Pimp A - info121A

Programmation IMPérative Avancée

Frédéric Vernier

Programme

- *Tri*
- *Récursivité*
- *Dichotomie*
- *Arbre*
- *Graphe*
- *Examen blanc*

Rappels 1/3

- *Mathématiques / Formalismes*
- *Informatique*
  - ***Infor**mation*
  - *Traitement auto**matique***
- *Electronique / Matériel*

niveaux d'abstraction ↑

Rappels 2/3

- *Langage de programmation*
  - *Ecrire & Lire un programme (dev)*
  - *Compiler & Traduire (dev+compilo)*
  - *Exécuter & Utiliser (dev+U)*

Rappels 3/3

- *Programme Informatique*
  - *Très pointilleux , ;*
  - *Déroulement temporel*
  - *Déterministe = PAS DE HASARD*
  - *Peu de mots-clé, beaucoup de variables*
  - *Structuré (fonctions, nommage, objets, ...)*

Structures de base

- *Boucle (for, while, do-while)*
- *If (then else)*
- *Affectation (=)*
- *Sequence (;)*
- *Fonction (def + appel)*
- *Calcul "parenthésé" Numérique Boolléen*
  - *(x+1)*cos(y)*

... et la mémoire ?

Variables

- *Un peu*
  - *typé (int, float char, etc.)*
  - *nommé (x, i, j, max, equipeGagnante)*
- *Beaucoup*
  - *tableaux (typé+nommé)*
  - *indice entiers*

Tableau

- *Taille incroyable mais commence en 0 !*
  - *1 Milliard d'éléments : T[0]->T[999999999]*
- *3 informations*
  - *nom du tableau*
  - *valeurs du tableau*
  - *indices / positions dans le tableau*
    - *IMPLICITES*

Exemple

- *classLigue1_2010*
  - *tableaux de 20 strings*
  - *classement = indice+1*

Bravo Lille !

0	Lille
1	Marseille
2	Lyon
3	Paris
4	Sochaux
5	Rennes
6	Bordeaux
7	Toulouse
8	Auxerre
9	St Etienne
10	Lorient
11	Valenciennes
12	Nancy
13	Montpellier
14	Caen
15	Brest
16	Nice
17	Monaco
18	Lens
19	Arles-Avignon

Qui est le champion ?

```
#include <iostream>
using namespace std ;

string classLigue1_2010[20]={"Lille",   "Marseille",   "Lyon",     "Paris",
                             "Sochaux", "Rennes",      "Bordeaux", "Toulouse",
                             "Auxerre", "St Etienne",  "Lorient",  "Valenciennes",
                             "Nancy",   "Montpellier", "Caen",     "Brest",
                             "Nice",    "Monaco",      "Lens",     "Arles-Avignon"};

int main (int argc, char * const argv[]) {
    cout << "Bravo " << classLigue1_2010[0] << " !"<< endl;
}
```

IL NE FAUT **PAS** DE BOUCLE !

Quel est le classement de Lorient ?

```
#include <iostream>
using namespace std ;

string classLigue1_2010[20]={"Lille",   "Marseille",   "Lyon",     "Paris",
                             "Sochaux", "Rennes",      "Bordeaux", "Toulouse",
                             "Auxerre", "St Etienne",  "Lorient",  "Valenciennes",
                             "Nancy",   "Montpellier", "Caen",     "Brest",
                             "Nice",    "Monaco",      "Lens",     "Arles-Avignon"};

int main (int argc, char * const argv[]) {
    for (int i=0; i<20; i++)
       if (classLigue1_2010[i]=="Lorient")
         cout << classLigue1_2010[i] << " est " << (i+1)<< "eme ";
}
```

Il faut une boucle

... ET UN IF

2 tableaux valent mieux qu'un

0	Lille
1	Marseille
2	Lyon
3	Paris
4	Sochaux
5	Rennes
6	Bordeaux
7	Toulouse
8	Auxerre
9	St Etienne
10	Lorient
11	Valenciennes
12	Nancy
13	Montpellier
14	Caen
15	Brest
16	Nice
17	Monaco
18	Lens
19	Arles-Avignon

classLigue1_2010

- *Y a t-il redondance d'information ?*
- *Peut-on utiliser un seul tableau à trou ?*

0	76
1	68
2	64
3	60
4	58
5	56
6	51
7	50
8	49
9	49
10	49
11	48
12	48
13	47
14	46
15	46
16	46
17	44
18	35
19	20

pointsLigue1_2010

Questions

- *Y a t-il une équipe avec 48 points ?*
- *Combien y a t-il d'équipes avec 48 points ?*
- *Quelles sont les équipes de la 2ème moitié ?*
- ***Le tableau de points est-il trié?...***
- *Le 17ème a t-il le même nombre de points que le 16ème (premier non-relégable)*
- *Combien de point a le dernier ?*

For	*If*

Le tableau de points est-il trié ?

LA BOUCLE VA DE 0 A 18
1 IF DEDANS, 1 APRES

```
#include <iostream>
using namespace std ;
string classLigue1_2010[20]={"Lille",    "Marseille",   "Lyon",      "Paris",
                             "Sochaux",  "Rennes",      "Bordeaux",  "Toulouse",
                             "Auxerre",  "St Etienne",  "Lorient",   "Valenciennes",
                             "Nancy",    "Montpellier", "Caen",      "Brest",
                             "Nice",     "Monaco",      "Lens",      "Arles-Avignon"};
int pointsLigue1_2010[20]={76, 68, 64, 60,
                           58, 56, 51, 50,
                           49, 49, 49, 48,
                           48, 47, 46, 46,
                           46, 44, 35, 20};
int main (int argc, char * const argv[]) {
    bool res = true;
    for (int i=0; i<19; i++)
      if (pointsLigue1_2010[i]<pointsLigue1_2010[i+1]) res = false;

    if (res)
      cout << "Le tableau de points est trie";
    else
      cout << "Le tableau de points n'est pas trie";
```


3 tableaux

	classementLigue1_2010
0	*Lille*
1	*Marseille*
2	*Lyon*
3	*Paris*
4	*Sochaux*
5	*Rennes*
6	*Bordeaux*
7	*Toulouse*
8	*Auxerre*
9	*St Etienne*
10	*Lorient*
11	*Valenciennes*
12	*Nancy*
13	*Montpellier*
14	*Caen*
15	*Brest*
16	*Nice*
17	*Monaco*
18	*Lens*
19	*Arles-Avignon*

	pointsLigue1_2010
0	*76*
1	*68*
2	*64*
3	*60*
4	*58*
5	*56*
6	*51*
7	*50*
8	*49*
9	*49*
10	*49*
11	*48*
12	*48*
13	*47*
14	*46*
15	*46*
16	*46*
17	*44*
18	*35*
19	*20*

	diffButsLigue1_2010
0	*32*
1	*23*
2	*21*
3	*15*
4	*17*
5	*3*
6	*1*
7	*2*
8	*4*
9	*-1*
10	*-2*
11	*4*
12	*-5*
13	*-11*
14	*-5*
15	*-7*
16	*-15*
17	*-4*
18	*-23*
19	*-49*

Rappel : Accumulateur

- *La somme des différences de buts vaut-elle 0 ?*
  - *On déclare et initialise l'accumulateur*
  - *On écrit une boucle*
  - *On fait évoluer l'accumulateur dans la boucle*
  - *On utilise la valeur de l'acc. à la sortie*

Vous avez 3 minutes !

Correction

```
int acc = 0;

for (int i=0; i<20; i++)
   acc = acc+diffButsLigue1_2010[i];

cout <<"La somme des differences de buts vaut " <<acc<<endl;
```


Questions

- *Comment trier les équipes en fonction de leurs différences de buts ?*
  - *Quelles sont les 10 meilleures équipes*
  - *Combien de places gagnerait Lorient ?*
  - *etc.*

2 approches

- *Approche interne* — *+ elem. déjà-triés*, *+ listes presque triées*, *- perm multiples*
  - *On permute les équipes au mauvais endroits*
- *Approche externe* — *+ perm. optimales*, *- rech deja triée*, *- espace mémoire*, *+ fichiers*
  - *On créé un(des) tableau(x)*
  - *On duplique/remplit avec les valeurs qu'on trouve*

Permutation

- *Il faut permuter tous les tableaux pour rester cohérents !*

Var. intermédiaire 3x

```
void permute (int i, int j) {

  string temp0 = classLigue1_2010[i];
  int temp1    = pointsLigue1_2010[i];
  int temp2    = diffButsLigue1_2010[i];

  classLigue1_2010[i]    = classLigue1_2010[j];
  pointsLigue1_2010[i]   = pointsLigue1_2010[j];
  diffButsLigue1_2010[i] = diffButsLigue1_2010[j];

  classLigue1_2010[j]    = temp0;
  pointsLigue1_2010[j]   = temp1;
  diffButsLigue1_2010[j] = temp2;
}
```

Tri naïf

- *On suppose que la liste est triée de **0 à k-1***
- *On considère l'élément k*
- *On regarde si on trouve mieux de k+1 à max*
- *Si il y a mieux, on permute k et ce "meilleur"*

➡ *On a une liste triée de **0 à k***

Implémentation 1/3

Et on teste !

```
void triDiffButs() {


  int k=3;
  int meilleur = k;
  for (int i=k+1; i<20; i++)
    if (diffButsLigue1_2010[meilleur]<diffButsLigue1_2010[i])
      meilleur = i;
   permute(k, meilleur);
```

Et on teste !

```
}
triDiffBut();
for (int i=0; i<20; i++)
  cout << classLigue1_2010[i] <<" " << pointsLigue1_2010[i] << "  " << diffButsLigue1_2010[i]  << endl;
```


Implémentation 2/3 : Comment

- *On trie avec k=0, k=1 ou k=2*
  - *rien ne change car Les meilleurs au classement sont aussi ceux qui ont la meilleure différence de buts*
- *On trie avec k=3*
  - *PSG(+15) <=> Sochaux (+17)*
- *... mais ça ne trie pas encore TOUT le tableau*

Implémentation 3/3 : On répète !

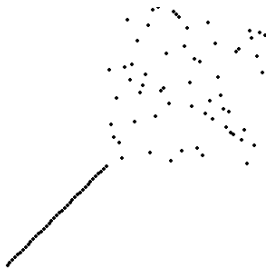
```
void triDiffButs() {
  int k;
  for (k=0; k<19; k++){
    int meilleur = k;
    for (int i=k+1; i<20; i++)
      if (diffButsLigue1_2010[meilleur]<diff
        meilleur = i;
    permute(k, meilleur);

  }
}
```

```
20 Arles-Avignon          -49
35 Lens          -23
46 Nice          -15
47 Montpellier          -11
46 Brest          -7
48 Nancy          -5
46 Caen          -5
44 Monaco          -4
49 Lorient          -2
49 St Etienne          -1
51 Bordeaux          1
50 Toulouse          2
56 Rennes          3
49 Auxerre          4
48 Valenciennes          4
60 Paris          15
58 Sochaux          17
64 Lyon          21
68 Marseille          23
76 Lille          32
```

Et on re-teste !

Bravo !

- *Ce premier tri s'appelle Tri par sélection*
- *Il est pas très rapide $O(n^2)$*
  - *mais il est rapide sur de petits tableaux*
  - *PEU d'échanges*
    - *beaucoup de recherches*
- *Autres tris (peu) efficaces*
  - *Tri à bulles et Tri insertion*



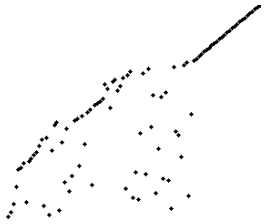
Tri Sélection

```
void triDiffButs() {
  int k;
  for (k=0; k<19; k++){
    int meilleur = k;
    for (int i=k+1; i<20; i++)
      if (diffButsLigue1_2010[meilleur]<diffButsLigue1_2010[i])
        meilleur = i;
    permute(k, meilleur);
  }
}

void triselection (int n) {
  int k=0;
  for (k=0; k<n-1; k++){
    int meilleur = k;
    for (int i=k+1; i<n; i++)
      if (diffButsLigue1_2010[meilleur]<diffButsLigue1_2010[i])
        meilleur = i;
    permute(k, meilleur);
  }
```

+ Passer le tableau à trier en paramètre

Tri à Bulle 1/2

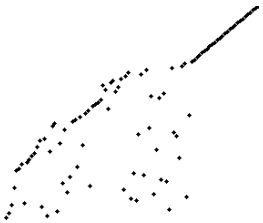
- *Fait remonter le meilleur pas par pas*
  - *MaJ du meilleur*



```
void tri_a_bulle(int n){
  bool echange;
  do {
    echange = false;
    for (int j=1; j<n; j++) {
       if(diffButsLigue1_2010[j-1]>diffButsLigue1_2010[j]){
         permute(j, j-1);
         echange = true;
       }
     }
  } while (echange);
}
```

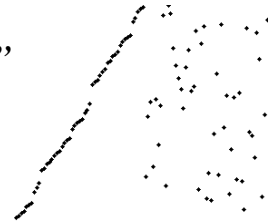

Tri à Bulle 2/2

- *Tri dans l'autre sens !*
  - *à cause du > dans le if*
- *Fait remonter palier par palier !*
  - *inefficace*
  - *tri un petit peu le debut et trie bien le dernier*



Tri Insertion

- *Comme aux cartes*
  - *prendre les cartes mélangées une à une*
  - *former une main en insérant chaque carte à sa place*
- *Tri sélection = élitiste*
- *Tri insertion = "come as you are"*
- *Déplace des éléments déjà placés*
- *1 tableau = 2 fonctions+limite*



code C++

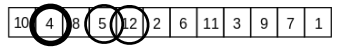
```
void tri_insertion(int n){
  for (int i=1; i<n; i++) {
    int j =  i;
    while (j > 0 && diffButsLigue1_2010[j-1]>diffButsLigue1_2010[j]) {
      permute(j, j-1);
      j--;
    }
  }
}
```

- *reste en $O(n^2)$*
  - *simple, efficace sur de petits tableaux, stable*
  - *marche très bien sur des tableaux presque triés*

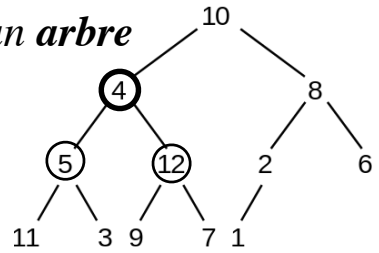
Tri par tas

- *HeapSort*
- *Plus rapide O(n.log(n))*
- *Nécessite de changer de perspective*
- *Recherche d'un **invariant***
- *Utilise une fonction intermédiaire*
  - *tamisage ou percolation*

Nouvelle perspective

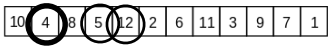


- *Un tableau est vu comme un **arbre***
  - *binaire,*
  - *quasi-équilibré,*
  - *"tassé" à gauche*
- *Parcours*
  - *...*

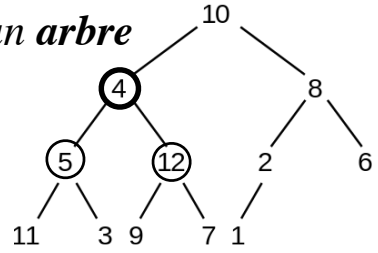


T[i] a comme fils ?
pos de 4=> pos de 5 et 12

Nouvelle perspective



- *Un tableau est vu comme un **arbre***
  - *binaire,*
  - *quasi-équilibré,*
  - *“tassé” à gauche*

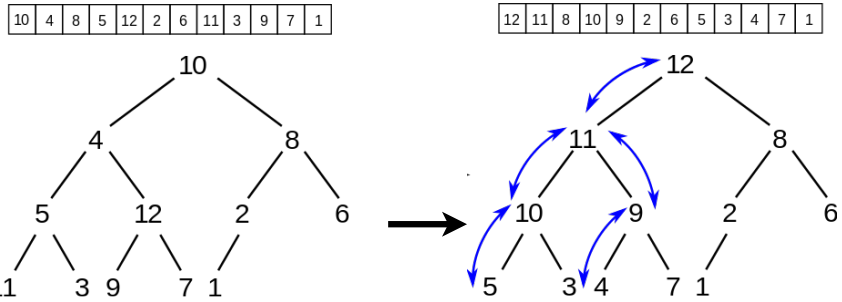


- *Parcours*
  - *DG ou GD*

T[i] a comme fils :
*T[2*i+1] et T[2*i+2]*
*... si 2*i+1<N et 2*i+2<N*

Invariant : arbre en tas

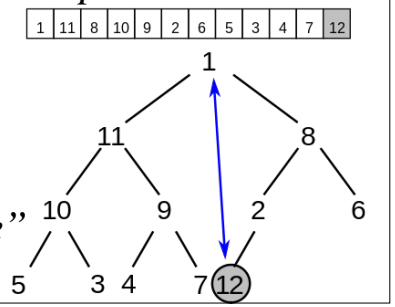
- *La valeur au dessus EST toujours plus grande que les deux valeurs en dessous*



Corollaire : La PLUS GRANDE valeur est en 0 comme une etape du tri Selection

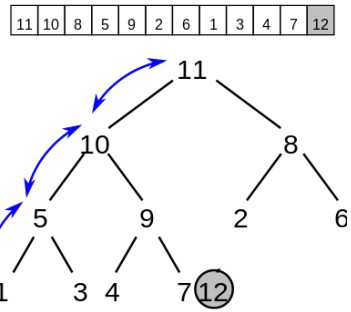
Fonctionnement général

- *HYP: Tamisé. Puisqu’on a trouvé le plus grand, on le met de coté et on trie le reste !*
- *On permute le dernier et le premier et on coupe le dernier lien de l’arbre pour le décrocher*
- *L’arbre n’est plus tout à fait tamisé*
- *Opération “quasi-tamisage”*



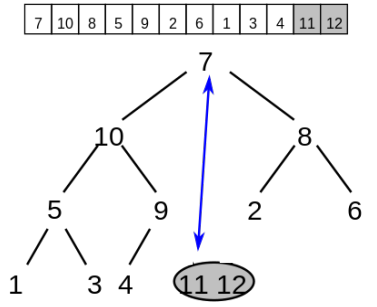
... jusqu’au bout

on quasi-tamise



qu’un seul chemin impacté !

et on permute



eh ! mais les 2 derniers sont triés !

Quasi-Tamisage

T[i] *T[2*i+1]* *T[2*i+2]*

- *Si père<max(fils droit, fils gauche) => permute*
- *Plus simple que le tamisage*
- *Hyp = arbre presque tamisé SAUF tout en haut*
  - *parcours de haut en bas (HB = GD) OK*
  - *Magique = fait redescendre la petite valeur ET remonte la nouvelle plus grande*
- *Tamisage = parcours BH de quasi-tamisage*

Tamisage (op. initiale)

T[i] *T[2*i+1]* *T[2*i+2]*

- *Si père<max(fils droit, fils gauche) => permute*
  - *un seul petit bout OK*
- *parcours de haut en bas (HB = GD)*
  - *pas tamisé en haut*
- *parcours de bas en haut (BH = DG)*
  - *pas tamisé en bas*
- *parcours DG de (parcours HB)*

Implémentation 1/2

```
void tri_par_tas(int n){
  // tamise = parcours DG de quasi-tamise
  for (int debut=(n-2)/2; debut>=0; debut--)
    quasi_tamise(debut, n-1);

  // fonctionnement general
  for (int fin=n-1; fin>0; fin--) {
    permute (0, fin);
    quasi_tamise(0, fin-1);
  }
}
```


Implémentation 2/2

```
void quasi_tamise(int debut, int fin) {
  int pere = debut;

  while (pere*2+1 <= fin) { // il y a encore des fils dans lesquels la valeur PEUT descendre
    int candidat = pere;
    int filsGauche = pere*2+1;
    int filsDroit  = pere*2+2;
    if (diffButsLigue1_2010[candidat]<diffButsLigue1_2010[filsGauche])
      candidat = filsGauche;

    if (filsDroit<=fin && diffButsLigue1_2010[candidat]<diffButsLigue1_2010[filsDroit])
      candidat = filsDroit;

    if (candidat != pere) {
      permute (pere, candidat);
      pere = candidat; // avant de boucler on pointe vers le fils permute
    } else {
      return; // le quasi-tamisage est fini. La "petite" valeur a trouve sa place definitive
    }
  }
}
```

Tri par tas

- *Plus compliqué mais plus rapide O(n.log(n))*
- *Peu rapide sur les tableaux presque triés*
  - *le tamisage casse l'aspect presque-trié*
- *Nécessite 2 fonctions pour la clarté*
  - *changement de perspective*
  - *opérations de tamisage et quasi-tamisage*

Ordres

- *Tri croissant, décroissant, autre…*
  - *point + diff. de but par exemple*
  - *que changer dans le code ?*

```
void tri_insertion(int n){
  for (int i=0; i<n; i++) {
    int j = i;
    while (j > 0 && TabPts[j-1]>TabPts[j]) {
      permute(j, j-1);
      j--;
    }
  }
}
```

Critères Multiples (points+ diff. buts)

Ordres

- *Tri croissant, décroissant, autre…*
  - *point + diff. de but par exemple*
  - *que changer dans le code ?*

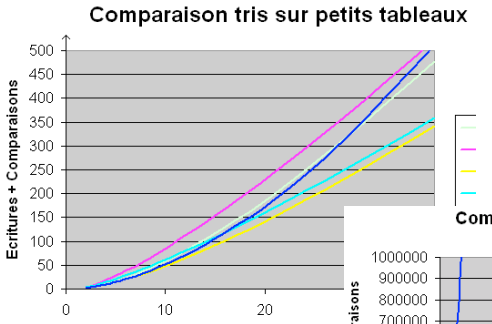
```
void tri_insertion(int n){
  for (int i=0; i<n; i++) {
    int j = i;
    while (j>0 && (TabPts[j-1]>TabPts[j] ||
              (TabPts[j-1]>TabPts[j] && diffButs[j]> diffButs[j-1])) {
      permute(j, j-1);
      j--;
    }
  }
```

C'est un OU paresseux (||)

Autres bons tris

- *Tri de Shell (shell sort) : Amélioration du tri par insertion, mais pas stable*
- *Tri fusion (merge sort) : tris deux sous-tableaux et fusionne (cartes collaboratives)=>TimSort*
- *Smoothsort tri inspiré du tri par tas (arbre non inversé), très rapide pour les ensembles déjà presque triés*
- *Quicksort … au prochain cours "récursivité"*

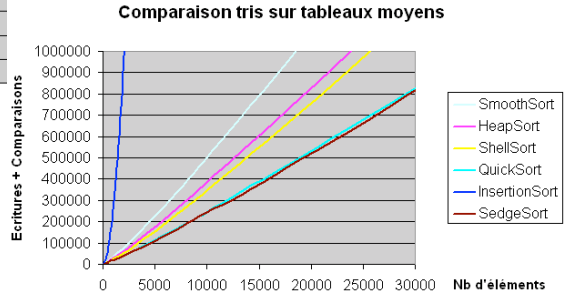
Comparaisons



Taille Données

Compl. Algo O(n.log(n))

Best, Avg, Worst



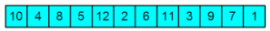
Mémoire

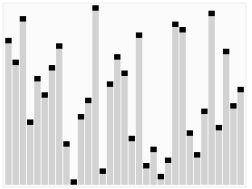
Liste vs Tableau

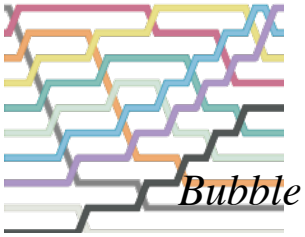
Visualisation

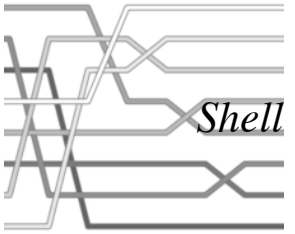
- *Animations pos/valeur*
- *Diagramme rankChart*
- *Arbre+Tableau en "BD"*

Merci Wikipedia









Tris

- *Tris Stables*
  - *gardent l'ordre initial des ex-aequos*
- *Tris sur place*
  - *ne nécessitent pas de mémoire supp.*
- *Tris internes / externes*
  - *mémoire vive / mémoire de masse*
- *Tris parallèles...*

Résumé

- *Tri par Sélection*
- *Tri par Insertion*
- *Tri à bulle*
- *Tri par tas +tri rapide à venir...*

- *Utilité*
  - *Médiane, recherche (O(Log(n)), corresp*