UFR d'Informatique

Recherche du k^{ème} élément

2015--2016

F. Laroussinie

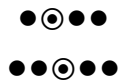
la médiane

Données:
- Un tableau T de taille n ≥ 1
- Des éléments tous distincts deux à deux

Résultat:
l'élément x de T tel que:

$|\{ y \in T \mid y \leq x \}| = \lceil n/2 \rceil$

$|\{ y \in T \mid y > x \}| = \lfloor n/2 \rfloor$



le problème du «k^{ème}elt»

Données:
- Un tableau T de taille n ≥ 1
- Des éléments tous distincts deux à deux
- Un entier k

Résultat:
L'élément x de T tel que:

$|\{ y \in T \mid y < x \}| = k-1$

$|\{ y \in T \mid y > x \}| = n-k$

6 algorithmes

- Algo «naif»
- Algo «naif 2»
- Algo «select» (quickselect»)
- Algo «select opt»
- Algo avec des arbres (2 algos)

Algo naif

|T| = n

```
def algonaif(T,k) :
    if k> n : erreur
    for i = 1..k :
        indmin = indiceMin(T)
        if  i< k : T[indmin] = ∞
    return T[indmin]
```

n-1 comparaisons

Complexité: O(k.n)

ou $O(n^2)$ car k≤n/2

Algo (un tout petit peu moins) naif

|T| = n

n-k+1 comparaisons

```
def algonaif'(T,k) :
    if k> n : erreur
    for i = 1..k :
        indmin = indiceMin(T[1...n-k+2])
        if i≤k-2 : T[indmin]=T[n-k+2+i]
          else if i==k-1 : T[indmin] = ∞
    return T[indmin]
```

NB: le + petit de n-k+2 valeurs ne peut pas être le $k^{ème}$ plus petit des n valeurs !

Exemple: n=9 éléments, k=3
Le plus petit de 8 éléments ne peut pas être le 3ème plus petit…
C'est soit le plus petit, soit le 2ème plus petit.

Algo naif 2

Complexité: O(n log n)

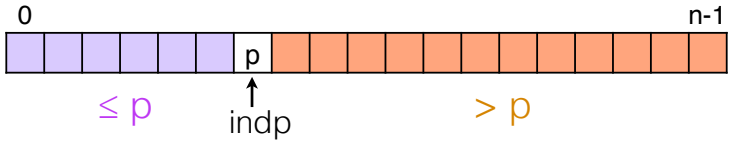
- Trier T[1..n]
- retourner T[k]

Algorithme Select (ou quickselect)

Algorithme select

Basé sur le quicksort («quickselect»).

On pivote T selon un pivot p:



On continue en cherchant le...
- k^{eme} élément sur la partie gauche si k<indp+1
- k-indp-1^{eme} élément sur la partie droite si k>indp+1

et on s'arrête si k=indp+1 !

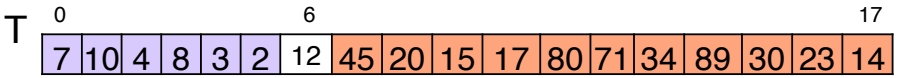
Algorithme select

inv: k ∈ {1,...,bd-bg+1} et bg ≤ bd

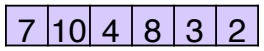
```
def select(T,k,bg,bd) :
    indp = indicepivot(T,bg,bd)
    rangpivot = indp-bg+1        → rang du pivot dans T[bg..bd]
    if (k<rangpivot) :
            return select(T,k,bg,indp-1)
    elif (k>rangpivot) :
            return select(T,k-rangpivot,indp+1,bd)
    else :
        return T[indp]
```

Exercice: vérifier l'invariant...

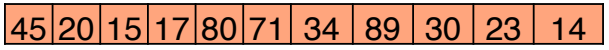
Exemple



- Si k= 7, alors le $k^{ème}$ plus petit est 12 !
- Si k<7, alors le $k^{ème}$ plus petit de T est le $k^{ème}$ plus petit de



- Si k>7, alors le k-ème plus petit de T est le k-$7^{ème}$ plus petit de



Pivotage...

```
def indicepivot(T,bg,bd) :
    p = T[bg]        → pivot = premier élément
    l=bg+1
    r=bd
    while (l<=r) :
        while (l<=bd) and (T[l]<=p) :  l += 1
        while (T[r] > p) : r  -= 1
        if (l < r) :
            T[r], T[l] = T[l], T[r]
            l , r = l+1, r-1
    T[r], T[bg]  = p, T[r]
    return r
```

Algorithme select

Complexité: $O(n^2)$

```
def select(T,k,bg,bd) :
    indp = indicepivot(T,bg,bd)
    rangpivot = indp-bg+1
    if (k<rangpivot) :
            return select(T,k,bg,indp-1)
    elif (k>rangpivot) :
            return select(T,k-rangpivot,indp+1,bd)
    else :
        return T[indp]
```

cas pire: recherche du plus petit élt dans T trié dans l'ordre décroissant.

Algorithme «Select-opt»

Algo «select-opt»

(proposé par Blum, Floyd, Pratt, Rivest et Tarjan en 1973)

- Si n est petit, utiliser l'algo select.
- Sinon:
  - diviser les n éléments en $\lfloor n/5 \rfloor$ blocs B_i de 5 éléments.
  - trouver le médian m_i de chaque B_i
  - trouver le médian M des m_i
  - pivoter avec M comme pivot
  - continuer dans la bonne partie du tableau (comme dans select)...

Idée: garantir que la prochaine étape se fera sur une fraction du tableau initial...

Algo «select-opt»

IndiceMedianQuintuplet(T,i): retourne l'ind. du médian de T[i],...,T[i+4]

```
def selectopt(T,k,bg,bd) :
    n = bd - bg + 1
    if (n < 50) : return select(T,k,bg,bd)[0]

    Taux = [T[IndiceMedianQuintuplet(T,bg+j*5)]   for   j = 0... ⌊n/5⌋ -1  ]

    M  = selectopt(Taux, ⌈|Taux|/2⌉ )
    indp = indicepivotfixe(T,M,bg,bd)
    rangpivot = indp-bg+1

    if (k < rangpivot) : return selectopt(T,k,bg,indp-1)
    elif (k > rangpivot) : return selectopt(T,k-rangpivot,indp+1,bd)
    else :
        return T[indp]
```

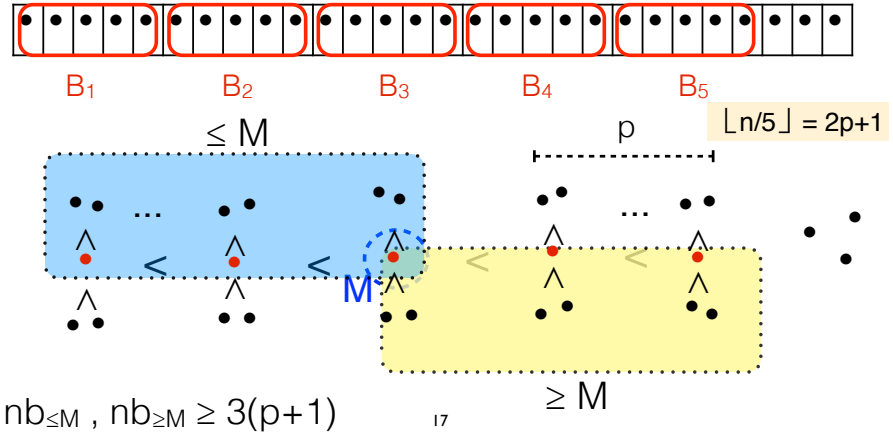
«2» appels récursifs à selectopt !

pivotage selon M

Algo «select-opt» - complexité

La complexité de l'algorithme est bonne car le pivot M choisi n'est jamais «trop mauvais»...

Comment évaluer le nb d'éléts ≤M et ≥M ?

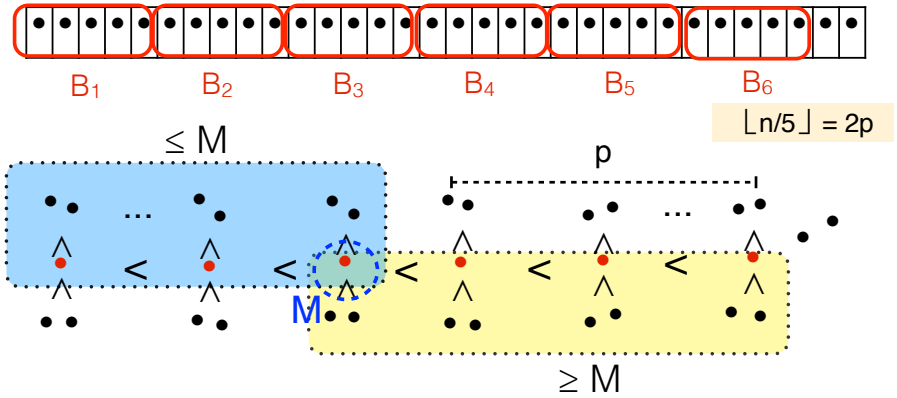


$nb_{\le M}$, $nb_{\ge M} \ge 3(p+1)$

Algo «select-opt» - complexité

La complexité de l'algorithme est bonne car le pivot M choisi n'est jamais «trop mauvais»...

Comment évaluer le nb d'éléts ≤M et ≥M ?



$nb_{\le M} \ge 3p$ $nb_{\ge M} \ge 3(p+1)$

Algo «select-opt» - complexité

1) $\lfloor n/5 \rfloor = 2p$, $nb_{\le M} \ge 3p$ $nb_{\ge M} \ge 3(p+1)$

$p = \frac{1}{2} \lfloor n/5 \rfloor = \lceil \frac{1}{2} \lfloor n/5 \rfloor \rceil$

$p+1 = \lceil \frac{1}{2}(2p+1) \rceil = \lceil \frac{1}{2}(\lfloor n/5 \rfloor +1) \rceil$

2) $\lfloor n/5 \rfloor = 2p+1$, $nb_{\le M}$, $nb_{\ge M} \ge 3(p+1)$

$p+1 = \lceil \frac{1}{2} (\lfloor n/5 \rfloor) \rceil$

$p+1 = \frac{1}{2} (\lfloor n/5 \rfloor +1) = \lceil \frac{1}{2} (\lfloor n/5 \rfloor +1) \rceil$

Conclusion:

$nb_{\le M} \ge 3 \lceil \frac{1}{2} \lfloor n/5 \rfloor \rceil$

$nb_{\ge M} \ge 3 \lceil \frac{1}{2} (\lfloor n/5 \rfloor +1) \rceil \ge 3 \lceil \frac{1}{2} \lfloor n/5 \rfloor \rceil$

Algo «select-opt» - complexité

$nb_{\le M}$, $nb_{\ge M} \ge 3 \lceil \frac{1}{2} \lfloor n/5 \rfloor \rceil$

comme $\lfloor n/5 \rfloor \ge (n-4)/5$, on a:

$nb_{\le M}$, $nb_{\ge M} \ge 3 \lceil \frac{1}{2} \lfloor n/5 \rfloor \rceil \ge 3/2 \cdot \lfloor n/5 \rfloor \ge (3n-12)/10$

$\ge 3n/10 - 2$

Les appels récursifs se font donc sur des sous-tableaux de taille ≤ n-3n/10+2, donc ≤ 7n/10+2.

$$C(n) = C(\lfloor n/5 \rfloor) + C(7n/10+2) + O(n)$$

recherche de M — suite du calcul

Algo «select-opt» - complexité

$C(n) = C(\lfloor n/5 \rfloor) + C(7n/10+2) + O(n)$

Proposition:
Si $t(n) = \sum_i a_i\, t(n/b_i) + c.n$ avec $\sum_i a_i/b_i < 1$
alors $t(n) = \Theta(n)$

Corollaire: $C(n) = \Theta(n)$

L'algorithme select-opt est en temps linaire !

Algorithme avec des arbres

k^{eme} élément et TAS

```
Construire un TAS T avec les n éléments.
Pour i=1 à k:
    x = ExtraireMin(T)
Retourner x
```

Complexité:
- En O(n) pour la construction du tas.
- O(*log*(n)) pour l'extraction du min.

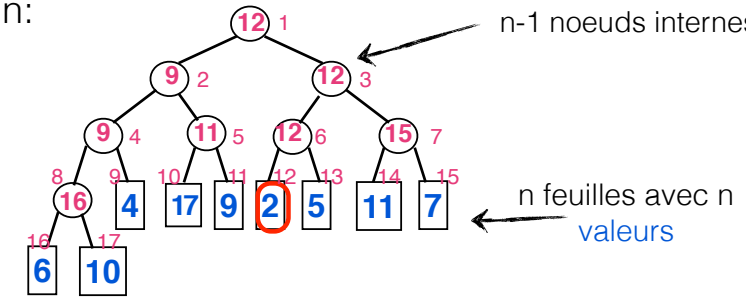
→ O(n + k *log*(n))

Algorithme avec des arbres Min

Voir «The Art of Computer Programming», volume 3, D. Knuth.

k^eme élément et arbres min

Arbre min:
n=9

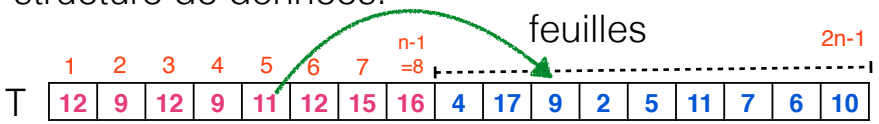


arbre binaire parfait...
9 feuilles + 8 noeuds internes numérotés de 1 à 17

calcul: indiquer le numéro de la feuille de valeur min dans le sous-arbre.

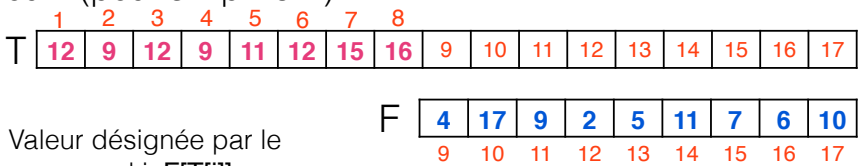
k^eme élément et arbres min

structure de données:



Valeur désignée par le noeud i: T[T[i]] si i<n, ou T[i] si i ≥ n.

ou... (pour simplifier !)



Valeur désignée par le noeud i: F[T[i]]

k^ème élément et arbres min

Arbre binaire parfait: hauteur ≤ ⌈*log*(n)⌉

Calculer les n° de feuilles des noeuds internes : CalculArbre
n-1 noeuds internes... n-1 comparaisons : O(n) !

Changer une valeur d'une feuille et recalculer : MaJ
O(log n) !

idée:
1) On fait un CalculArbre pour trouver le min…
2) Et k-1 mises à jour (après extraction du min précédent).

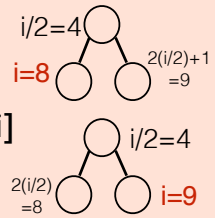
k^eme élément et arbres min

noeuds 1…n-1 : noeuds internes

```
def CalculArbre(T,F,n) :
  for i =n-1..1 :
     if (F[T[2i]] <= F[T[2i+1]]) : T[i] = T[2i]
     else : T[i] = T[2i+1]
```

```
def MaJ(T,v,nf,F) :
  F[nf] = v
  i = nf
  while (i/2 >= 1) :
     i = i/2
     if (F[T[2i]] <= F[T[2i+1]]) : T[i] = T[2i]
     else :T[i]=T[2i+1]
```

v: nouvelle valeur
nf: numéro de la feuille



k^eme élément et arbres min

Algo-kpp (V[1...n],k) :
1) appliquer CalculArbre sur un arbre avec n-k+2 val:
V[1]... V[n-k+2]
2) Pour i = 1 à k-2:
2') Appliquer MaJ(T,V[n-k+2+i],T[1],F)

3) Renvoyer le 2ème plus petit élément de l'arbre.

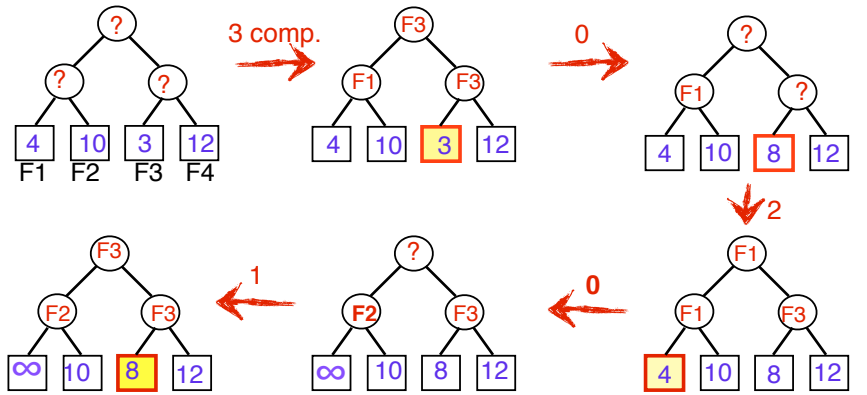
1) MaJ(T,∞,T[1],F)
2) retourner F[T[1]]

NB: le + petit de n-k+2 valeurs ne peut pas être le k^ème + petit des n valeurs !

k^eme élément et arbres min

Le cas des **quintuplets**. Par ex: ici n-k+2 = 4

4	10	3	12	8



Total = 6 comparaisons suffisent pour trouver le median de 5 eleᵗˢ.

k^eme élément et arbres min

Complexité:

$$O\left(n-k + (k-1)\log(n-k+2)\right)$$

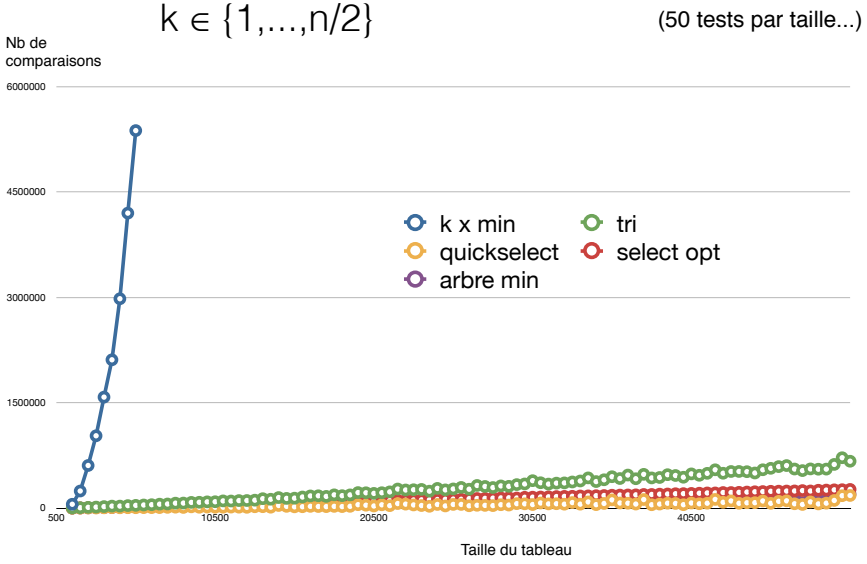
CalculArbre

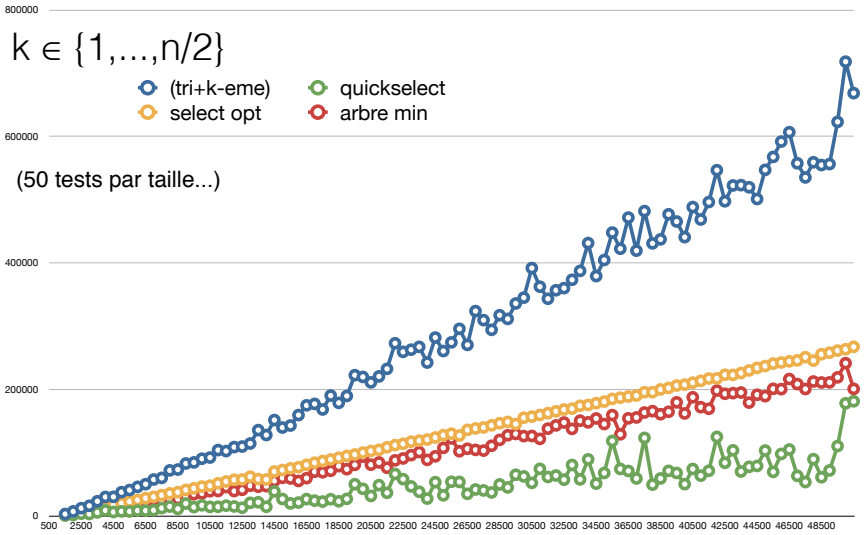
k-1 MaJ

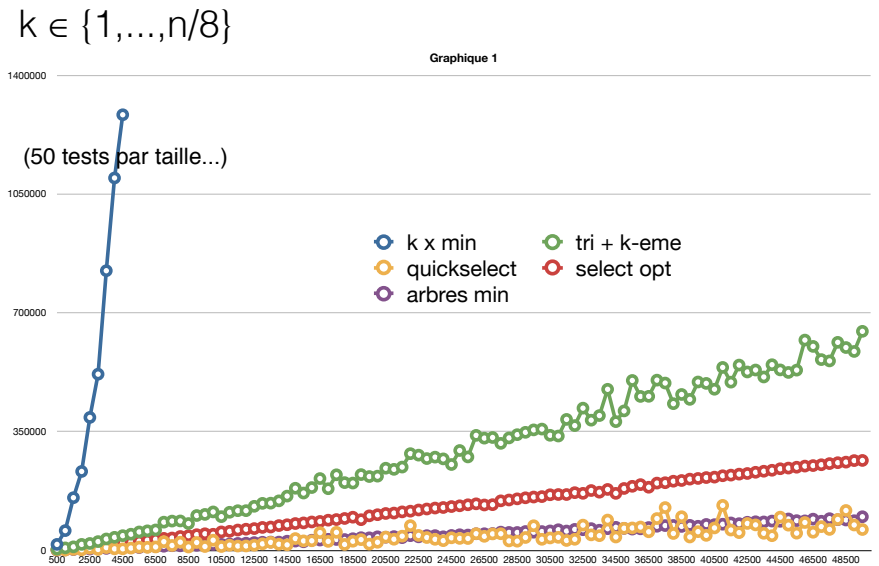
Conclusion 1

naif	naif 2	select	select opt	arbres min et tas
O(k.n) $O(n^2)$	O(n.log n)	$O(n^2)$	O(n)	O(n +k.log(n))

Test...


k ∈ {1,...,n/2}
(50 tests par taille...)
Nb de comparaisons
6000000
4500000
3000000
1500000
0
500
10500
20500
30500
40500
Taille du tableau
k x min
quickselect
arbre min
tri
select opt


k ∈ {1,...,n/2}
(50 tests par taille...)
800000
600000
400000
200000
0
500 2500 4500 6500 8500 10500 12500 14500 16500 18500 20500 22500 24500 26500 28500 30500 32500 34500 36500 38500 40500 42500 44500 46500 48500
(tri+k-eme)
select opt
quickselect
arbre min


k ∈ {1,...,n/8}
Graphique 1
(50 tests par taille...)
1400000
1050000
700000
350000
0
500 2500 4500 6500 8500 10500 12500 14500 16500 18500 20500 22500 24500 26500 28500 30500 32500 34500 36500 38500 40500 42500 44500 46500 48500
k x min
quickselect
arbres min
tri + k-eme
select opt

$k \in \{1,\ldots,n/8\}$