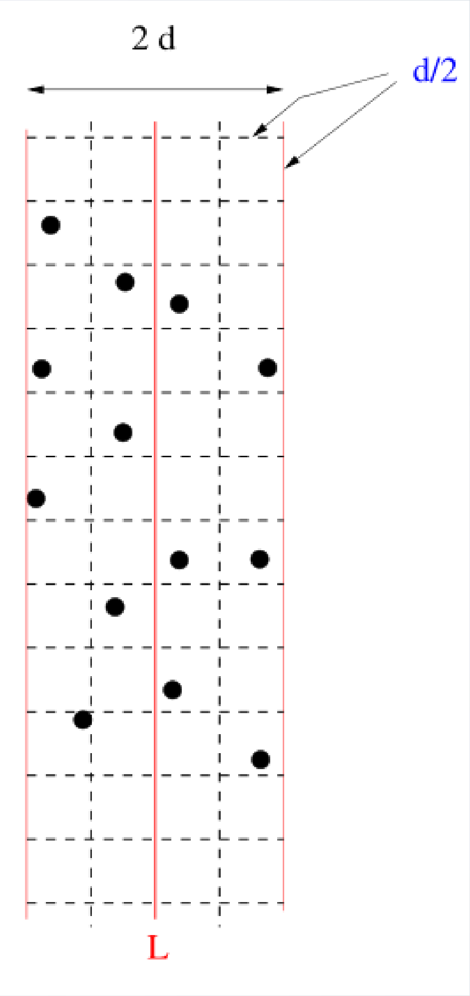
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

```
ClosestPair(p1, …, pn)
1. if n≤3, solve using brute force approach
2. Sort by x-coord to compute L
3. Split points into E and D //left and right
4. dE = ClosestPair(E)
5. dD = ClosestPair(D)
6. d = min(dE, dD)
7. Delete points > d from L
8. Sort remaining points by y-coord
9. minD = d
10. for i=1 to m
11.     j=i+1
12.     while yj-yi < d
13.         minD = min(minD, dist(qi, qj))
14.          j=j+1
15.  return minD
```

Aparent: $O(n^2)$

CEA MAI APROPIATA PERECHE DE PUNCTE IN 2D – ANALIZA

- Cate celule pot sa influenteze 1 punct?
  - Se considera un grid avand dimensiunea celulei $d/2$ in interiorul benzii
  - Exista cel mult `1` punct in interiorul fiecarei celule $d/2 \times d/2$ (diagonala celulei = $d/\sqrt{2} < d$)
  - 2 pcte aflate la o distanta > 2 randuri au sigur o distanta > d (distanta max. intre 2 pcte in celule consecutive este $d\sqrt{\frac{5}{4}} = 1.18d > d$)
  - 2 pcte aflate la o distanta > 2 coloane au sigur o distanta > d (acelasi argument ca inainte)

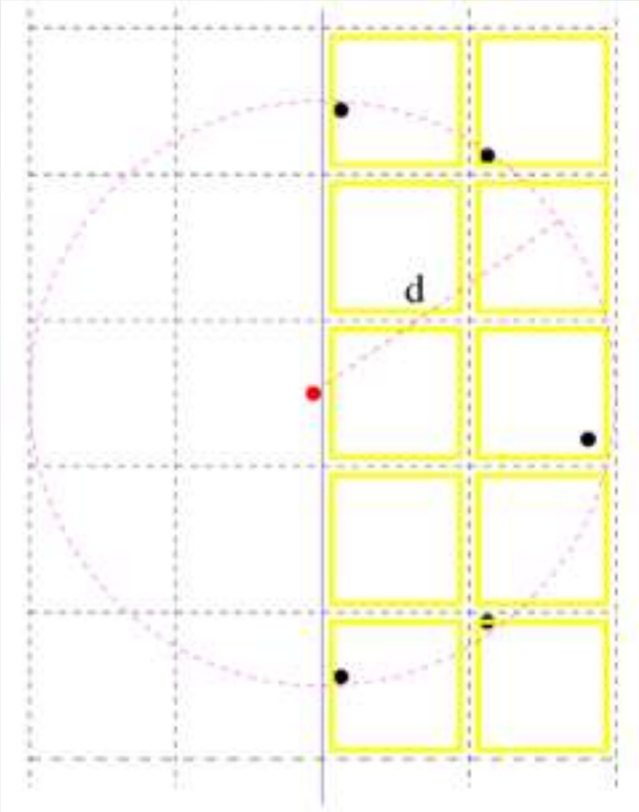


CEA MAI APROPIATA PERECHE DE PUNCTE IN 2D – ANALIZA

- Cate celule pot sa influenteze 1 punct?
  - $d(q_i , q_j) \leq d$ if $|j - i| \leq 10$
- Deci, fiecare punct q_i din banda trebuie comparat cu urmatoarele 10 puncte q_j de dupa el (de ce?)
- $T(n) = 2T(n/2) + O(nlogn)$

  $= O(nlg^2n)$
- Daaar … daca dam pctele gata sortate dupa x si y, alg. mentine ordinea:
  - $T(n) = 2T(n/2) + O(n)$

    $= O(nlogn)$



ALTI ALGORITMI CE UTILIZEAZA D&C

- *QuickSelect*: selectia elementului de rang k dintr-o secventa neordonata
  - Rang k = al k-lea cel mai mic element din secventa
  - F similar cu Quicksort, doar ca e cautare (1 apel recursiv)
- Inmultire matrici
- Inmultire intregi mari
- Inaltime/diametru ABC

BIBLIOGRAFIE

- Th. Cormen et al.: Introduction to Algorithms, cap. 4, cap. 2.3, cap. 7

PROBLEME PROPUSE (A)

1. Se citeste un vector cu n elemente numere naturale. Folosind tehnica divide et impera sa se determine:
   a) Minimul/maximul din vector
   b) Suma si produsul elementelor din vector
   c) Cel mai mare divizor comun al elementelor din vector
   d) Numarul elementelor impare din vector
   e) Numarul de aparitii al unei valori x in vector
   f) Suma elementelor pare din vector
   g) Numarul elementelor prime din vector
   h) Daca vectorul contine elemente cu exact k divizori
   i) Numarul de elemente din vector mai mici decat o valoare data, x.
2. Se dau doua siruri de numere, A, B ordonate crescator. Fiecare sir are dimensiunea n.
   a) Scrieti un algoritm care determina elementul de pe pozitia mediana a sirului C care se obtine prin interclasarea sirurilor A si B.
   b) Scrieti un algoritm care determina elemental de pe o pozitie oarecare, k a sirului C care se obtine prin interclasarea sirurilor A si B.
      Exemplu:
      A = {2, 3, 6, 7, 9}, B = {1, 4, 8, 10, 17} => C = {1,2,3,4,6,7,8,9,10,17};

Elementul median: 7

Elementul de pe pozitia k = 3 este 4;

PROBLEME PROPUSE (B)

1. Descrieti un algoritm eficient pentru gasirea sumei maxime a unui sub-sir (sub-array) dintr-un sir dat.
   - E.g. pt. A = [-2,-3,4,-1,-2,1,5,-3] va returna 7
   - Hint: solutie si prin D&C, si prin PD

2. Mediana a 2 siruri sortate de dimensiuni egale: se dau 2 siruri, A si B, sortate, fiecare de dimensiune n. Descrieti un algoritm eficient pentru gasirea medianei intregii colectii (i.e. a sirului obtinut prin interclasarea celor 2 siruri date).

3. Se da un sir sortat, pe care il rotim circular spre dreapta un numar de pozitii (necunoscut). Descrieti un algoritm eficient pentru a cauta un element in sirul rotit.