UFR d'Informatique

Recherche du k^{ème} élément

F. Laroussinie

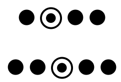
la médiane

Données:
- Un tableau T de taille $n \geq 1$
- 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 $n \geq 1$
- 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 $k \leq n/2$

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 $3^{ème}$ plus petit… C'est soit le plus petit, soit le $2^{ème}$ 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

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
4	9	1	8	-2	5	10	14	21	15	7	-4	9	11	13	55	61	-8	6	2

i=1 indmin=12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
4	9	1	8	-2	5	10	14	21	15	7	-8	9	11	13	55	61	-8	6	2

i=2 indmin=12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
4	9	1	8	-2	5	10	14	21	15	7	6	9	11	13	55	61	-8	6	2

i=3 indmin= 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

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
4	9	1	8	2	5	10	14	21	15	7	6	9	11	13	55	61	-8	6	2

i=4 indmin=3

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
4	9	∞	8	2	5	10	14	21	15	7	6	9	11	13	55	61	-8	6	2

i=5 indmin=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 $k \leq n/2$

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.

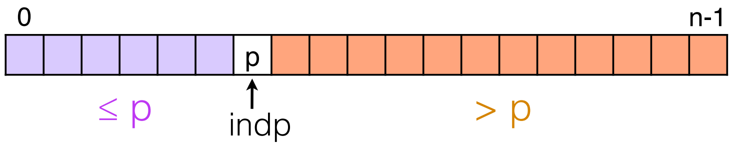
→ $O(n + k \log(n))$

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
- $k\text{-}indp\text{-}1^{eme}$ élément sur la partie droite si k>indp+1

et on s'arrête si k=indp+1 !

Exemple

T

0						6											17
7	10	4	8	3	2	12	45	20	15	17	80	71	34	89	30	23	14

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 $k\text{-}7^{ème}$ 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 ?

$\boldsymbol{C}(n,k)$= coût moyen de de la recherche du k-ème élélement dans une zone de taille n.

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

→ moyenne sur tous les k

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

(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 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]
```

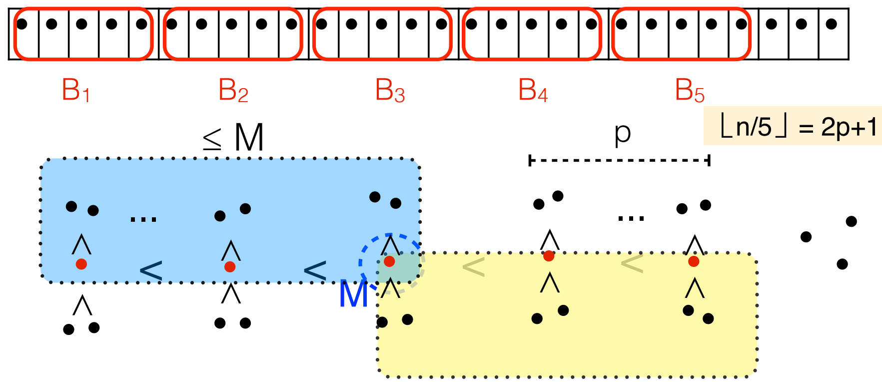
«2» appels récursifs à selectopt !

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'élets ≤M et ≥M ?

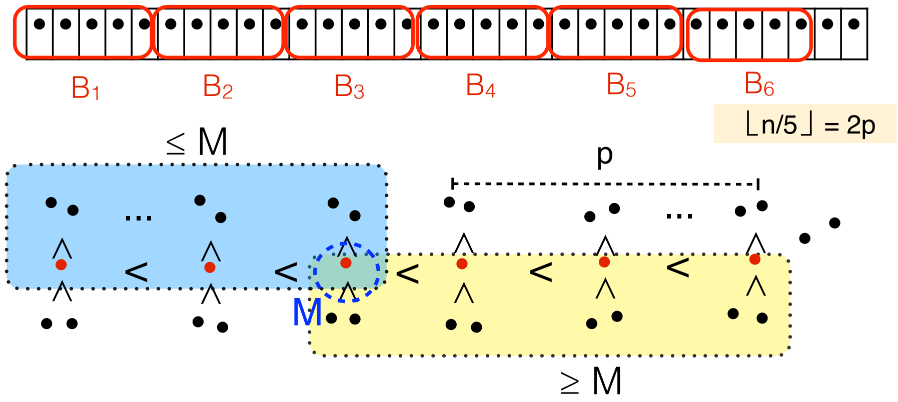


$nb_{\leq M}$, $nb_{\geq M} \geq 3(p+1)$

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'élets $\leq M$ et $\geq M$?



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

Algo «select-opt» - complexité

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

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

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

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

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

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

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

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

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

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

$\geq 3n/10 - 2$

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

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

recherche de M

suite du calcul

Algo «select-opt» - complexité

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

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. $t(n) \leq \alpha \,.\, n$

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

Et $(c + \alpha \sum a_i/b_i) \leq \alpha \quad$ si $\alpha \geq \dfrac{c}{1-\sum a_i/b_i}$

Algo «select-opt» - complexité

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

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: $\mathcal{C}(n) = \Theta(n)$

L'algorithme select-opt est en temps linaire !

(dans le pire cas !)

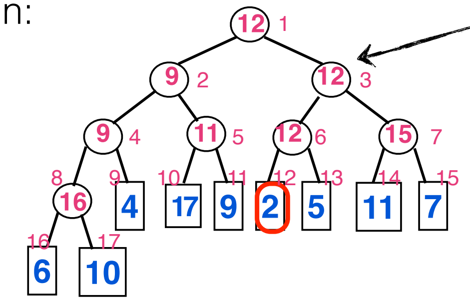
Algorithme avec des arbres

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



n-1 noeuds internes

n feuilles avec n valeurs

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:

feuilles

	1	2	3	4	5	6	7	n-1 =8									2n-1
T	12	9	12	9	11	12	15	16	4	17	9	2	5	11	7	6	10

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

ou... (pour simplifier !)

	1	2	3	4	5	6	7	8									
T	12	9	12	9	11	12	15	16	9	10	11	12	13	14	15	16	17

F	4	17	9	2	5	11	7	6	10
	9	10	11	12	13	14	15	16	17

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).

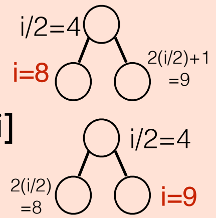
k^{eme} élément et arbres min

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:

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

CalculArbre

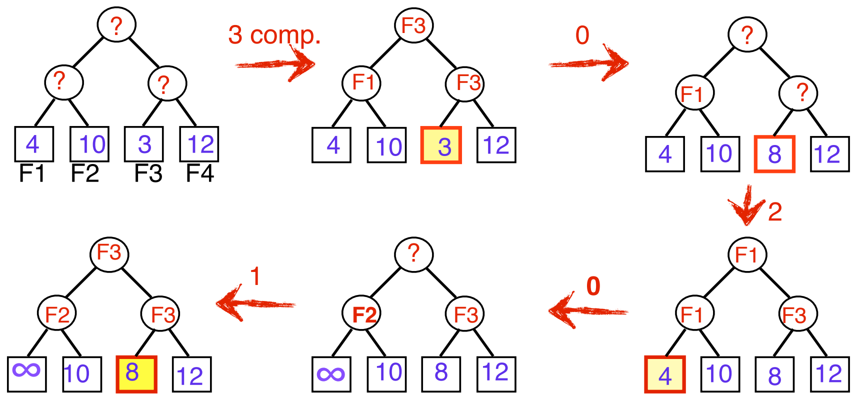
k-1 MaJ

k^{eme} élément et arbres min

Le cas des **quintuplets**. Par ex:

4	10	3	12	8

ici $n-k+2 = 4$

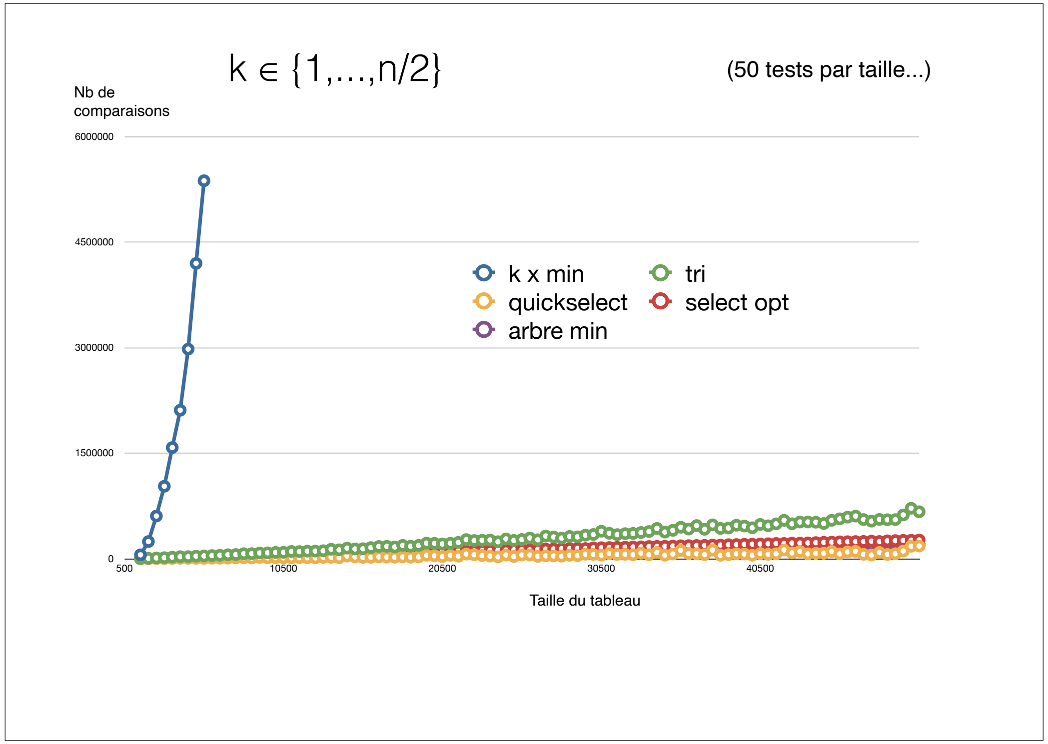


Total = 6 comparaisons suffisent pour trouver le median de 5 ele^ts^.

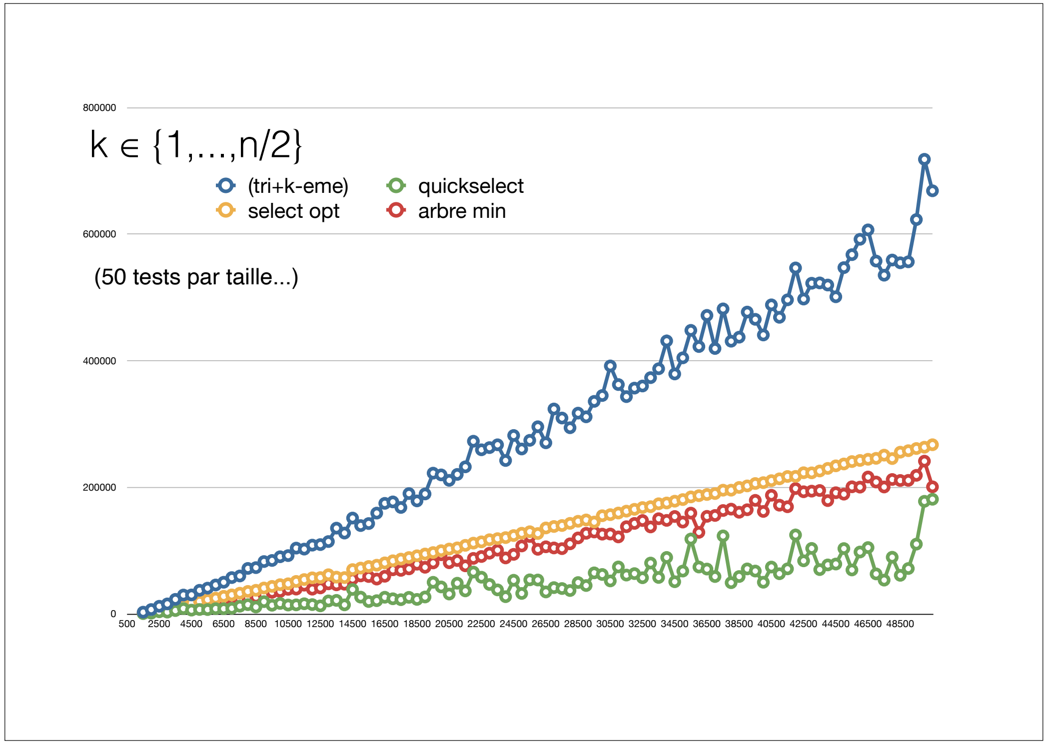
Conclusion 1

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

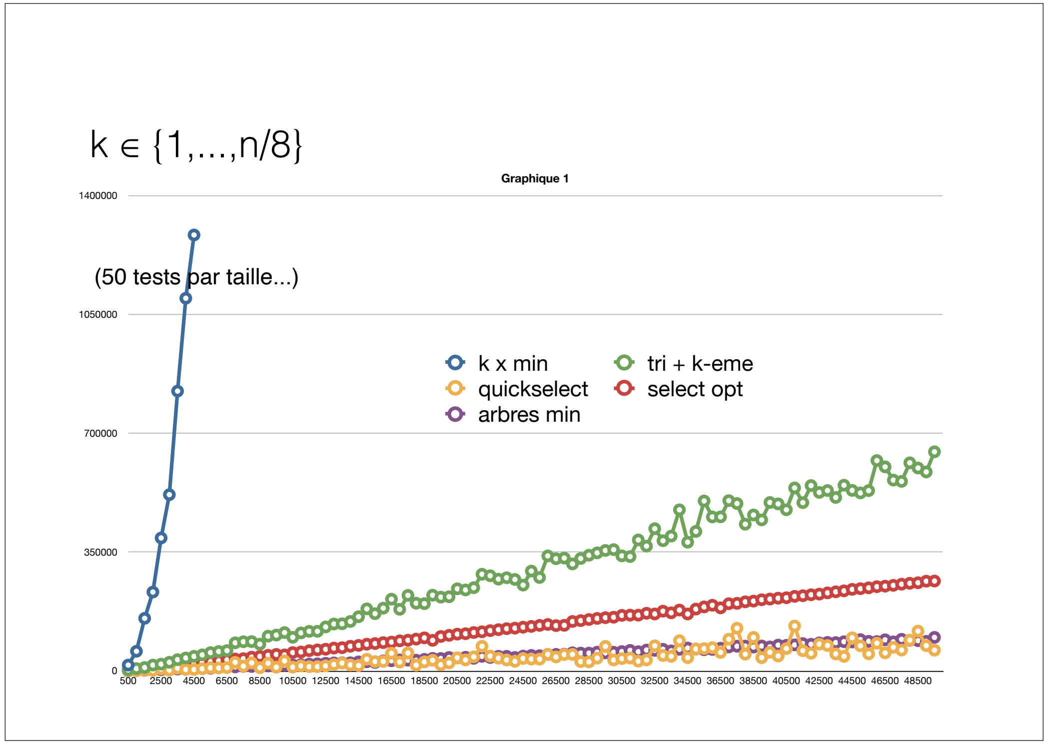
Test...



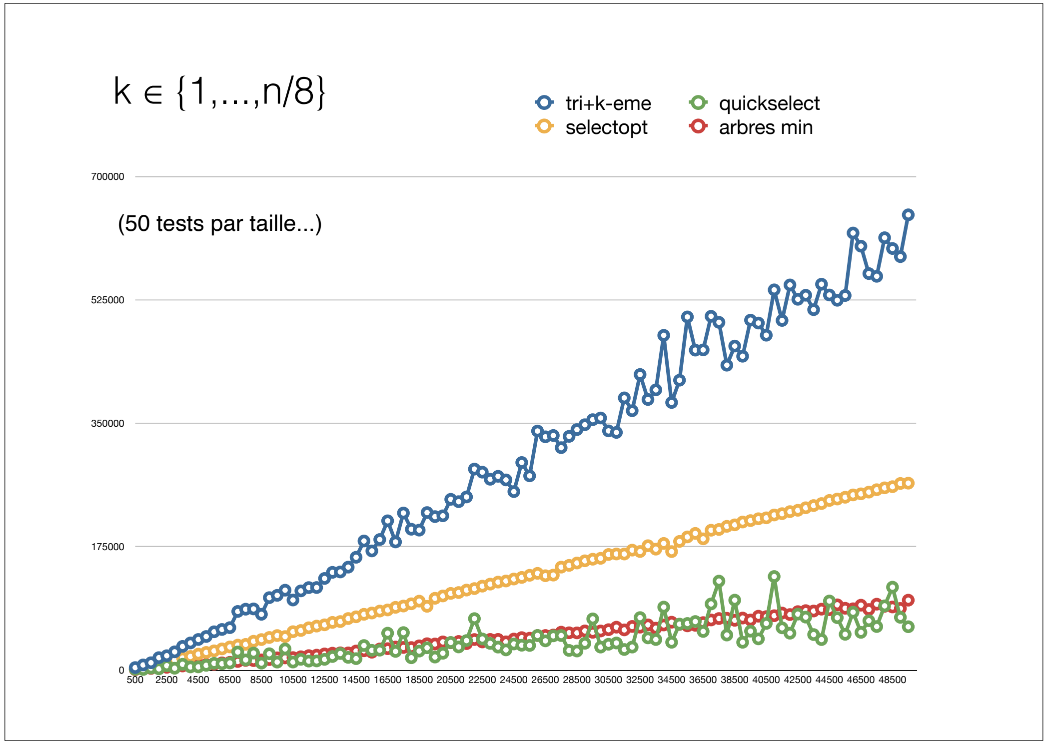
k ∈ {1,...,n/2}
(50 tests par taille...)
Nb de comparaisons
6000000
4500000
3000000
1500000
0
500
10500
20500
30500
40500
k x min
tri
quickselect
select opt
arbre min
Taille du tableau



800000
k ∈ {1,...,n/2}
(tri+k-eme)
quickselect
select opt
arbre min
600000
(50 tests par taille...)
400000
200000
0



k ∈ {1,...,n/8}
Graphique 1
1400000
(50 tests par taille...)
1050000
k x min
tri + k-eme
quickselect
select opt
arbres min
700000
350000
0



k ∈ {1,...,n/8}
tri+k-eme
quickselect
selectopt
arbres min
700000
(50 tests par taille...)
525000
350000
175000
0

N=10000

```
>>> select.test(10000,10,[True,True,True,True,True,True])
------------------
algo naif, cout moy = 5282471.7
Tps = 1.7541483

algo naif 2 (tri), cout moy = 102947.4
Tps = 0.04911099999999986

select, cout moy = 17341.5    prof moy=16.5
cout min : 9879          cout max : 32980
Tps = 0.007421899999999848

select tas, cout moy = 38666.8
Tps = 0.020800000000000197

select opt, cout total moy = 51694.8     prof moy=11.6
    select opt, cout NBC = 10042.4
    select opt, cout NBC2 = 41652.4
cout min : 50607           cout max : 52451
Tps = 0.03316210000000006

kpp, cout moyen  = 16457.5
cout min : 12808           cout max : 20400
Tps = 0.016043100000000133
(5282471.7, 1.7541483, 102947.4, 0.04911099999999986, 17341.5, 0.0074218999999998
48, 51694.8, 0.03316210000000006, 16457.5, 0.016043100000000133, 38666.8, 0.02080
0000000000197)
>>> 
```

Exemples de temps d'exec en **rose**…

N=50000

```
>>> >>> select.test(50000,10,[True,True,True,True,True,True])
------------------
algo naif, cout moy = 123237535.2
Tps = 42.122312199999996

algo naif 2 (tri), cout moy = 632916.0
Tps = 0.27930520000000525

select, cout moy = 78588.5    prof moy=19.4
cout min : 52964           cout max : 118298
Tps = 0.0342397000000048

select tas, cout moy = 200003.0
Tps = 0.11629580000000743

select opt, cout total moy = 268366.0     prof moy=14.0
    select opt, cout NBC = 50057.0
    select opt, cout NBC2 = 218309.0
cout min : 257163          cout max : 271037
Tps = 0.16345900000000527

kpp, cout moyen  = 85938.9
cout min : 50320           cout max : 122433
Tps = 0.07965430000000637
(123237535.2, 42.122312199999996, 632916.0, 0.27930520000000525, 78588.5, 0.03423
97000000048, 268366.0, 0.16345900000000527, 85938.9, 0.07965430000000637, 200003.
0, 0.11629580000000743)
>>> 
```

N=100000

```
>>> select.test(100000,10,[False,True,True,True,True,True])
------------------
algo naif 2 (tri), cout moy = 1367614.6
Tps = 0.5988719000000117

select, cout moy = 144256.4    prof moy=18.7
cout min : 94983           cout max : 237532
Tps = 0.059355999999991124

select tas, cout moy = 379848.0
Tps = 0.22008410000000253

select opt, cout total moy = 544917.0     prof moy=16.3
    select opt, cout NBC = 99957.3
    select opt, cout NBC2 = 444959.7
cout min : 537963           cout max : 547894
Tps = 0.3340401999999926

kpp, cout moyen  = 162414.6
cout min : 114312           cout max : 239107
Tps = 0.17779550000000768
(0.0, 0.0, 1367614.6, 0.5988719000000117, 144256.4, 0.059355999999991124, 544917.
0, 0.3340401999999926, 162414.6, 0.17779550000000768, 379848.0, 0.220084100000002
53)
>>> 
```