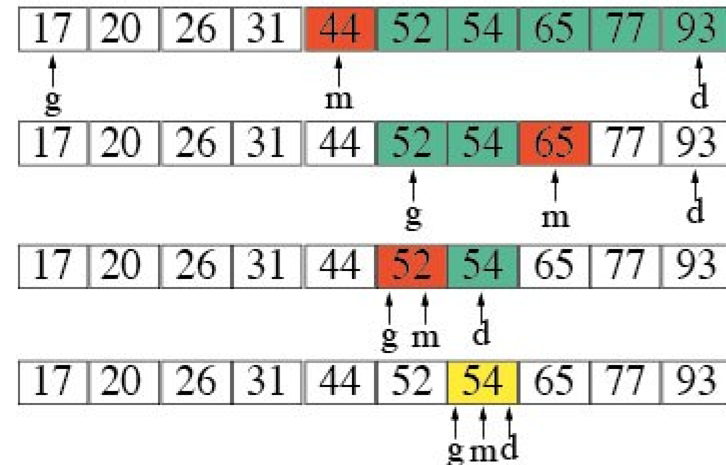
Algorithmes plus avancés (draft)

Recherche dichotomique et Tris

Recherche dichotomique (rappel)

- On suppose un tableau (integer,…) trié
- On utilise 2 indices gauche (left, l et des fois less) et droite (right, r et des fois upper, u)
- consiste à comparer l'élément cherché avec l'élément central de la liste (à l'indice m). Si l'élément cherché lui est inférieur, il doit se trouver dans la première moitié sinon dans la deuxième moitié
- Appliquer récursivement à la moitié choisie
- L'algorithme aboutira
  - soit à la position de l'élément cherché,
  - soit à aucune position (return -1)
    - auquel cas l'élément n'appartient pas



Être trié

- Il faut déjà définir formellement « être trié » :
  - Key/OpenJML ≡
    - ***\forall int** x; (**\forall int** y; 0 <= x && x < y && y < a.length; a[x] <= a[y])); ou BIEN **\forall int** x; 0 <= x && x < a.length-1 ; a[x]<=a[x+1]*
  - Krakatoa ≡

    predicate is_sorted{L}(**int**[] t) = t != **null**

    && **\forall integer** i j; 0 <= i && i <= j && j < t.length ==> t[i] <= t[j] ;

    Puis is_sorted(t) (le L pour utiliser un label ; implicite sinon)
- Puis la post-condition (voir correction) : avec « j'ai trouvé » et « je n'ai pas trouvé »

Code en Java (une version)

```java
static int search(int[] a, int v) {
    int l = 0;
    int r = a.length - 1;
    if(a.length == 0) return -1;
    if(a.length == 1) return a[0] == v ? 0 : -1;
    while (r > l + 1) {
            int mid = (l+r) / 2;
            if (a[mid]==v) return mid;
            else if (a[mid]>v) r = mid;
            else l = mid;
        }
    if (a[l] == v) return l;
    if (a[r] == v) return r;
    return -1;
```

Exo1 ; Testez l'algo sur de petits tableaux pour voir un peu les propriété (SOP les vars)

Exo2 : trouvez

a) pre/post conditions

b) variant

c) invariants

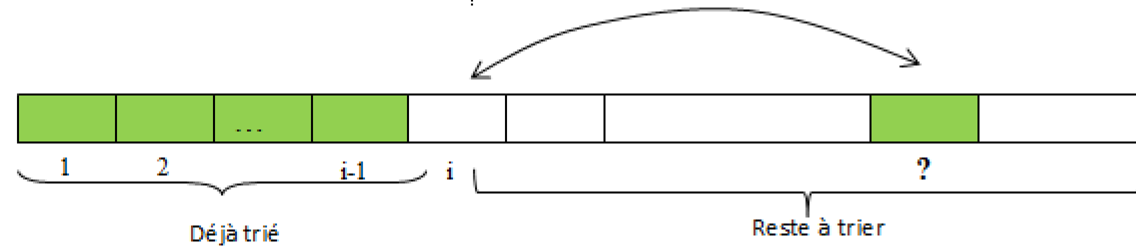
Bug Google

- https://ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html
- https://thebittheories.com/the-curious-case-of-binary-search-the-famous-bug-that-remained-undetected-for-20-years-973e89fc212
- OVERFLOW avec (l+r)/2 !
- Regardons avec Key et Krakatoa
- Solution l+(r-l)/2 !
- En Krakatoa, il y a donc bien « integer » et « int » pour parler avec entiers mathématiques et machines (toujours machine si option checkarith)

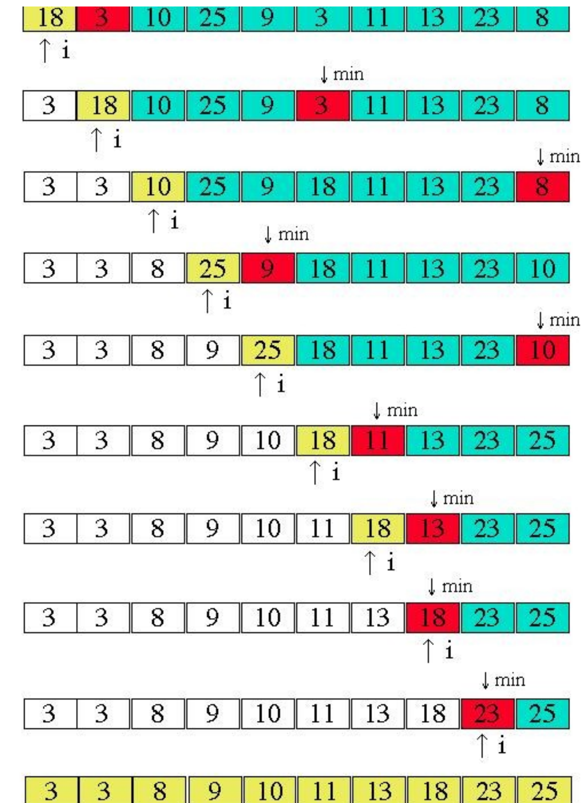
Tri par sélection

1. Chercher le plus petit élément du tableau restant

2. Échanger cet élément avec l'élément en position i



- à chaque étape, rechercher le plus petit élément non encore trié et à le placer à la suite des éléments déjà triés
- A une étape i, les i– 1 plus petits éléments sont en place, et il nous faut sélectionner le ième élément à mettre en position i ; Donc :
  - Rechercher le ième élément (le plus petit)
  - Le placer en position i !



Qu’est-ce que trier ?

- {4,2,5,7} ==sorting==> {2,4,5,7}
  - Éléments sont dans l’ordre
  - Cela suppose aussi que les éléments sont comparables (ouf, c’est le cas pour les integer)
- {4,2,5,7} ==sorting==> {2,4,5} ???
- {4,2,5,7} ==sorting==> {1,2,3,4,5,6,7} ???
  - Il ne faut pas perdre d’éléments
  - Il ne faut pas en rajouter
  - La sortie doit être une permutation de l’entrée

Permutation (difficile)

- Key (?)
  - \seq(arr) == \seq(\old(arr));
  - \dl_seqPerm(\dl_array2seq(arr), \old(\dl_array2seq(arr)));
- Krakatoa, on génère pour des prouveurs dont Coq ;
  - **inductive** Permut{L1,L2}(**int** a[], **integer** l, **integer** h) {
  - **case** Permut_refl{L}: **\forall int** a[], **integer** l h; Permut{L,L}(a, l, h) ;
  - **case** Permut_sym{L1,L2}: **\forall int** a[], **integer** l h;

    Permut{L1,L2}(a, l, h) ==> Permut{L2,L1}(a, l, h) ;
  - **case** Permut_trans{L1,L2,L3}: **\forall int** a[], **integer** l h;

    Permut{L1,L2}(a, l, h) && Permut{L2,L3}(a, l, h) ==> Permut{L1,L3}(a, l, h) ;
  - **case** Permut_swap{L1,L2}: **\forall int** a[], **integer** l h i j;

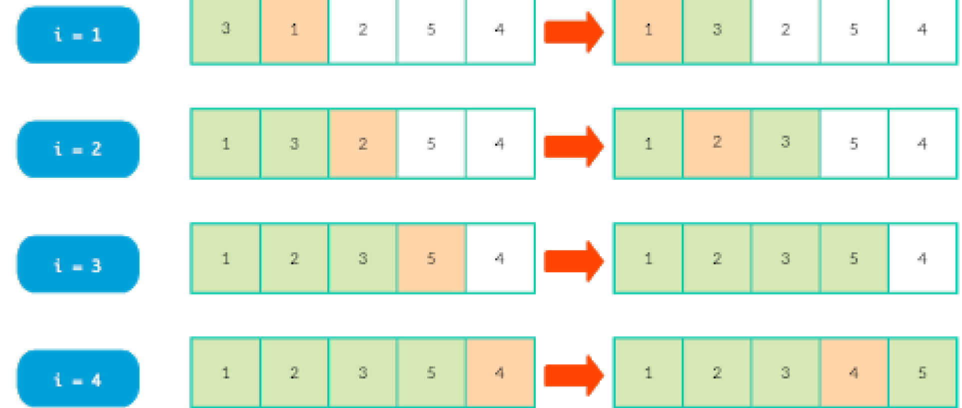
    l <= i <= h && l <= j <= h && Swap{L1,L2}(a, i, j) ==> Permut{L1,L2}(a, l, h) ;
  - **On s’en sert Permut{Pre,Here}(t,0,t.length-1) (ou avec Old)**

L’algorithme en Java

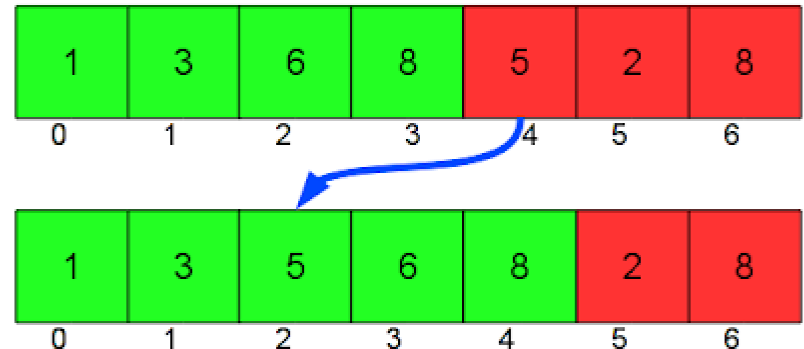
```
void selection_sort(int tab[]) {
    int mi,mv;
    for (int i=0; i<tab.length-1; i++) {
        mv = t[i]; mi = i;
        for (int j=i+1; j < tab.length; j++)
            if (tab[j] < mv) { mi = j ; mv = tab[j]; }
    swap(tab,i,mi);
}
```

Tri par insertion (rappel)

- Insèrer un élément dans un partie du tableau déjà triés
- On décale les autre éléments pour insérer cet élément



- On commence avec soit une partie de tableau vide soit avec un unique élément ; Et on itère



L’algorithme en Java

- public void insertSort(int t[]) {
- for (int i = 1; i < t.length; i++) {

```
    int key = t[i];
    int j = i ;
    while (j > 0 && t[j-1] > key) { t[j] = t[j-1]; j=j-1 } ;
    t[j] = key;
}
```

Problème

- On ne fait l’insertion qu’à la toute fin de la boucle interne ; Donc, temporairement, le tableau n’est plus une permutation du tableau initial (il manque « key » en t[j])
- 4 solutions possibles :

 1) Considérer un « tableau virtuelle » où on aurait key en j ; Utilisation d’une variable **« ghost »**

 2) Un JML qui accepte une expression du genre t[j←v]

 3) Trouver une expression JML qui décrit (2) (très très lourd)

 4) Swap tout le temps (moins efficace) avec le code suivant

```
while (j > 0 && t[j-1] > key) { swap(t,j-1,j); j = j - 1; }
```

Autres outils connus

- Dafny
  - https://www.microsoft.com/en-us/research/project/dafny-a-language-and-program-verifier-for-functional-correctness/
  - http://cse-212294.cse.chalmers.se/courses/tdv/dafny/
  - https://en.wikipedia.org/wiki/Dafny
- Spec Sharp
  - https://en.wikipedia.org/wiki/Spec_Sharp
- Frama-C (boite à outil pour le C, assertions avec ACSL)
  - https://frama-c.com/
  - https://github.com/fraunhoferfokus/acsl-by-example