Recherche sequentielle dans les tableaux

```
int TrouverEntierDansTableau (int clef, int tab[], int n)
{
   int i;

   for (i = 0; i < n; i++)
      if (clef == tab[i]) return (i);
   return (-1);
}
```

Recherche dichotomique (binaire) dans les tableaux.

```
int TrouverDansTableauTrie (int clef, int tab[], int n)
{
   int gauche, droite, milieu, val;

   gauche = 0;
   droite = n - 1;

   while (gauche <= droite) {
      milieu = (gauche + droite) / 2;
      val = tab[milieu];
      if (clef == val) return milieu;
      if (clef < val) {
         droite = milieu - 1;
      } else {
         gauche = milieu + 1;
      }
   }
   return (-1);
}
```


Algorithmes de tri.

Algorithmes de tri simples.

Tri par bulles (bubble sort).

```
bubble (int a[ ], int N)
{
   int i, j, temp;

   for ( i=N; i>=1; i--)
      for (j=2; j<=i; j++)
         if (a[j-1] > a[j]) {
            temp = a[j-1];
            a[j-1] = a[j];
            a[j] = temp;
         }
}
```


Tri par sélection.

1. i =1
2. répéter jusqu’à i==N
   2.1. sélectionner le minimum de a[i] , . . . , a[N],
   2.2. l’échanger avec a[i],
   2.3. i++

```
selection (int a[ ], int N)
{
   int i, j, min, temp;
   for (i=1; i<N; i++) {
      min = i;
      for (j=i+1; j<=N; j++)
         if (a[j] < a[min])
            min = j;
      temp = a[min];
      a[min] = a[i];
      a[i] = temp;
   }
}
```

Analyse des algorithmes de tri simples

Algorithmes de tri avancés.

Tri rapide.

1. Choisir un élément pivot.
2. Partitionner le tableau en deux parties:
      partie gauche: a[1] , . . . , a[pivot-1]
      partie droite: a[pivot+1] , . . . , a[N]
   de sorte que tout élément de la partie gauche est inférieur ou égal que pivot, et tout élément de la partie droite est supérieur ou égal que pivot.
3. Faites récursivement deux tris indépendants: un tri pour la partie gauche et un autre tri pour la partie droite.

```
tri_rapide (int a[ ], int gauche, int droite)
{
   int pivot;
   if (droite>gauche) {
      pivot = partitionner (gauche, droite);
      tri_rapide (a, gauche, pivot-1);
      tri_rapide (a, pivot+1, droite);
   }
}
```

Procédure de partition:

1. l’élément a[i] est dans sa place finale dans le tableau pour une valeur de i,
2. tous les éléments dans les positions a[1] , . . . , a[i-1] sont inférieurs ou égaux à a[i],
3. tous les éléments dans les positions a[i+1] , . . . , a[r] sont supérieurs ou égaux à a[i],

```
echanger (int a[ ], int i, int j)
{
   int temp;

   temp = a[i];
   a[i] = a[j];
   a[j] = temp;
}


tri_rapide (int a[ ], int gauche, int droite)
{
   int val, i, j;

   if (droite>gauche) {
      val = a[droite]; i = gauche-1; j = droite;
      for ( ; ; ) {

         while (a[++i] < val) ;
         while (a[--j] > val) ;
         if (i>=j) break;
         echanger ( a, i, j)
      }
      echanger ( a, i, droite)

      tri_rapide (a, gauche, i-1);
      tri_rapide (a, i+1, droite);
   }
}
```


INF4393 - Jurek Czyzowicz

Tri par fusion.

Fusion de deux tableaux ordonnés.

1. trier la première moitié du tableau
2. trier la deuxième moitié du tableau
3. fusionner les deux parties déjà triées.

```
FUSION (A, p, q, r)
   n1 ← q - p + 1
   n2 ← r - q
   créer tableaux L[1 . . n1+1] et R[1 . . n2+1]
   pour i ← 1 à n1
         faire L[i] ← A[p+i-1]
   pour j ← 1 à n2
         faire R[j] ← A[q+j]
   L[n1+1] ← ∞
   L[n2+1] ← ∞
   i ← 1
   j ← 1
   pour k ← p à r
         faire si L[i] <= R[j]
                  alors   A[k] ← L[i]
                          i ← i+1
               sinon    A[k] ← R[j]
                          j ← j+1
```


INF4393 - Jurek Czyzowicz

```
TRI-FUSION(A, gauche, droite)

   si (droite>gauche)
      alors   milieu = (droite+gauche)/2
              TRI-FUSION (a, gauche, milieu)
              TRI-FUSION (a, milieu+1, droite)
              FUSION (a, gauche, milieu, droite)
```

Analyse du tri par fusion

Types abstraits de données - TDA (ang. ADT).

Un *type abstrait de données* est un ensemble de données avec ses opérateurs associés (parfois appelés opérateurs primitifs).

Exemple 1:

Les nombres entiers avec l'addition, la soustraction, la multiplication et la division. Le type **int**, prédéfini en C, est un modèle logiciel du type abstrait de données "nombre entier".

INF4393 - Jurek Czyzowicz

Exemple 2: Les nombres rationnels (ADA).

```
package NombreRationnel is
      type Rationnel is private;
      function '= ' (X, Y: Rationnel) return Boolean;
      function '+ ' (X, Y: Rationnel) return Rationnel;
      function '* ' (X, Y: Rationnel) return Rationnel;
      function Creer (X, Y: Integer) return Rationnel;
end;
```

L usager peut seulement deviner la sémantique des opérateurs!

Exemple 3: Type abstrait de données *complex*.

$z = \alpha + i\beta$

α et β sont des nombres réels,
α représente la partie réelle,
β représente la partie imaginaire.

Partie définition.

```
void load_complex
                (complex *p_complex, double real, double imaginary);
void retrieve_complex (complex *p_complex,
                double *p_real, double *p_imaginary);
void add_complex (complex *p_sum,
                complex *p_complex1, complex *p_complex2);
void multiply_complex (complex *p_product,
                complex *p_complex1, complex *p_complex2);

typedef struct {
    double real;
    double imaginary;
} complex;
```

Partie implémentation.

$$i = \sqrt{-1}$$


```
void load_complex
                (complex *p_complex, double real, double imaginary);
{
    p_complex->real = real;
    p_complex->imaginary = imaginary;
}

void retrieve_complex (complex *p_complex,
                double *p_real, double *p_imaginary);
{
    *p_real = p_complex->real;
    *p_imaginary = p_complex->imaginary;
}

void add_complex (complex *p_sum, complex *p_complex1,
            complex *p_complex2);
{
  /*
   * p_sum = p_complex1 + p_complex2;
   *       = (a + bi) + (c + di)
   *       = (a + c) + (b + d)i
   */
  p_sum->real = p_complex1->real + p_complex2->real;
  p_sum->imaginary =
      p_complex1->imaginary + p_complex2->imaginary;
}
```



```
void multiply_complex (complex *p_product,
                complex *p_complex1, complex *p_complex2);
{
  /*
   * p_product = p_complex1 * p_complex2;
   *           = (a + bi) * (c + di)
   *           = (ac - bd) + (ad + bc)i
   */
  p_product->real = p_complex1->real *p_complex2->real
      - p_complex1->imaginary * p_complex2->imaginary;
  p_product->imaginary =
      p_complex1->real * p_complex2->imaginary
      + p_complex1->imaginary * p_complex2->real;
}
```

Partie définition et partie implémentation d'un type abstrait.

Portabilité d'un type abstrait.

Une structure de données est une méthode particulière pour représenter un type abstrait de données.

Un modèle logiciel d'un type abstrait de données consiste en un ensemble d'opérateurs pour créer, manipuler et détruire un type particulier de données objets.

L'interface fonctionnelle - prototype de fonction.