UFR d'Informatique

Recherche du k^{ème} élément

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

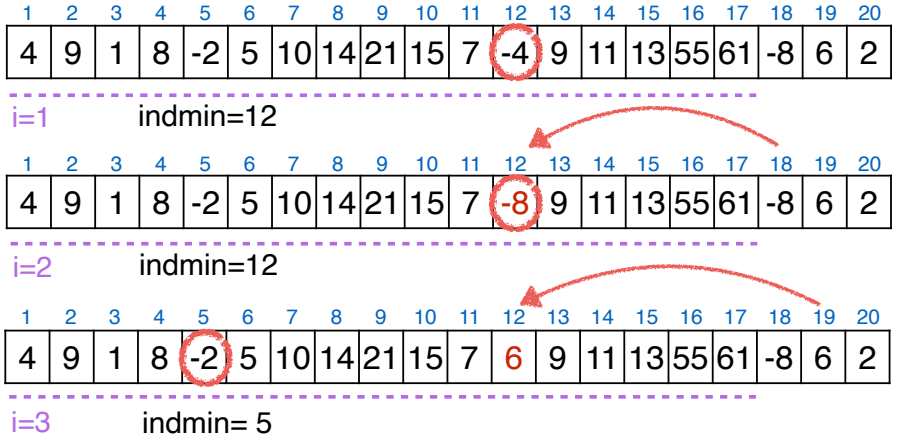
```
def algonaif'(T,k) :
    if k> n : erreur
    for i = 1..k :
        indmin = indiceMin(T[1...n-k+2])
        if i≤k-2 : T[indmin]=T[n-k+2+i]
          else if i==k-1 : T[indmin] = ∞
    return T[indmin]
```

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.

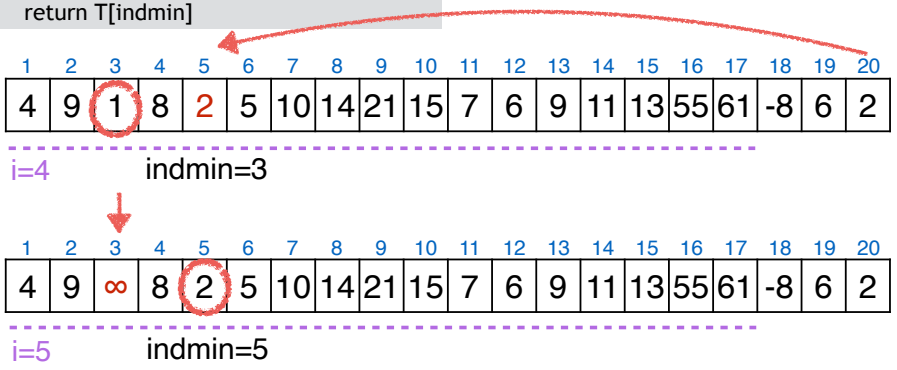

```
def algonaif'(T,k) :
    if k> n : erreur
    for i = 1..k :
        indmin = indiceMin(T[1...n-k+2])
        if i≤k-2 : T[indmin]=T[n-k+2+i]
          else if i==k-1 : T[indmin] = ∞
    return T[indmin]
```


n=20
k=5




```
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]
```

n=20
k=5



resultat = 2

Algo (un tout petit peu moins) naif

complexité

|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]
```

Complexité: O(k.n)
ou $O(n^2)$ car k≤n/2

Algo naif 2

Complexité: O(n log n)

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

1er algo bien

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.

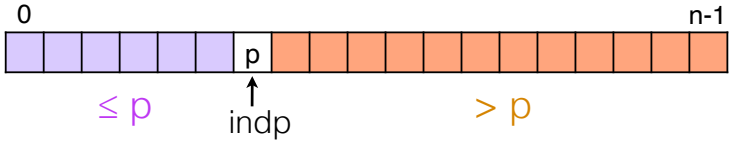
→ O(n + k *log*(n))

Algorithme Select (ou quickselect)

Algorithme select

Basé sur le quicksort («quickselect»).

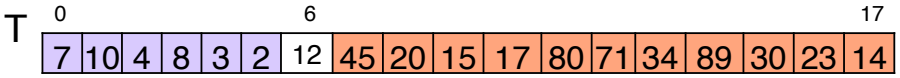
On pivote T selon un pivot p:



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 !

Exemple



indp = 6

- Si k= 7, alors le $k^{ème}$ plus petit est le pivot (ie12)!
- Si k<7, alors le $k^{ème}$ plus petit de T est le $k^{ème}$ plus petit de 7 10 4 8 3 2
- Si k>7, alors le k-ème plus petit de T est le k-$7^{ème}$ plus petit de 45 20 15 17 80 71 34 89 30 23 14

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...

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 élement dans T trié dans l'ordre décroissant.

Algorithme select

```
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]
```

Et en moyenne ?

$C(n,k)$= coût moyen de de la recherche du k-ème élément dans une zone de taille n.

$$\mathcal{C}(n) = \frac{1}{n}\sum_{k=1}^{n}\mathcal{C}(n,k).$$

→ moyenne sur tous les k

$C(n,k)$= coût moyen de de la recherche du k-ème élément dans une zone de taille n.

$$\mathcal{C}(n,k) = \frac{1}{n}\underbrace{\left(\sum_{i=1}^{k-1}\mathcal{C}(n-i,k-i) + \sum_{i=k+1}^{n}\mathcal{C}(i-1,k)\right)}_{i \text{ correspond à la pos. du pivot.}} + \underbrace{n-1}_{\text{pivotage}}$$

le k-eme élément est à droite du pivot

(on cherche le k-i ème élément à droite)

le k-eme élément est à gauche du pivot

(on cherche le k ème élément à gauche)

$$\mathcal{C}(n) = \frac{1}{n}\sum_{k=1}^{n}\mathcal{C}(n,k).$$

$$\mathcal{C}(n,k) = \underbrace{\frac{1}{n}\Big(\sum_{i=1}^{k-1}\mathcal{C}(n-i,k-i) + \sum_{i=k+1}^{n}\mathcal{C}(i-1,k)\Big)}_{i \text{ correspond à la pos. du pivot.}} + \underbrace{n-1}_{\text{pivotage}}$$

d'où:

$$\mathcal{C}(n) = \frac{1}{n^2}\Big(\underbrace{\sum_{k=1}^{n}\sum_{i=1}^{k-1}\mathcal{C}(n-i,k-i)}_{A} + \underbrace{\sum_{k=1}^{n}\sum_{i=k+1}^{n}\mathcal{C}(i-1,k)}_{B}\Big) + n-1$$

A

$$\sum_{k=1}^{n}\sum_{i=1}^{k-1}\mathcal{C}(n-i,k-i) = \mathcal{C}(n-1,1) + \mathcal{C}(n-1,2)+\mathcal{C}(n-2,1) + \mathcal{C}(n-1,3)+\mathcal{C}(n-2,2)+\mathcal{C}(n-3,1) + \cdots + \mathcal{C}(n-1,n-1)+\mathcal{C}(n-2,n-2)+\mathcal{C}(n-3,n-3) + \cdots + \mathcal{C}(2,1)+\mathcal{C}(1,1)$$

$$\to \mathcal{C}(1,1) + \big(\mathcal{C}(2,1)+\mathcal{C}(2,2)\big) + \big(\mathcal{C}(3,1)+\mathcal{C}(3,2)+\mathcal{C}(3,3)\big) + \cdots$$
$$\big(\mathcal{C}(n-1,1)+\mathcal{C}(n-1,2)+\cdots+\mathcal{C}(n-1,n-1)\big)$$
$$= \sum_{i=1}^{n-1}\sum_{k=1}^{i}\mathcal{C}(i,k)$$

$$A : \sum_{k=1}^{n}\sum_{i=1}^{k-1}\mathcal{C}(n-i,k-i)$$

B:

$$\sum_{k=1}^{n}\sum_{i=k+1}^{n}\mathcal{C}(i-1,k) = \mathcal{C}(1,1)+\mathcal{C}(2,1)+\cdots+\mathcal{C}(n-1,1) + \quad k=1$$
$$\mathcal{C}(2,2)+\mathcal{C}(3,2)\cdots\mathcal{C}(n-1,2)+ \quad k=2$$
$$\mathcal{C}(3,3)+\mathcal{C}(4,3)\cdots\mathcal{C}(n-1,3)+ \quad k=3$$
$$\vdots$$
$$\mathcal{C}(n-2,n-2)+\mathcal{C}(n-1,n-2) \quad k=n-2$$
$$+\mathcal{C}(n-1,n-1) \quad k=n-1$$

$$\sum_{i=1}^{n-1}\sum_{k=1}^{i}\mathcal{C}(i,k) = \mathcal{C}(1,1)+$$
$$\mathcal{C}(2,1)+\mathcal{C}(2,2)+$$
$$\mathcal{C}(3,1)+\mathcal{C}(3,2)+\mathcal{C}(3,3)+$$
$$\vdots$$
$$\mathcal{C}(n-1,1)+\mathcal{C}(n-1,2)+\cdots+\mathcal{C}(n-1,n-1)$$

$$B : \sum_{k=1}^{n}\sum_{i=k+1}^{n}\mathcal{C}(i-1,k)$$

$$A : \sum_{k=1}^{n}\sum_{i=1}^{k-1}\mathcal{C}(n-i,k-i) = \sum_{i=1}^{n-1}\sum_{k=1}^{i}\mathcal{C}(i,k).$$

$$B : \sum_{k=1}^{n}\sum_{i=k+1}^{n}\mathcal{C}(i-1,k) = \sum_{k=1}^{n}\sum_{i=k}^{n-1}\mathcal{C}(i,k) = \sum_{i=1}^{n-1}\sum_{k=1}^{i}\mathcal{C}(i,k).$$

d'où:

$$\mathcal{C}(n) = \frac{2}{n^2}\Big(\sum_{i=1}^{n-1}\sum_{k=1}^{i}\mathcal{C}(i,k)\Big) \quad + \quad n-1.$$

$$\mathcal{C}(i) = \frac{1}{i}\sum_{k=1}^{i}\mathcal{C}(i,k) \quad \rightarrow \quad \sum_{k=1}^{i}\mathcal{C}(i,k) = i\cdot\mathcal{C}(i)$$

$$\mathcal{C}(n) \;=\; \frac{2}{n^2}\sum_{i=1}^{n-1}\Big(i\cdot\mathcal{C}(i)\Big) \quad + \quad n-1$$

(voir le détail sur la note de synthèse)

$$\mathcal{C}(n) = O(n).$$

C'est efficace !!!

(En moyenne !!)

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 le tableau 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»

```
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]
```

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

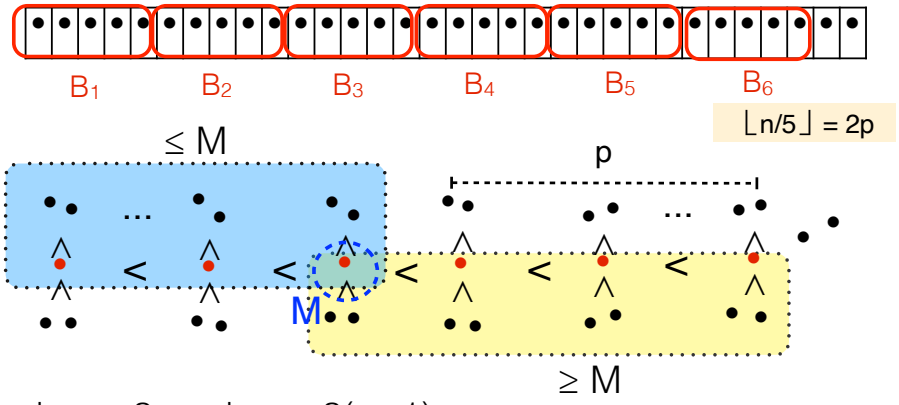
«2» appels récursifs à selectopt !

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 ?

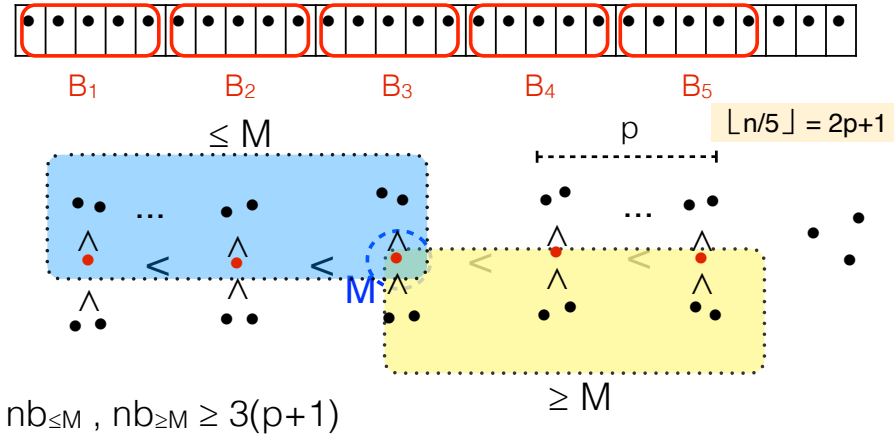


$nb_{\leq M} \geq 3p \quad nb_{\geq M} \geq 3(p+1)$

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 ?



$nb_{\leq M}$, $nb_{\geq M} \geq 3(p+1)$

Algo «select-opt» - complexité

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

$p = ½ \lfloor n/5 \rfloor = \lceil ½ \lfloor n/5 \rfloor \rceil$

$p+1 = \lceil ½(2p+1) \rceil = \lceil ½(\lfloor n/5 \rfloor +1) \rceil$

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

$p+1 = \lceil ½ (\lfloor n/5 \rfloor) \rceil$

$p+1 = ½ (\lfloor n/5 \rfloor +1) = \lceil ½ (\lfloor n/5 \rfloor +1) \rceil$

Conclusion:

$nb_{\leq M} \geq 3 \lceil ½ \lfloor n/5 \rfloor \rceil$

$nb_{\geq M} \geq 3 \lceil ½ (\lfloor n/5 \rfloor +1) \rceil \geq 3 \lceil ½ \lfloor n/5 \rfloor \rceil$

Algo «select-opt» - complexité

$nb_{\le M}$, $nb_{\ge M} \ge 3 \lceil ½ \lfloor n/5 \rfloor \rceil$

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

$nb_{\le M}$, $nb_{\ge M} \ge 3 \lceil ½ \lfloor n/5 \rfloor \rceil \ge 3/2 \cdot \lfloor n/5 \rfloor \ge (3n-12)/10$

$\ge 3n/10 - 2$

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

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

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 $t(n) = \Theta(n)$

Preuve:

supp. $t(n) \le \alpha \, . \, n$

$t(n) = \sum a_i\, t(n/b_i) + c.n \le \sum a_i \, . \, \alpha \, . \, n / b_i + c.n = (c + \alpha \sum a_i/b_i) \, . \, n$

Et $(c + \alpha \sum a_i/b_i) \le \alpha$ si $\alpha \ge \dfrac{c}{1-\sum a_i/b_i}$

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 !

(dans le pire cas !)

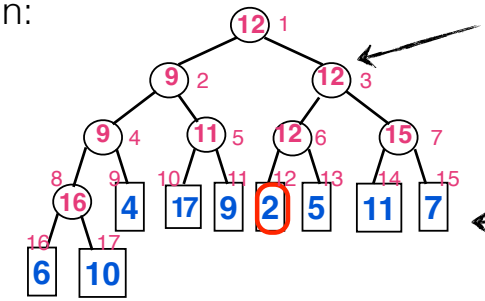
Algorithme avec des arbres

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

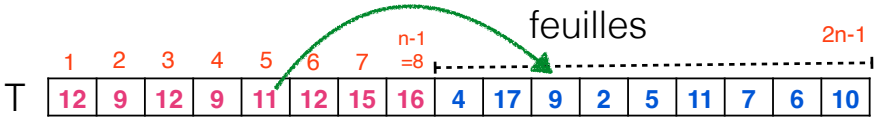


arbre binaire parfait...
9 feuilles + 8 noeuds internes numérotés de 1 à 17

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

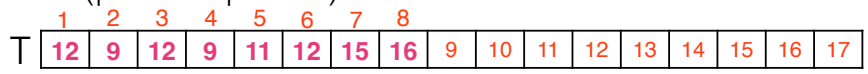
k^eme élément et arbres min

structure de données:

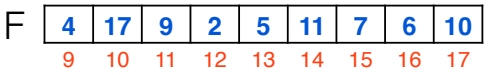


Valeur désignée par le noeud i: T[T[i]] si i<n, ou T[i] si i ≥ n.

ou... (pour simplifier !)



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



k^ème élément et arbres min

Arbre binaire parfait: hauteur ≤ $\lceil log(n) \rceil$

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).

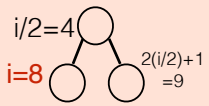
k^eme élément et arbres min

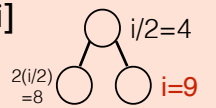
noeuds 1…n-1 : noeuds internes

```
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

Complexité:

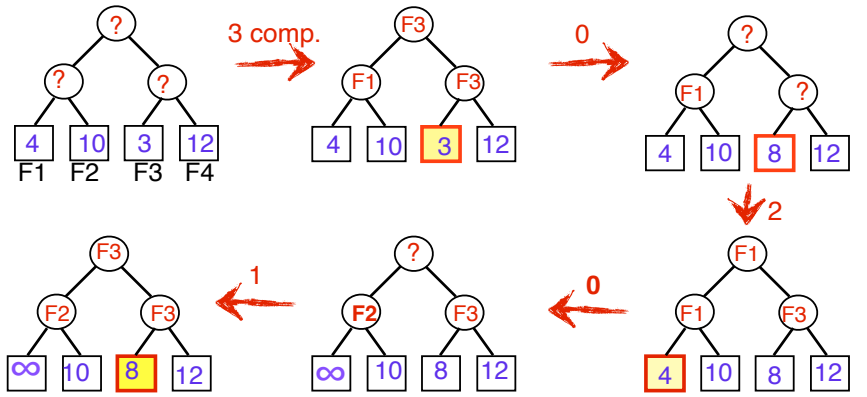
$$O\left(n-k + (k-1)\log(n-k+2)\right)$$

CalculArbre — k-1 MaJ

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 ele^ts.

Conclusion 1

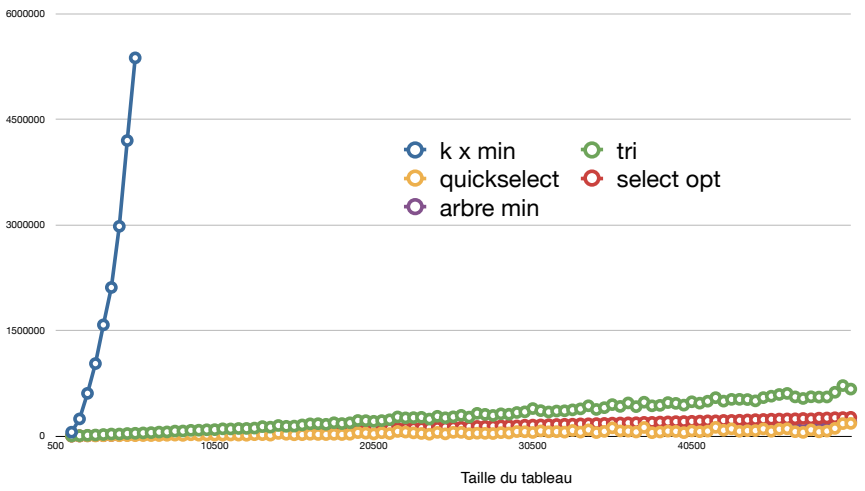
naif	naif 2	select	select opt	arbres min et tas
O(k.n) $O(n^2)$	O(n.log n)	$O(n^2)$	O(n)	O(n+k.log(n))

Test...

Nb de comparaisons

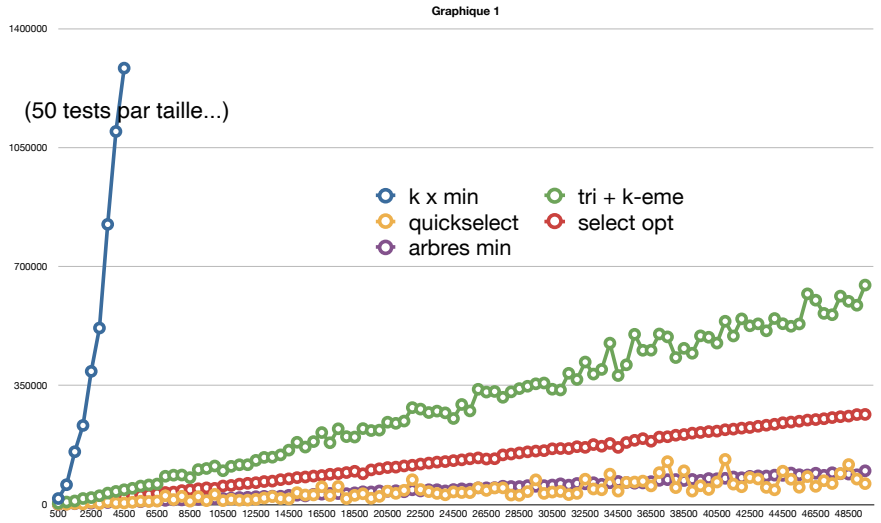
$k \in \{1,\dots,n/2\}$

(50 tests par taille...)



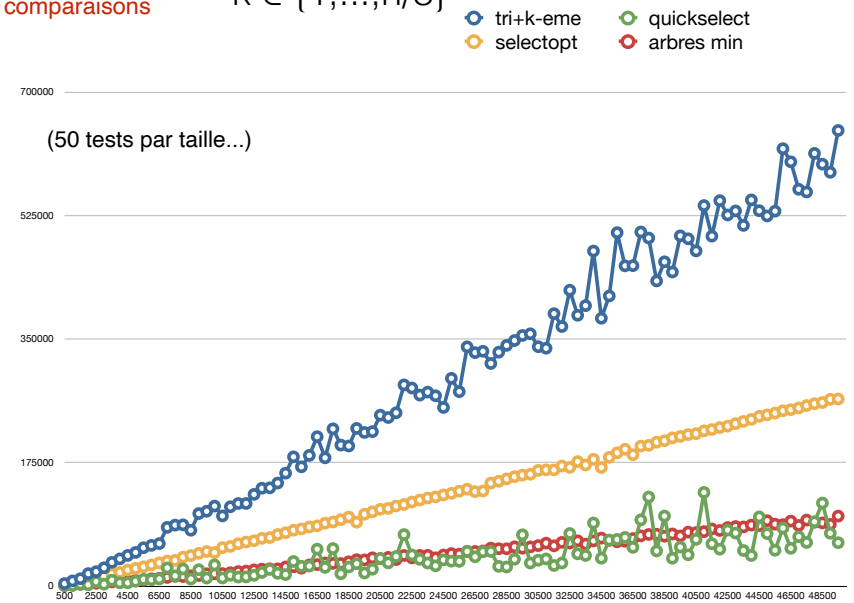
Nb de comparaisons

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



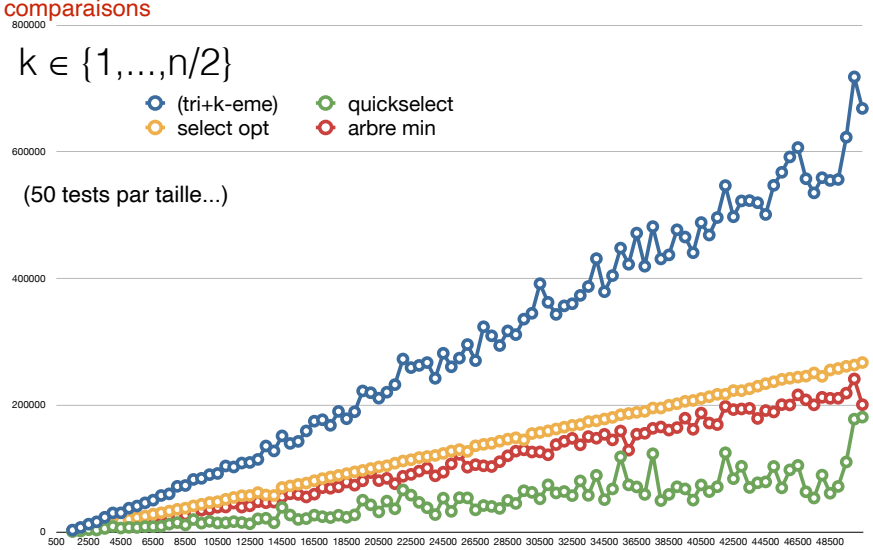
Nb de comparaisons

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

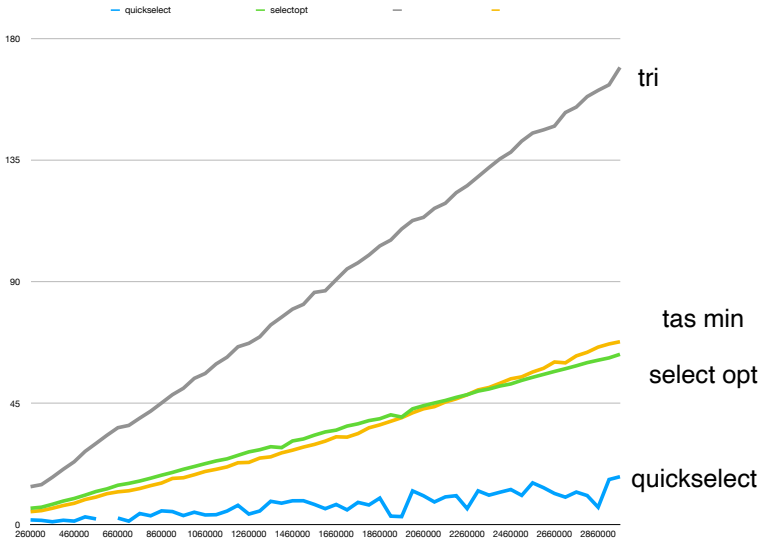


Nb de comparaisons

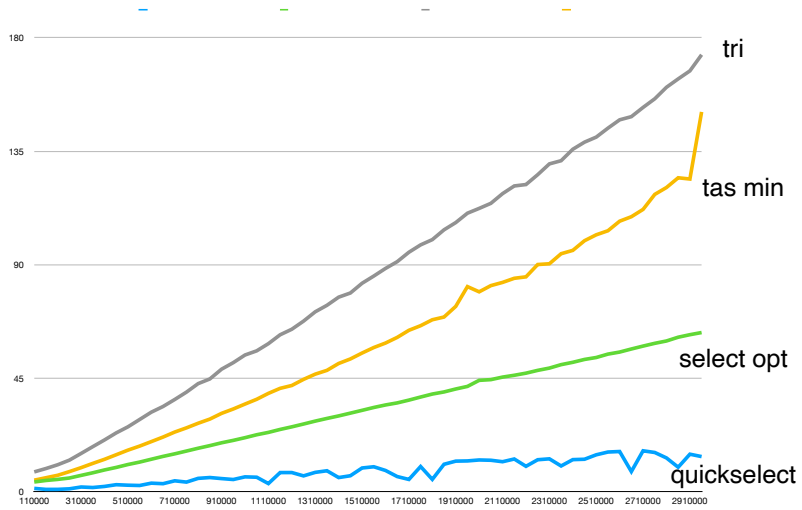
$k \in \{1,\ldots,n/2\}$



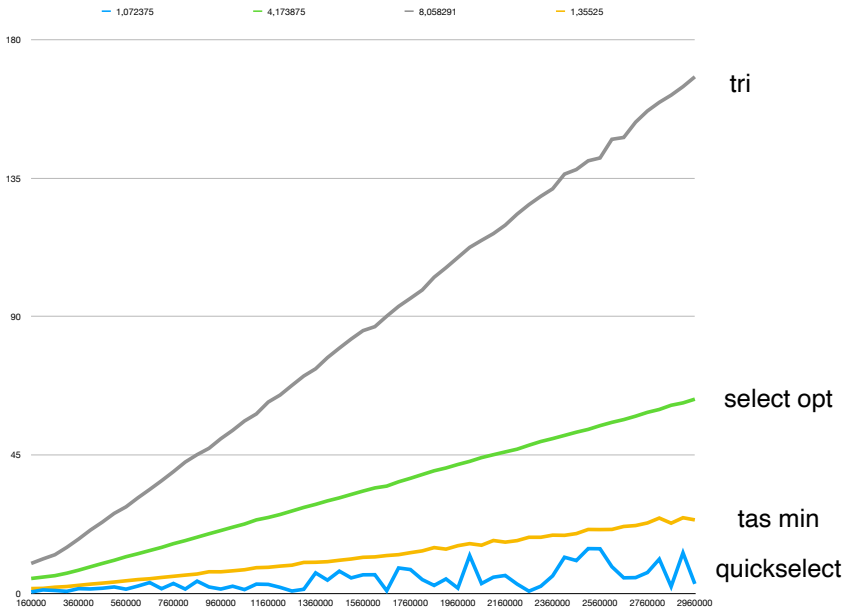
En temps d'exécution



Go $k = n/10$



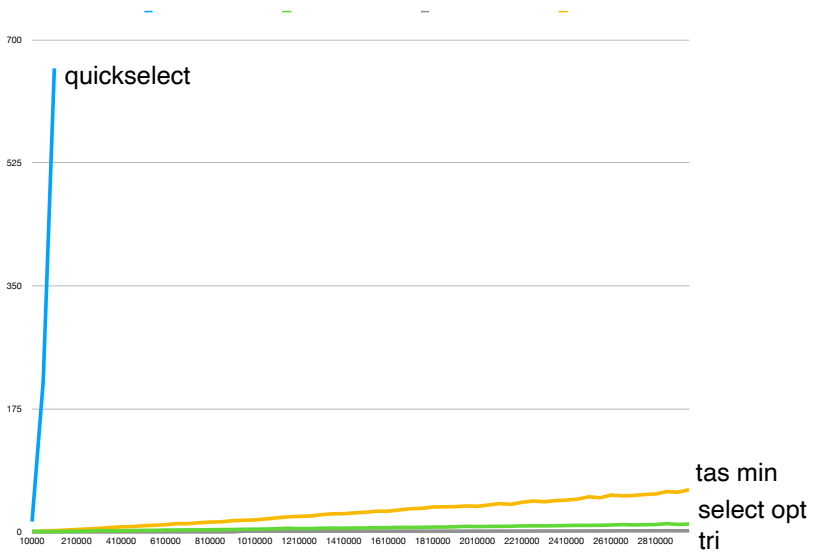
Go k = n/4



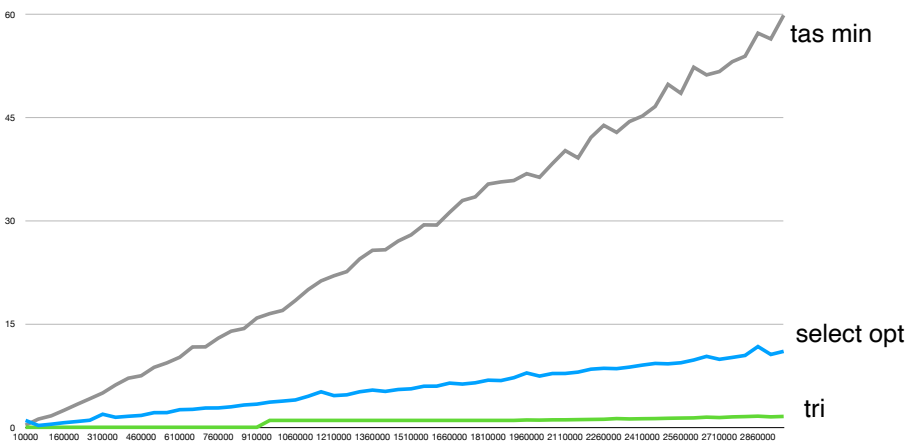
Go k = 50

Et sur un tableau déjà trié ?

(oui, ça devient un pb trivial…)



Go k = n/4


(ms)
60
45
30
15
0
10000 160000 310000 460000 610000 760000 910000 1060000 1210000 1360000 1510000 1660000 1810000 1960000 2110000 2260000 2410000 2560000 2710000 2860000
tas min
select opt
tri

Go k = n/4