

## ALG 14

### Hledání k-tého nejmenšího prvku

Randomized select

CLRS varianta Partition v Quicksortu

Select metodou medián z mediánů

## Hledání k-tého nejmenšího prvku

- 1. Seřad' seznam/pole a vyber k-tý nejmenší, složitost  $\Theta(N \cdot \log(N))$ .  
Nevýhodou je zbytečné řazení úplně všech dat.**
- 2. Využij Randomized select založený na principu Quick sortu.  
Očekávaná složitost  $\Theta(N)$ , nejhorší možná  $\Theta(N^2)$ .**
- 3. Využij Select založený na principu Quick sortu s výběrem pivota metodou medián z mediánů.  
Zaručená složitost  $\Theta(N)$ ,  
velká režie - vhodná jen pro velká data.**

## Randomized select

**Partition -- dělení na "malé" a "velké" v Quicksortu.**

### Randomized partition

- Vyber prvek na náhodné pozici pole/úseku jako pivot.
- Proved' partition jako v Quicksortu s vybraným pivotem.

### Randomized select (k-tý nejmenší prvek)

- Když je délka úseku rovna 1, vrať jeho jediný prvek. Jinak:
- Proved' Randomized partition tohoto úseku, vznikne levý a pravý podúsek.
- Aplikuj Randomized select rekurzivně buď na levý nebo pravý podúsek, podle toho, v kterém z nich může ležet k-tý nejmenší prvek celého pole.

## Randomized select

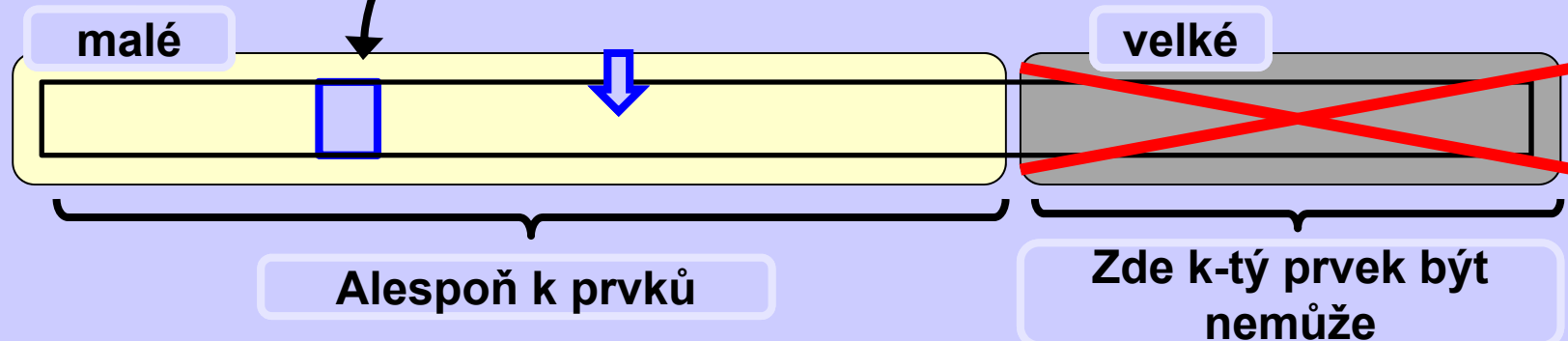
pozice k

k-tý nejmenší (není na pozici k !!)



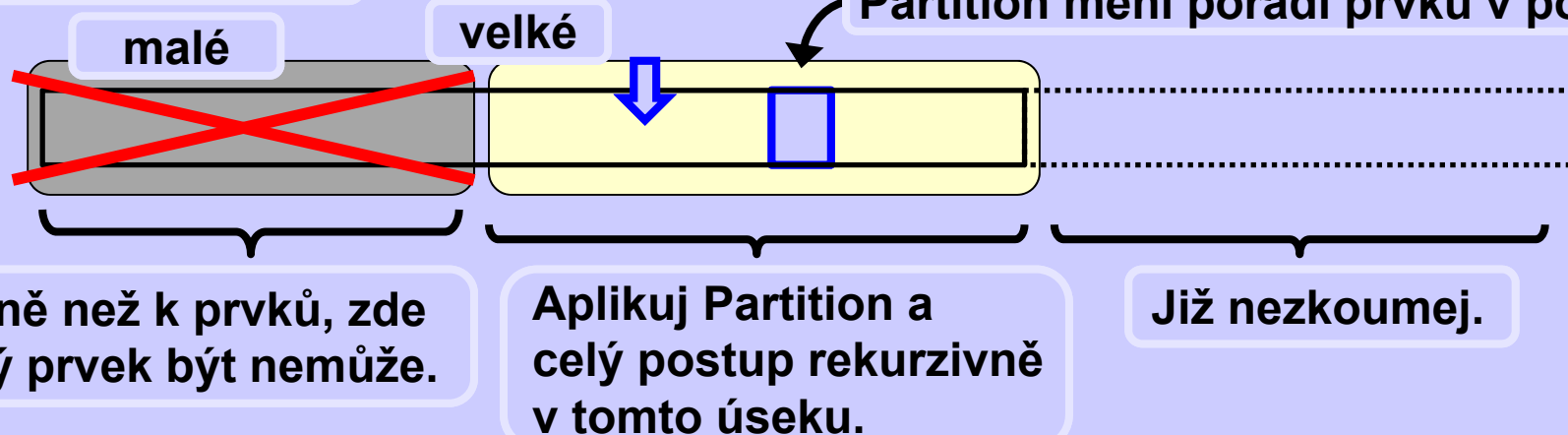
### (Quicksort) Partition

Partition mění pořadí prvků v poli.



### (Quicksort) Partition


Partition mění pořadí prvků v poli.



## Randomized select

Příklad -- hledání 7. nejmenšího prvku

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Pivot 23	14	16	21	11	15	24	12	18	17	25	20	19	23	13	22
Pivot 13	14	16	21	11	15	22	12	18	17	13	20	19	23	25	24
Pivot 14	13	12	11	21	15	22	16	18	17	14	20	19	23	25	24
Pivot 17	13	12	11	14	15	22	16	18	17	21	20	19	23	25	24
Pivot 16	13	12	11	14	15	17	16	18	22	21	20	19	23	25	24
	13	12	11	14	15	16	17	18	22	21	20	19	23	25	24



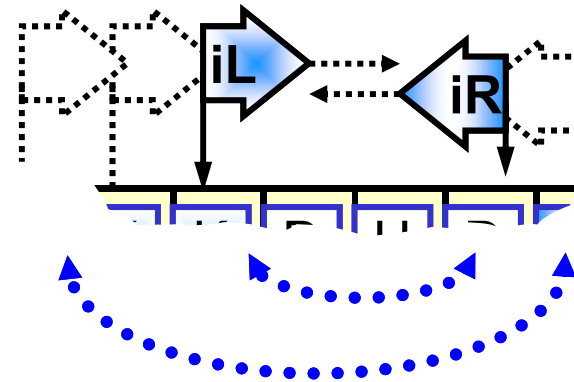
## Randomized select

Dělení úseku pole od indexu L, až po index R včetně na "malé" a "velké" hodnoty podle předem zvolené pivotní hodnoty.

Analogicky jako v Quick sortu.

```
// returns index of the first element in the "big" part
int partition( int [] a, int L, int R, int pivot) {
    int iL = L, iR = R;
    do {
        while (a[iL] < pivot) iL++;
        while (a[iR] > pivot) iR--;
        if (iL < iR)
            swap(a, iL++, iR--);
    } while(iL < iR);

    // if necessary move right by 1
    if ((a[iL] < pivot) || (iL == L)) iL++;
    return iL;
}
```



## Randomized select

```
int quickselect(int [] a, int iL, int iR, int k) {  
    if (iL == iR) return a[iL];           // all done, found  
  
    int pivot = a[randomInt(L, R)];       // random pivot  
    int iMidR = partition(a, L, R, pivot);  
  
    // recursively process **only** the part in which  
    // the k-th element **surely** is present  
    if (k < iMidR)  
        return quickselect(a, iL, iMidR-1, k);  
    else  
        return quickselect(a, iMidR, iR, k);  
}
```

### Příklad volání

```
int k = ...; // whatever value  
int kth_element = quickselect(a, 0, a.length(), k);
```

## Randomized select

### Asymptotická složitost -- podobně jako u Quicksortu

Nevhodná volba pivotu -- složitost může degenerovat až na  $\Theta(n^2)$ , pokud by se v každém kroku zmenšila délka zpracovávaného úseku jen o malou konstantu.

Náhodná volba pivotu činí tuto možnost velmi nepravděpodobnou, prakticky nenastává.

### Očekávaná složitost

Ve skoro každém kroku klesá délka zpracovávaného úseku úměrně konstantě  $\lambda < 1$ , délky úseků pak odpovídají hodnotám

$$N, N\lambda, N\lambda^2, N\lambda^3, \dots,$$

celková délka všech úseků je pak menší než součet řady

$$N + N\lambda + N\lambda^2 + N\lambda^3 + \dots = N(1 + \lambda + \lambda^2 + \lambda^3 + \dots) =$$

$$= N \left( \frac{1}{1-\lambda} \right) = N / (1-\lambda) \in \Theta(N).$$



## Alternativní metoda Partition

V literatuře (např. [CLRS] a na webu, např. Wikipedie) najdeme postup popsany kódem níže. Je koncepčně jednoduchý, fakticky ale pomalejší, faktorem cca 2 až 3.

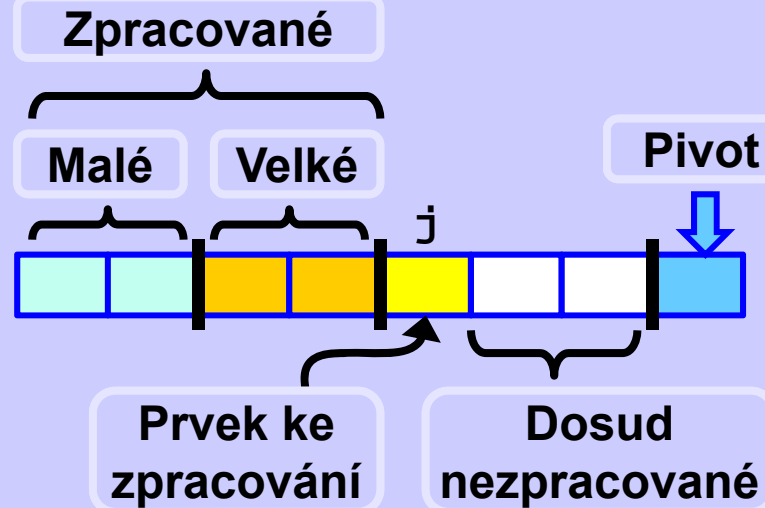
```
// returns index of the first element in the "big" part  
int partitionCLRS(int [] a, int iL, int iR) {  
    int pivot = a[iR];  
    int iMid = iL-1;  
    for (int j = iL; j < iR; j++ )  
        if (a[j] <= pivot)  
            swap(a, ++iMid, j);  
    swap(a, ++iMid, R);  
    return iMid;  
}
```

Swapuje se každý prvek menší nebo roven pivotu, Očekáváme, že cca polovina prvků v úseku bude swapována. V klasické variantě se swapuje mnohem méně.

## Alternativní metoda Partition

### Ukázka průběhu

#### Postup zleva doprava



```

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

```



## Select metodou medián z mediánů

Randomized Select se od Quicksortu liší hlavně tím, že nevolá rekurzi na úsek s "malými" a "velkými" hodnotami, ale jen na jeden z nich, podle toho, v kterém z nich lze očekávat výskyt hledaného k-tého nejmenšího prvku.

Jak si pamatujeme z Quicksortu, pokud je úsek "malých" nebo "velkých" hodnot opakovaně extrémně krátký, může to vést až na kvadratickou složitost Quicksortu.

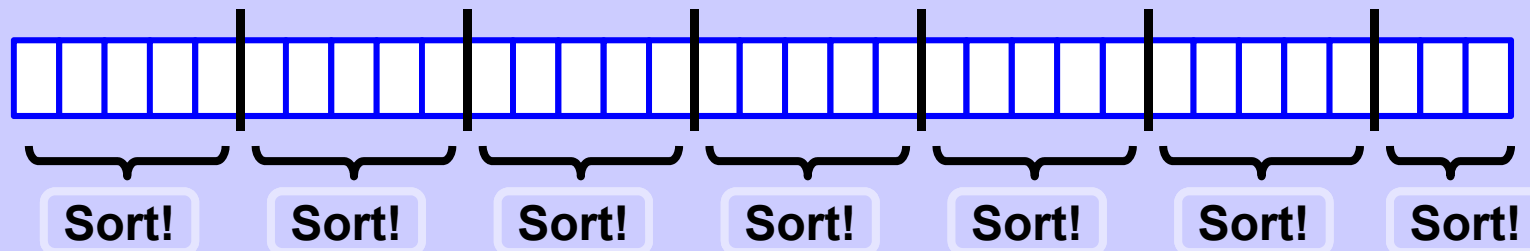
Stejná potíž by mohla nastat i metodě select.

Je nutno zajistit, aby v metodě partition byl vybrán pokaždé dostatečně vhodný pivot, tj. takový, který dobře rozdělí daný úsek na "malé" a "velké", tj. takový, který je pokud možno co nejbližší mediánu hodnot tohoto úseku.

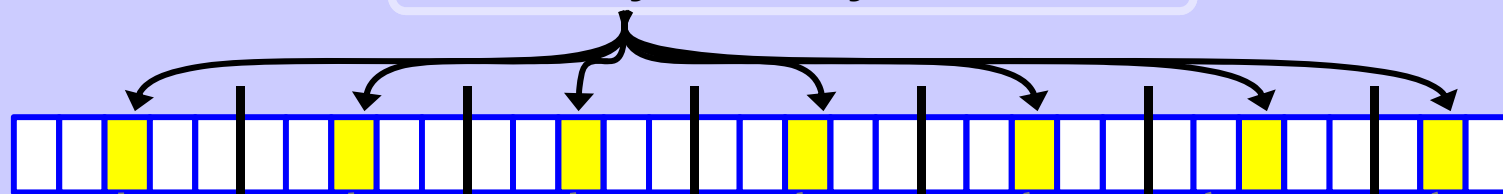
## Select metodou medián z mediánů

Volba pivota pro funkci Partition -- ukázka pro pole délky 33

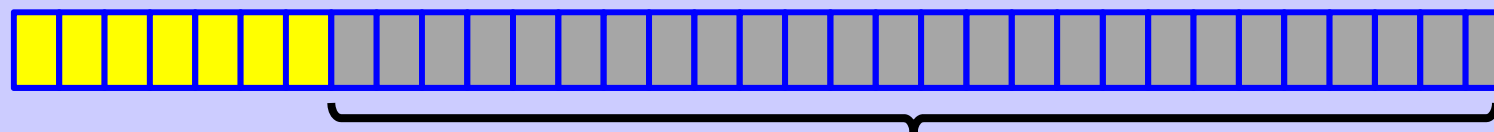
Úseky délky 5



mediány seřazených 5-úseků



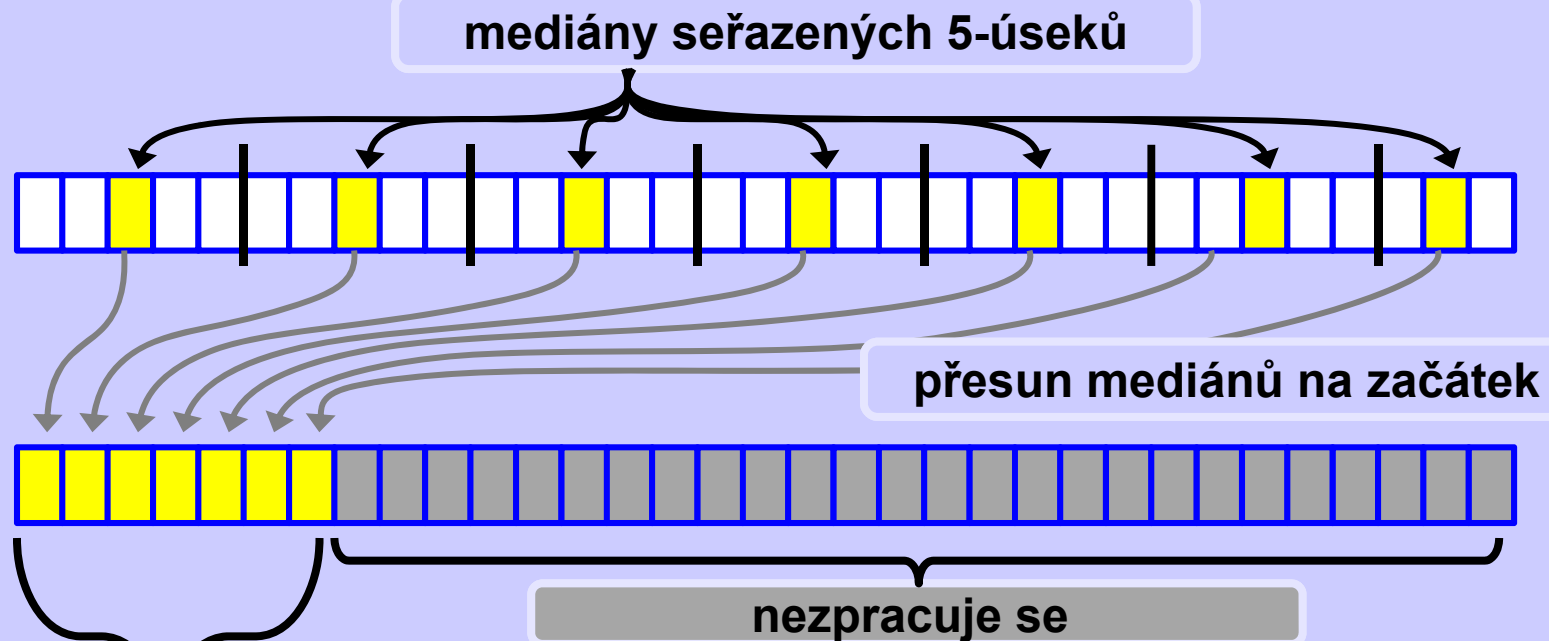
přesun mediánů na začátek



nezpracuje se

## Select metodou medián z mediánů

Volba pivota pro funkci Partition -- ukázka pro pole délky 33



V této části rekurzivní aplikací právě popisované metody najdi medián. To je "medián z mediánů" pro původní pole. Tento medián použij jako pivot ve funkci Partition.

## Select metodou medián z mediánů

0. Pokud má aktuální úsek pole délku 1, vrať jeho jediný prvek.
1. Rozděl pole nebo úsek pole délky  $N$  na  $\lfloor N/5 \rfloor$  skupin po pěti prvcích a nejvýše jednu zbylou s méně než 5 prvky.
2. Každou skupinu seřaď (např. Insert sort) a vyber z ní její medián. Všechny tyto mediány přesuň na začátek pole/úseku.
3. Aplikuj rekurzivně celý postup na úsek obsahující právě nalezené mediány, tak získáš medián původních  $\lceil N/5 \rceil$  mediánů z bodu 2. Tento medián  $M$  bude pivotem.
4. Rozděl pole/úsek na "malé" a "velké" pomocí metody partition s pivotem  $M$ .
5. Podle velikosti úseků "malých" a "velkých" rozhodni, v kterém z nich se nachází  $k$ -tý nejmenší prvek celého pole a na něj aplikuj rekurzivně celý postup.

## Select metodou medián z mediánů

## Rozbor složitosti

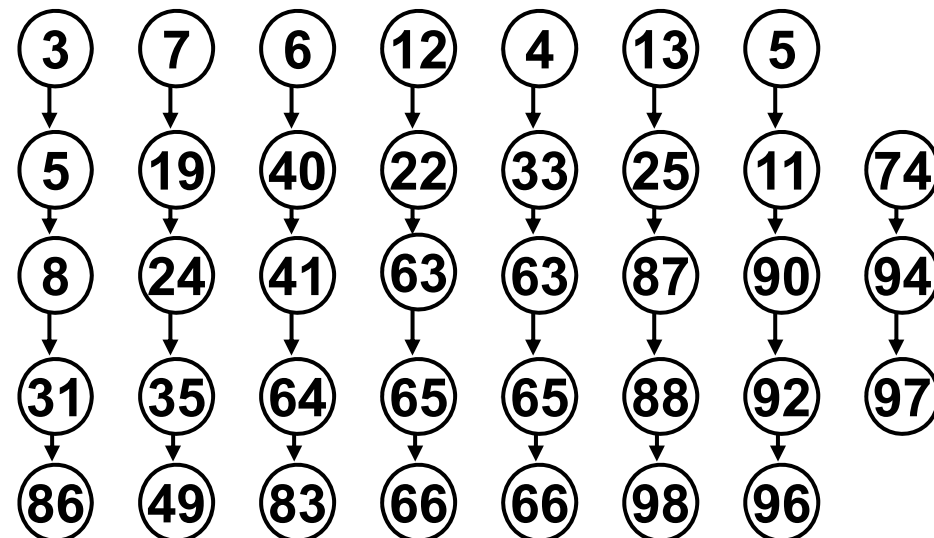
Rozděl na úseky délky 5

5 3 86 31 8	24 19 7 49 35	64 41 40 6 83	22 12 66 65 63	66 33 63 65 4	13 98 88 87 25	90 11 96 92 5	94 74 97
-------------	---------------	---------------	----------------	---------------	----------------	---------------	----------

Seřad' úseky délky 5

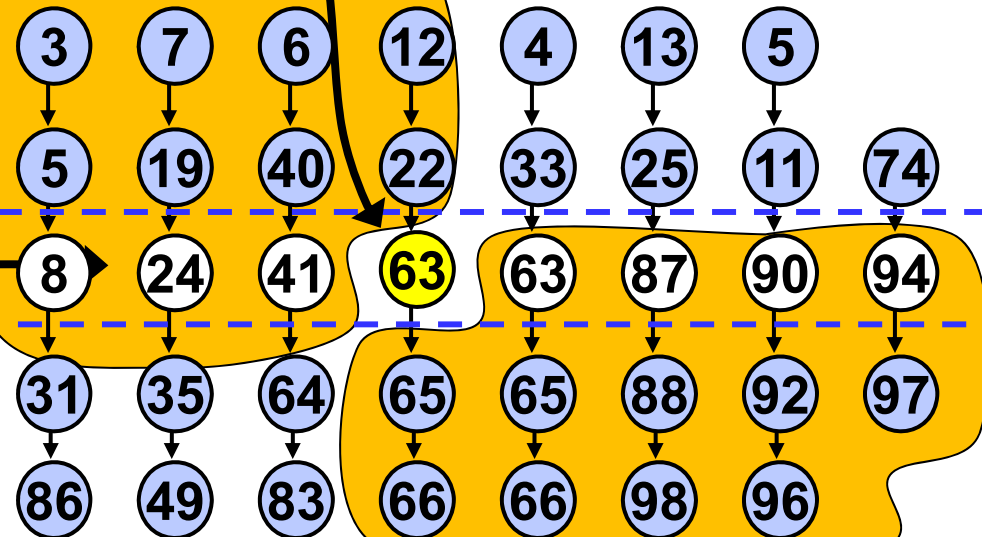
3 5 8 31 86	7 19 24 35 49	6 40 41 64 83	12 22 63 65 66	4 33 63 65 66	13 25 87 88 98	5 11 90 92 96	74 94 97
-------------	---------------	---------------	----------------	---------------	----------------	---------------	----------

Seřazené úseky délky 5 pro úsporu místa a pro větší názornost si nakreslíme vertikálně a zachováme jejich vzájemné pořadí.



## Select metodou medián z mediánů

## Rozbor složitosti

 $M = \text{medián z } \lceil N/5 \rceil \text{ mediánů}$ 
Prvky zaručeně  
menší než M $\lceil N/5 \rceil$  mediánůPrvky zaručeně  
větší než M

Prvků zaručeně větších než M je alespoň  $3N/10 - 6$ .

Analogicky, prvků menších než M je také alespoň  $3N/10 - 6$ .

V bodě 5 se rekurze zavolá na úsek obsahující nejvýše

$N - (3N/10 - 6) = 7N/10 + 6$  prvků. Je tak zaručeno, že v nejhorším případě se úsek zkrátí o alespoň cca 30%.

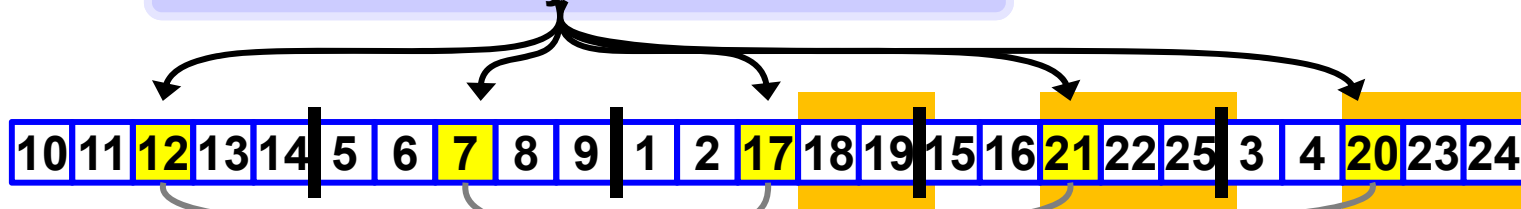


## Select metodou medián z mediánů

## Rozbor složitosti

Medián mediánů není obecně roven mediánu celého souboru, nemůže být ale ani příliš velký ani malý.


## Mediány 5-úseků



Medián mediánů = 17

12 7 17 21 20

Medián pole  $\{1, 2, 3, \dots, 25\} = 13$

Zvýrazněné hodnoty  nemohou být menší než medián mediánů. Těchto hodnot je celkem 8. Analogické zjištění platí pro hodnoty menší než medián mediánů. Pro pole délky 25 s různými prvky je zaručeno, že vybraný pivot bude co do velikosti mezi 9. -- 17. prvkem včetně, tj.  $16/25 = 64\%$  nevýhodných voleb pivotu odpadá.

## Select metodou medián z mediánů

Metoda subSort je obyčejný Insert Sort,  
rychlý na malých úsecích pole.

Zde slouží k nalezení mediánu hodnot v úseku délky 5 nebo menší.

```
void subSort(int [] a, int L, int R) {  
    int insVal, j;  
    for(int i = L+1; i <= R; i++) {  
        insVal = a[i];  
        j = i-1;  
        while ((j >= L) && (a[j] > insVal)) {  
            a[j+1] = a[j];  
            j--;  
        }  
        a[j+1] = insVal;  
    }  
}
```

## Select metodou medián z mediánů

Metoda select volá sama sebe dvakrát.

Poprvé pro nalezení mediánu z mediánů úseků délky 5,  
podruhé pro hledání k-tého prvku celého pole  
mezi buďto "malými nebo "velkými" hodnotami v aktuálním úseku.

```
int select(int [] a, int iL, int iR, int k) {  
    // 0. if the group is short terminate the recursion  
  
    // 1. move medians of groups of 5 to the left  
  
    // 2. also the median of the last possibly smaller group  
  
    // 3. find median M of medians of 5 recursively  
  
    // 4. use M for partition  
  
    // 5. recursively call select on the appropriate part  
  
}
```

## Select metodou medián z mediánů

```

int select(int [] a, int iL, int iR, int k) {

    // 0. if the group is short terminate the recursion
    if (iL == iR) return a[iL];
    if (iL+1 == iR) return (a[iL] < a[iR]) ? a[iL] : a[iR];

    // 1. move medians of groups of 5 to the left
    int lastM = iL; // place for medians
    int i5; // 5-elem groups index
    for (i5 = iL; i5 <= iR-4; i5 += 5) {
        subSort(a, i5, i5+4); // find median of 5
        swap(a, lastM++, i5+2); // put it to the left
    }

    // 2. also the median of the last possibly smaller group
    if (i5 <= iR) {
        subSort(a, i5, iR);
        swap(a, lastM, i5+(iR-i5)/2); // lastM - last of medians
    }

    ... // continue
}

```

## Select metodou medián z mediánů

```
... // continued:  
  
// 3. find median (medmed5) of medians of 5 recursively  
int medmed5 = select(a, iL, lastM, iL+(lastM-iL)/2);  
  
// 4. use medmed5 for partition,  
    a[iBig] = leftmost "big" value  
int iBig = partition(a, iL, iR, medmed5);  
  
// 5. recursively call select on the appropriate part  
if (iBig > k)  
    // the k-th elem is among the "small"  
    return select(a, iL, iBig-1, k);  
else  
    // the k-th elem is among the "big"  
    return select(a, iBig, iR, k);  
  
} // end of select
```

## Select metodou medián z mediánů

### Odvození složitosti

Rekurence:

$$T(N) \leq \begin{cases} O(1) & \text{pro } N < 140 \\ T(\lceil N/5 \rceil) + T(7N/10+6) + O(N) & \text{pro } N \geq 140 \end{cases}$$

**bod 3**
**bod 5**
**body 1, 2, 4**

Řešení substituční metodou, lze ukázat  
 $T(N) \leq cN \in \Theta(N)$ ,  
 pro dostatečně velké  $c$  a pro  $N > 0$ .

Viz [CLRS], kap.9.