ISIMA F2
ZZ3

ALGORITHMES AVANCÉS ET COMPLEXITÉ

David Hill

Algorithmes avancés et complexité

Au menu :

- Contest
  - ACM
  - Google code jam
  - 101 France
  - …
- Recruitement
  - Coding interview
- Hacker Rank
  - By country
- Complexity
- Data Structures

2

DATA + ALGORITHMS
=
PROGRAMS

Nicklaus Wirth

Concours

Et entretiens d'embauche avec une sélection algorithmique

« Coding Interview »

- Depuis une dizaine d'années, des concours de programmation sont organisés dans le monde entier à tous les niveaux.
- Les problèmes sont des variantes de problèmes algorithmiques classiques

4

Quelques Exemples de concours

ACM Association for Computing Machinery

- ICPC – International Collegiate Programming Contest
- South West European Regional Contest
- Limité aux étudiants jusqu'au Master

Google Code Jam

- Ouvert à tous

Prologin, France-101…

- Lycéens, étudiants

ACM/ICPC/SWERC

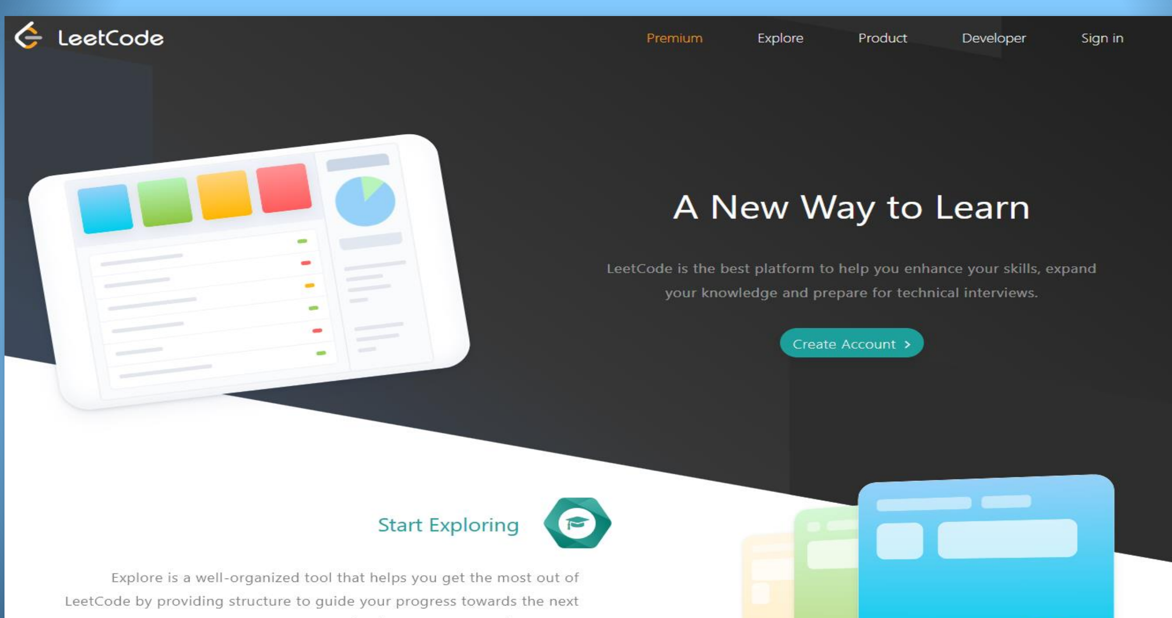
Equipes de 3 personnes

1 seul ordinateur à disposition

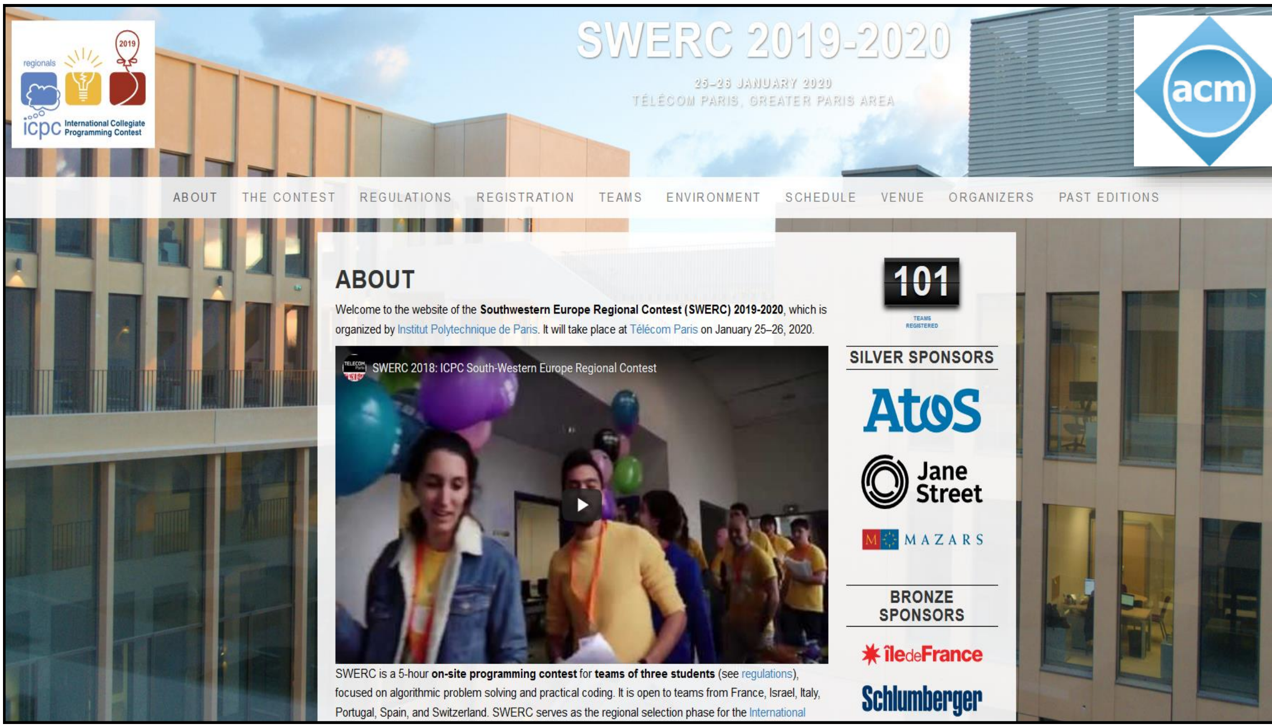
5H pour résoudre un nombre maximum de problèmes parmi les 10 posés

5

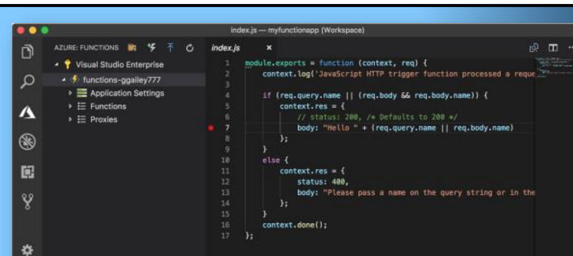
Leetcode.com



6



Pour un problème (une instance)
Une soumission de code
Testée contre un jeu de tests inconnu



Des sites d'entrainement avec des juges en ligne

livearchive.onlinejudge.org, uva.onlinejudge.org, icparchive.ecs.baylor.edu,…

Les réponses des juges

Accepted	Presentation error	Compilation error	Wrong Answer	Time Limit Exceeded	Runtime Error
Félicitation	Erreur de formatage des résultats attendus (rare)	Comparez la version du compilateur utilisée par le juge et la votre	Avez-vous testé les cas limite ? Avez-vous laissé les affichages de débogage dans le code ? Avez-vous loupé un détail de la spécification	Vous n'avez pas implémenté l'algorithme le plus efficace pour ce problème. Générez une grande instance	Une division par zéro, un dépassement de tableau, bref une « Seg fault »

8

Recrutement

Coding Interview
« Not only code… »

« Google doesn't look for deep, deep experts » (1/4)

"We would rather hire smart, curious people than people who are deep, deep experts in one area or another," he says, noting that people with strong learning ability can generally find the right answers to unfamiliar questions. "But somebody who's been doing the same thing forever will typically just replicate what they've seen before." - https://www.businessinsider.com/qualities-google-looks-for-in-job-candidates-2014-4?r=US&IR=T

Google does want people with high "cognitive ability."

"If you hire someone who is bright, and curious, and can learn, they're more likely to come up with a new solution that the world hasn't seen before," bock explained in a google+ q&a. "This looking for cognitive ability stems from wanting people who are going to reinvent the way their jobs are going to work rather than somebody who's going to come in and do what everybody else does. We recruit for aptitude, for the ability to learn new things and incorporate them."

Google seeks out people with "grit."

Bock spoke with the times about a time he was on a campus talking to a student double-majoring in computer science and math. The student was thinking about switching out of computer science — it was too difficult.

"I told that student they are much better off being a b student in computer science than an a+ student in english," he recalls. Taking computer science "signals a rigor in your thinking and a more challenging course load. That student will be one of our interns this summer."

— the ability to keep slogging through difficult work — is more important for success than raw iq.

« Google doesn't look for deep, deep experts » (1/2 oops… 2/4)

Google wants to know whether candidates can tackle difficult projects.

Google's interviews include questions about the candidate's concrete experiences, starting with queries like "give me an example of a time when you solved an analytically difficult problem."

By asking people to speak of their own experiences, bock says, you get two kinds of information: "you get to see how they actually interacted in a real-world situation, and the valuable 'meta' information you get about the candidate is a sense of what they consider to be difficult."

Google wants candidates with analytical skills.

Basic computer science skills will do, bock says, since they signal "the ability to understand and apply information" and think in a formal, logical, and structured way. But there are options beyond CS. Bock says that taking statistics while he was in business school was "transformative" for his career.

"Analytical training gives you a skill set that differentiates you from most people" he says.

Google expects people to meet ridiculously high standards.

"We don't compromise our hiring bar, ever," bock says. Because of this, job listings stay open longer at google than you'd expect, he says — they have to kiss a lot of frogs before finding the one. Google doesn't care about Top gpas (Grade Point AverageS). They'll be zealots about their point of view.

« Google doesn't look for deep, deep experts » (3/4)

Gpas (Grade Point AverageS) and test scores don't correlate with success at the company.

"Academic environments are artificial environments. People who succeed there are sort of finely trained; they're conditioned to succeed in that environment," bock says.

While in school, people are trained to give specific answers. "It's much more interesting to solve problems where there isn't an obvious answer," bock says. "You want people who like figuring out stuff where there is no obvious answer."

Google wants to know how much candidates have accomplished compared to their peers.

When bock was explaining how to write resumes to Thomas Friedman at the times, he said that most people miss that the formula for writing quality resumes is simple: "i accomplished x, relative to y, by doing z."

For example, bock explained that a lot of people would just write, "i wrote editorials for the new york times."

But a stand-out resume would be more specific about their accomplishments and how they compared to others. Bock gives a better example: "had 50 op-eds published compared to average of 6 by most op-ed [writers] as a result of providing deep insight into the following area for three years."

« Google doesn't look for deep, deep experts » (4/4)

Google looks for employees who know when to step up and take a leadership role.

Bock doesn't care for "traditional leadership.“ - "We don't care," he insists. "What we care about is, when faced with a problem and you're a member of a team, do you, at the appropriate time, step in and lead. And just as critically, do you step back and stop leading, do you let someone else? Because what's critical to be an effective leader in this environment is you have to be willing to relinquish power."

Google wants to see people who take ownership of projects.

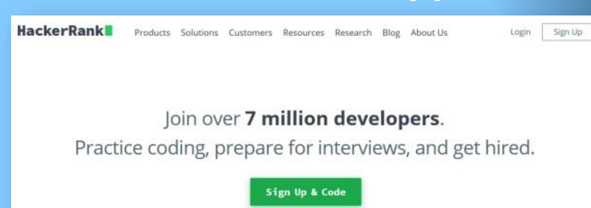
With that sense of ownership, you'll feel responsible for the fate of a project, making you ready to solve any problem. But you also need to defer when other people have better ideas: "your end goal," explained bock, "is what can we do together to problem-solve. I've contributed my piece, and then I step back."

Google wants to see humility, too.

You need "intellectual humility" to succeed at google. "Without humility, you are unable to learn." This is a common problem among the well-educated; elite school grads tend to plateau. Successful folks don't often experience failure. So they don't know how to learn from failure. Instead of having an opportunity to learn, they blame others. Bock explains: they, instead, commit the fundamental attribution error, which is if something good happens, it's because i'm a genius. If something bad happens, it's because someone's an idiot or I didn't get the resources or the market moved. ... But people who are the most successful here, who we want to hire, will show humility.

Le Hacker Rank (par pays… 1/2) - évaluation des développeurs :

The study looked at the results of 1.4 million of hackerrank's coding test submissions, called "challenges," during the last few years.



"According to our data, china and russia score as the most talented developers.

Chinese programmers outscore all other countries in mathematics, functional programming, and data structures challenges, while **Russians** dominate in algorithms, the most popular and most competitive arena," said Ritika Trikha, a blogger at hacker rank.

United States and India provide the majority of competitors on hacker rank but only manage to rank 28th and 31st, respectively. "If we held a hacking olympics today, our data suggests that China would win the gold, Russia would take home a silver, and Poland would nab the bronze," Trikha said. "Though they certainly deserve credit for making a showing, the United States and India have some work ahead of them before they make it at least into the top 25."

Une évaluation des développeurs de différents pays (2/2)

Hacker rank's coding challenges cover aspects of computing ranging from languages to algorithms, security and distributed systems. Developers are scored based on a combination of accuracy and speed. The algorithms category has nearly 40 percent of developers competing, featuring tests on sorting data, dynamic programming, keyword searches and other logic-based tasks. Following algorithms were java and data structure tests, with 10 percent of developers participating. Distributed systems and security were the least popular tests, although thousands still took them.

To determine which nation had the highest-scoring programmers, HackerRank looked at each country's average score across domains. Data was restricted to the top 50 countries with the most developers on HackerRank. Following China (1) and Russia(2) with the top developers were Poland (3), Switzerland (4), Hungary (5), Japan (6), Taiwan (7), **France** (8), Czech republic (9), and Italy (10).

The 100 score represents the country's being first in the rankings (China). "But China only won by a hair. Russia scored 99.9 out of 100, while Poland and Switzerland round out the top rankings with scores near 98. Pakistan scores only 57.4 out of 100 on the index, (ranking 50th)."

Poland was tops in java testing, **France led in C++,** Hong Kong in Python, Japan in artificial intelligence, and Switzerland in databases. Ukrainian programmers led in security, while Finland was top in ruby coding challenges.

AU PROGRAMME
MAITRISER
LA COMPLEXITÉ

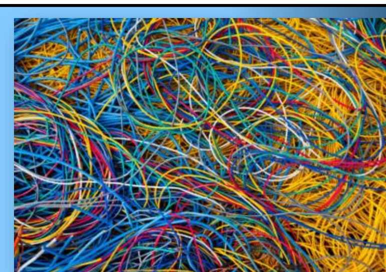
16

La complexité

Pour écrire un programme efficace, il faut un algorithme de bonne complexité face au problème.

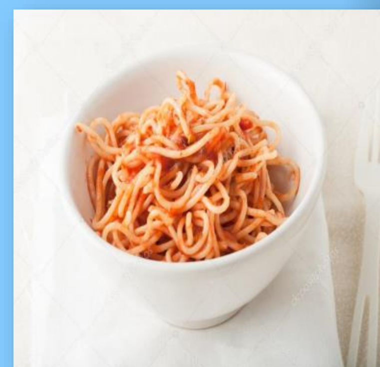
La complexité d’exprime en fonction de la taille de l’entrée ou un paramètre de l’entrée ‘n’

Pour comparer les complexités on utilise la notation de Landau.



Hiérachie, du terme le plus important au moins important (liste non-exhaustive) *(suite)*

- **Exponentiel** en n : **O(2^n)**
- **Polynomial** en n : **O(n^k)**
- Un terme **logarithmique**, c'est-à-dire de la forme **c x $\log_2(n)$** où c ne dépend pas de n. On le notera **O(log(n))**. ***Exemple:*** **6 $\log_2(n)$** opérations.
- Un terme **constant**, qui ne dépend pas de n. On le notera **O(1)**. **Exemple: 45** opérations.



Ordre de grandeur du temps nécessaire à l'exécution d'un algorithme d'un type de complexité

Temps	Type de complexité	Temps pour n = 5	Temps pour n = 10	Temps pour n = 20	Temps pour n = 50	Temps pour n = 250	Temps pour n = 1 000	Temps pour n = 10 000	Temps pour n = 1 000 000	Problème exemple
$O(1)$	complexité constante	10 ns	10 ns	10 ns	10 ns	10 ns	10 ns	10 ns	10 ns	accès à une cellule de tableau
$O(\log(n))$	complexité logarithmique	10 ns	10 ns	10 ns	20 ns	30 ns	30 ns	40 ns	60 ns	recherche dichotomique
$O(\sqrt{n})$	complexité racinaire	22 ns	32 ns	45 ns	71 ns	158 ns	316 ns	1 µs	10 µs	test de primalité naïf
$O(n)$	complexité linéaire	50 ns	100 ns	200 ns	500 ns	2.5 µs	10 µs	100 µs	10 ms	parcours de liste
$O(n \log(n))$	complexité linéarithmique	40 ns	100 ns	260 ns	850 ns	6 µs	30 µs	400 µs	60 ms	tris par comparaisons optimaux (comme le tri fusion ou le tri par tas)
$O(n^2)$	complexité quadratique (polynomiale)	250 ns	1 µs	4 µs	25 µs	625 µs	10 ms	1 s	2.8 heures	parcours de tableaux 2D

18

$O(n^2)$	complexité quadratique (polynomiale)	250 ns	1 µs	4 µs	25 µs	625 µs	10 ms	1 s	2.8 heures	parcours de tableaux 2D
$O(n^3)$	complexité cubique (polynomiale)	1.25 µs	10 µs	80 µs	1.25 ms	156 ms	10 s	2.7 heures	316 ans	multiplication matricielle naïve
$2^{\text{poly}(\log(n))}$	complexité sous-exponentielle	30 ns	100 ns	492 ns	7 µs	5 ms	10 s	3.2 ans	10^{20} ans	factorisation d'entiers avec GNFS (le meilleur algorithme connu en 2018)
$2^{\text{poly}(n)}$	complexité exponentielle	320 ns	10 µs	10 ms	130 jours	10^{59} ans	...	...	...	problème du sac à dos par force brute
$O(n!)$	complexité factorielle	1.2 µs	36 ms	770 ans	10^{48} ans	...	...	...	...	problème du voyageur de commerce avec une approche naïve
$2^{2^{\text{poly}(n)}}$	complexité doublement exponentielle	4.3 s	10^{278} ans	...	...	...	...	...	...	décision de l'arithmétique de Presburger

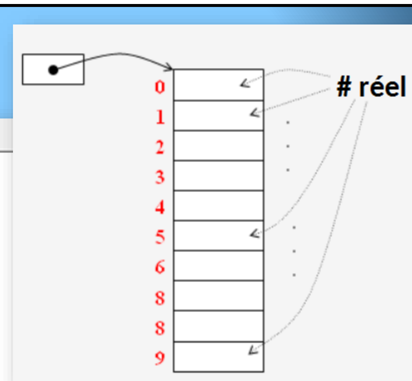
Computer Science

	2^n (Level 0)	n^2 (Level 1)	n (Level 2)	log(n) (Level 3)	Comments
data structures	Doesn't know the difference between Array and LinkedList	Able to explain and use Arrays, LinkedLists, Dictionaries etc in practical programming tasks	Knows space and time tradeoffs of the basic data structures, Arrays vs LinkedLists, Able to explain how hashtables can be implemented and can handle collisions, Priority queues and ways to implement them etc.	Knowledge of advanced data structures like B-trees, binomial and fibonacci heaps, AVL/Red Black trees, Splay Trees, Skip Lists, tries etc.	
algorithms	Unable to find the average of numbers in an array (It's hard to believe but I've interviewed such candidates)	Basic sorting, searching and data structure traversal and retrieval algorithms	Tree, Graph, simple greedy and divide and conquer algorithms, is able to understand the relevance of the levels of this matrix.	Able to recognize and code dynamic programming solutions, good knowledge of graph algorithms, good knowledge of numerical computation algorithms, able to identify NP problems etc.	Working with someone who has a good topcoder ranking would be an unbelievable piece of luck!
systems programming	Doesn't know what a compiler, linker or interpreter is	Basic understanding of compilers, linker and interpreters. Understands what assembly code is and how things work at the hardware level. Some knowledge of virtual memory and paging.	Understands kernel mode vs. user mode, multi-threading, synchronization primitives and how they're implemented, able to read assembly code. Understands how networks work, understanding of network protocols and socket level programming.	Understands the entire programming stack, hardware (CPU + Memory + Cache + Interrupts + microcode), binary code, assembly, static and dynamic linking, compilation, interpretation, JIT compilation, garbage collection, heap, stack, memory addressing...	

NIVEAU 1 – LES TABLEAUX

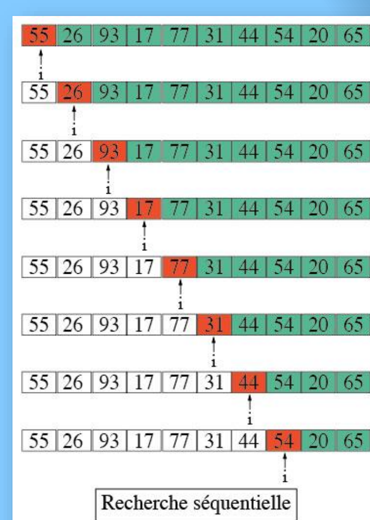
fibo.java | RandomDemo.java

```java
public class RandomDemo {
   public static void main( String args[] ){
        // creer un générateur de nombres pseudo-aléatoires
        Random randomGen = new Random();
        float  randArray[];
        float  Phi6D = (float) 0.618033;

        randArray = new float [10000000];

        // Remplir le tableau de valeurs aléatoires
        for( int i = 0; i < 10000000 ; ++i) {
            randArray[i] = randomGen.nextFloat();
        }

        // Rechercher si on les décimales de Phi
        for( int i = 0; i < 10000000 ; ++i) {
            System.out.println("Rand[" + i + "] = " + randArray[i]);
            if (Phi6D == randArray[i]) {
            System.out.println("6 Décimales trouvées : " + randArray[i]);
            }
        }
    }
}
```



RECHERCHE SEQUENTIELLE

- Au maximum N tests pour N valeurs dans le tableau.
- En moyenne N/2 tests



CETTE RECHERCHE EST DITE « D'ORDRE 'N' » LE MAXIMUM DE TESTS

Le maximum N tests est l'ordre de grandeur du temps d'attente maximal

Avec beaucoup de chance 1 tests ☺ !

```
Rand[249300] = 0.66389877
Rand[249301] = 0.0677259
Rand[249302] = 0.6618339
Rand[249303] = 0.2822563
Rand[249304] = 0.030273736
Rand[249305] = 0.21487772
Rand[249306] = 0.6430948
Rand[249307] = 0.016756117
Rand[249308] = 0.7717602
Rand[249309] = 0.3684681
Rand[249310] = 0.4684795
Rand[249311] = 0.17858976
Rand[249312] = 0.1234051
Rand[249313] = 0.4094249
Rand[249314] = 0.16900414
Rand[249315] = 0.75723165
Rand[249316] = 0.18286616
Rand[249317] = 0.13986063
Rand[249318] = 0.18248314
Rand[249319] = 0.5175898
Rand[249320] = 0.16588879
Rand[249321] = 0.49524796
Rand[249322] = 0.6256941
Rand[249323] = 0.07022601
Rand[249324] = 0.63770694
Rand[249325] = 0.100140214
Rand[249326] = 0.75005656
Rand[249327] = 0.959113
Rand[249328] = 0.90208834
Rand[249329] = 0.78084624
```

NIVEAU 2 - RECHERCHE DITE DICHOTOMIQUE

Le maximum de tests sera de l'ordre de log2(N)
Ex: pour 4 milliards de valeurs, 32 comparaisons au maximum…

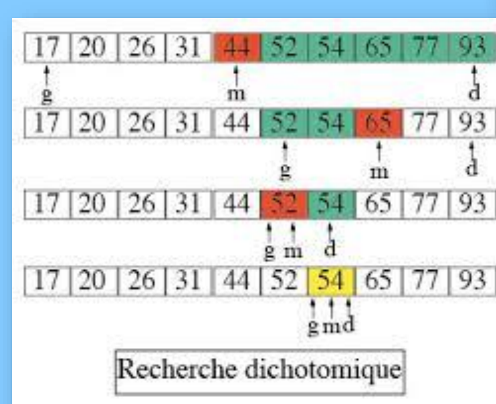
(1) Le tableau doit être trié (ordre croissant par exemple)

(2) On regarde au milieu du tableau

Si l'élément du milieu est plus petit,
Alors on va regarder dans le sous tableau à droite

Sinon on va regarder dans le sous tableau à gauche…

Finsi



NIVEAU 3
PREPARATION D'UN QUICKSORT

Un tri sophistiqué appelé – quicksort
Complexité moyenne : n.Log(n)
Linéarithmique / quasi-linéaire

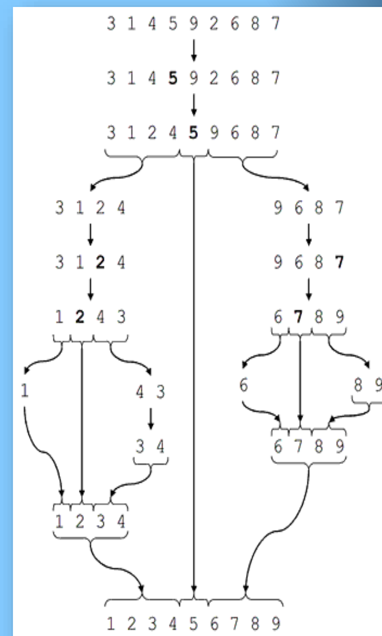
<u>Principe:</u>

On choisit une valeur « pivot »

Toutes les valeurs au dessus du pivot passent dans le sous tableau à droite.

Toutes les valeurs en dessous passent dans le sous tableau à gauche.

On recommence pour chaque sous tableau



NIVEAU 3– EXTRAIT DE CODE JAVA – QUICKSORT RECURSIF

```java
public static int Partition(int[] numbers, int left, int right)
{
    int pivot = numbers[left];

    while (true)
    {
        while (numbers[left] < pivot) left++;
        while (numbers[right] > pivot) right--;

        if (left < right) {
            int temp = numbers[right];

            numbers[right] = numbers[left];
            numbers[left] = temp;
        }
        else {
            return right;
        }
    }
}
```

```java
public static void main(String[] args)
{
    int[] numbers = { 3, 8, 7, 5, 2, 1, 9, 6, 4 };
    int len = 9;

    System.out.println("QuickSort By Recursive Method");

    QuickSort_Recursive(numbers, 0, len - 1);

    for (int i = 0; i < 9; i++) {
        System.out.println(numbers[i]);
```

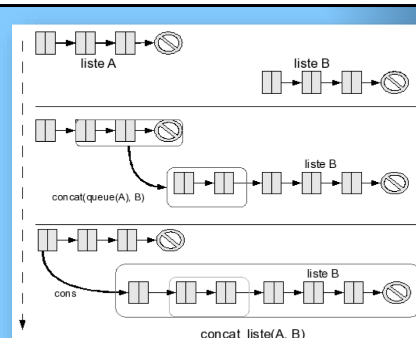
```java
public static void QuickSort_Recursive(int[] arr, int left, int right)
{
    if(left < right)
    {
        int pivot = Partition(arr, left, right);

        if(pivot > 1) QuickSort_Recursive(arr, left, pivot - 1);
        if(pivot + 1 < right) QuickSort_Recursive(arr, pivot + 1, right);
    }

}
```

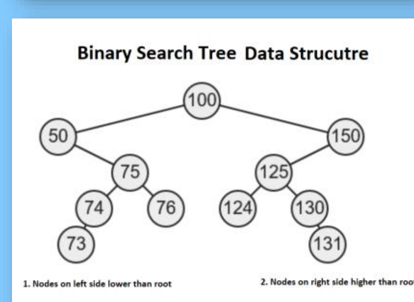
Synthèse en français des principales complexités algorithmique

http://fr.cppreference.com/w/cpp/complexity

Complexité	Vitesse	Description	Formulation	Exemple
Factorielle	très lent	temps d'exécution proportionnel à N^N	N!	Résolution par recherche exhaustive du problème du voyageur de commerce.
Exponentielle	lent	temps d'exécution proportionnel à une valeur donnée à la puissance N	K^N	Résolution par recherche exhaustive du Rubik's Cube.
Polynomiale	moyen	temps d'exécution proportionnel à N à une puissance donnée	N^K	Tris par comparaison, comme le tri à bulle (N^2).
Quasi-linéaire	assez rapide	temps d'exécution intermédiaire entre linéaire et polynomial	N * log(N)	Tris quasi-linéaires, comme le Quicksort.
Linéaire	rapide	temps d'exécution proportionnel à N	N	Itération sur un tableau.
Logarithmique	très rapide	temps d'exécution moyen proportionnel au logarithme de N	log(N)	Recherche dans un arbre binaire.
Constante	le plus rapide	temps d'exécution donné, quel que soit le nombre d'éléments	1	Recherche par index dans un tableau.

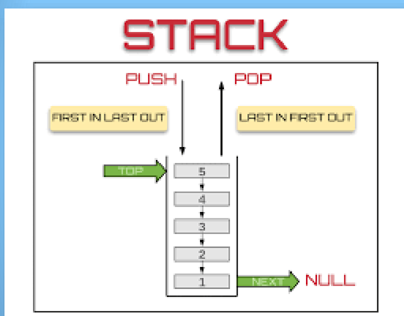
LES STRUCTURES DE DONNEES ESSENTIELLES

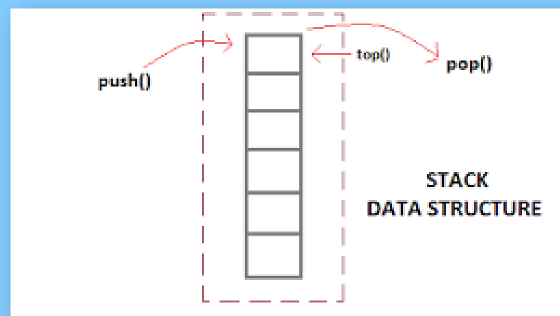




28

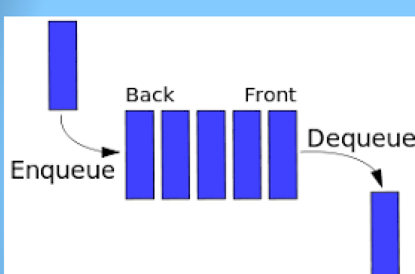
LES PILES

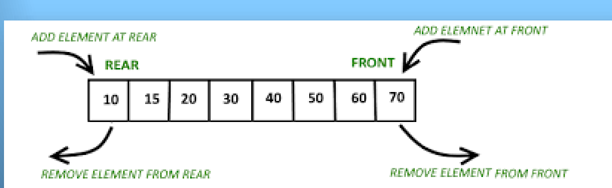




29

LES FILES (FIFO ou PAPS)





Fisrt In – First Out

Ou Premier Arrivé – Premier Servi
Utile avec la pile pour les parcours d'arbre.

Double-ended queue

Il existe également des files à 2 bouts.

Nommée souvent : deque ou dequeue ()

Cette SDD est utile pour les parcours de graphe.

Elle permet le retrait en fin de file pop()
Les ajouts en fin de file append(element)

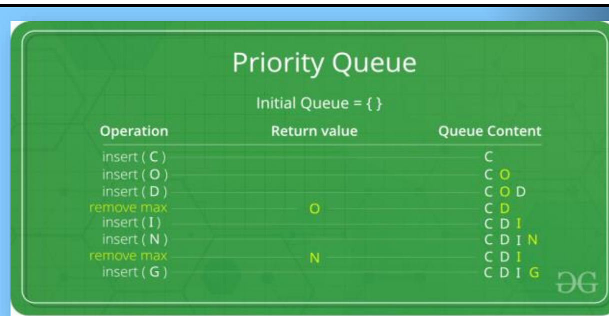
L'ajout en tête de file appendLeft(element)
Le retrait en début de file popleft()

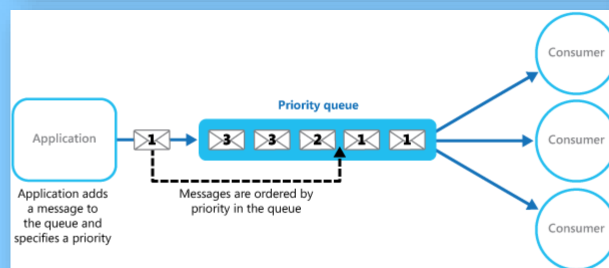
30

FILE DE PRIORITE (priority queues)

Une file de priorité est un type abstrait permettant d'ajouter des éléments et de retirer un élément de clé ordonné.

Utile dans la construction d'un code de compression style « Huffman » ou dans la recherche d'un plus cours chemin entre deux nœuds d'un graphe (algorithme de Dijkstra).



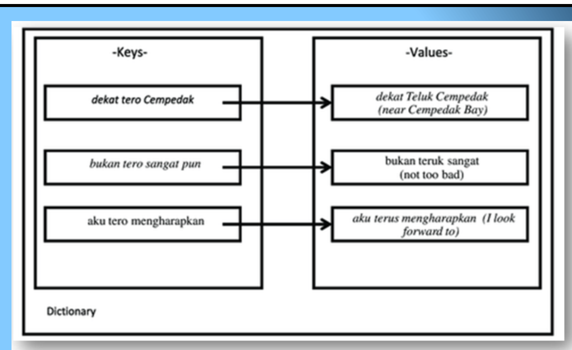


31

LES DICTIONNAIRES

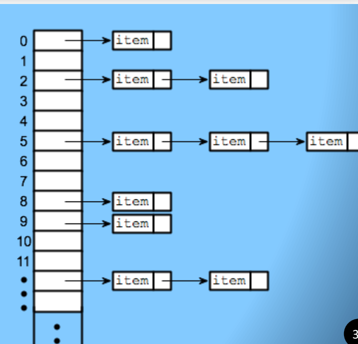
Un dictionnaire permet d'associer un valeur à une clé

Le fonctionnement interne est basé sur une table de hachage (adressage dispersé) pour associer aux éléments un indice dans une table grâce à une fonction de hachage.



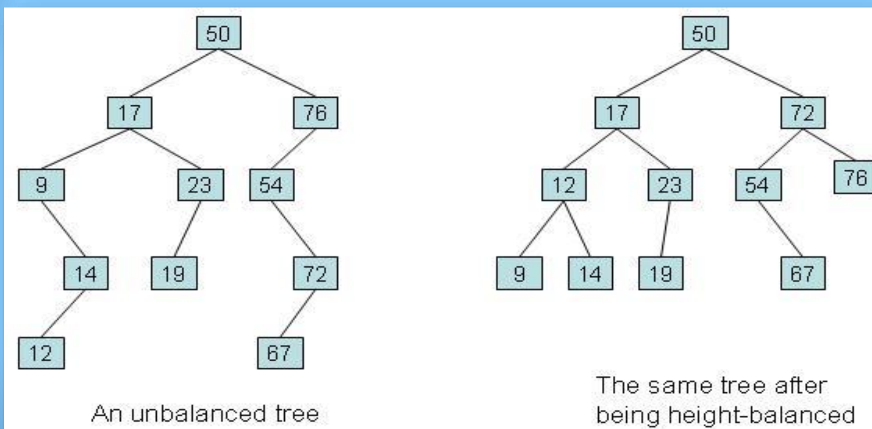

```
unsigned long
hash(unsigned char *str)
{
    unsigned long hash = 5381;
    int c;

    while (c = *str++)
        hash = ((hash << 5) + hash) + c;

    return hash;
}
```

Les tables implémentent un mécanisme de gestion des collisions.



32

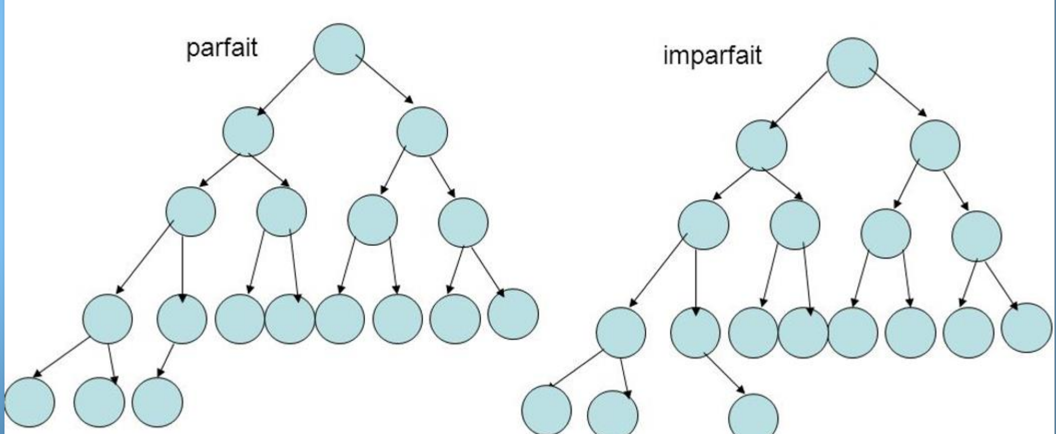
ARBRES BINAIRES EQUILIBRES



33

ARBRES BINAIRES PARFAITS

- tous les niveaux sont complètement remplis, sauf éventuellement le dernier niveau. Dans ce dernier cas les nœuds (feuilles) du dernier niveau sont groupés le plus à gauche possible.



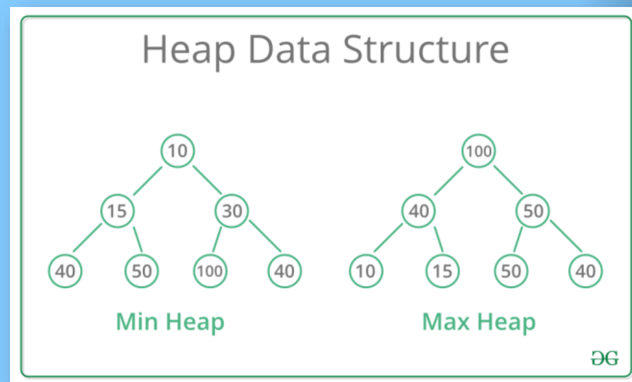
34

LES TAS

Un tas (heap) en anglais est appelé arbre tournoi. C'est un arbre vérifiant la propriété de « tas » - il est partiellement trié.

Les fils de chaque nœud ont une clé plus grande de le nœud racine pour une certaine relation d'ordre (Min Heap)

Il existe une variante ou les fils ont tous une clé plus petite (Max Heap)



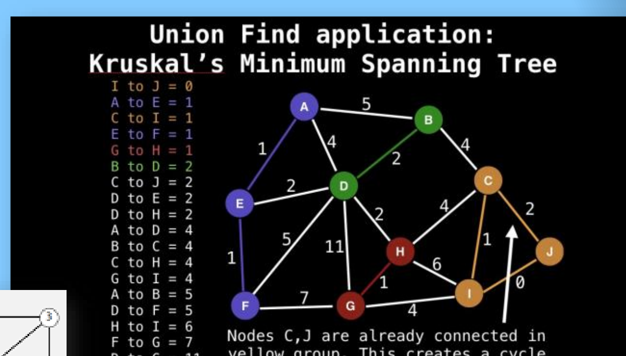
Quand ces arbres sont binaires, on peut extraire l'élément le plus petit immédiatement et insérer des éléments avec un coût logarithmique.

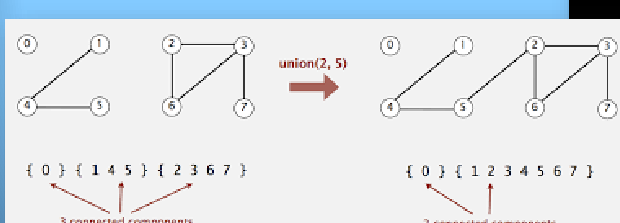
35

AUTRES STRUCTURES

Union-Find (pour déterminer des composantes connexes de graphe)

Structures de données en temps quasi-constant par requête





36

LES TECHNIQUES UTILES

- Comparer des éléments
- Trier
- Balayage (parcours de gauche à droite avec un traitement spécifique à chaque élément)
  > for_each
- Algorithmes gloutons – établissement de solution pas à pas avec un choix de maximisation de critère local (voir structures combinatoires de type matroïdes)
- Programmation dynamique – décomposer un problème en sous-problèmes et trouver la solution optimale du problème principal à partir des solutions aux sous-problèmes.

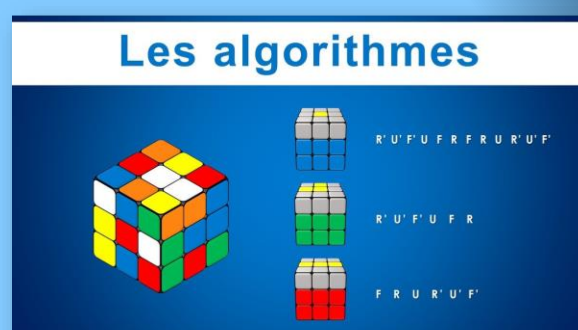
Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

37

AUTRES TECHNIQUES UTILES

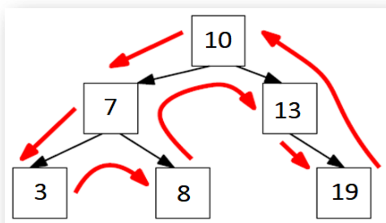
- Coder des ensembles dans des entiers
- Partages équitables en 3 parties
- Recherche dichotomique
- Recherche sans bornes supérieures connues
- Inverser une fonction
- Recherche trichotomiques
- Recherche dans un intervalle [0.. 2^k]
- …



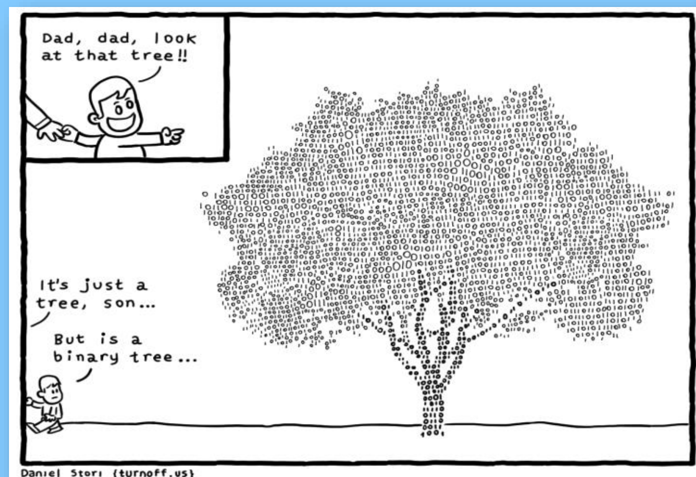
38

Echauffement…
Un parcours d'arbre de recherche trié G-D
Avec un parcours récursif – Niveau « Alogo L1 »

```c
typedef struct node
{
    unsigned int    key;
    struct node   * left;
    struct node   * right;
} node_t;
```

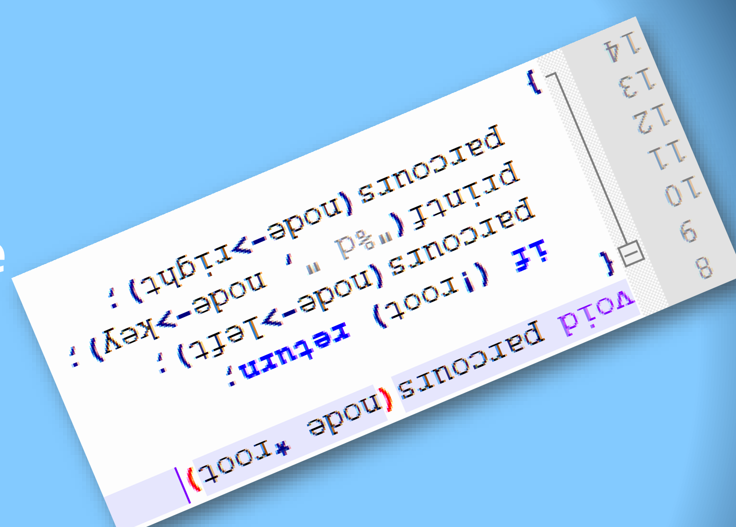


Parcours voulu: 3, 7, 8, 10, 13, 19



39

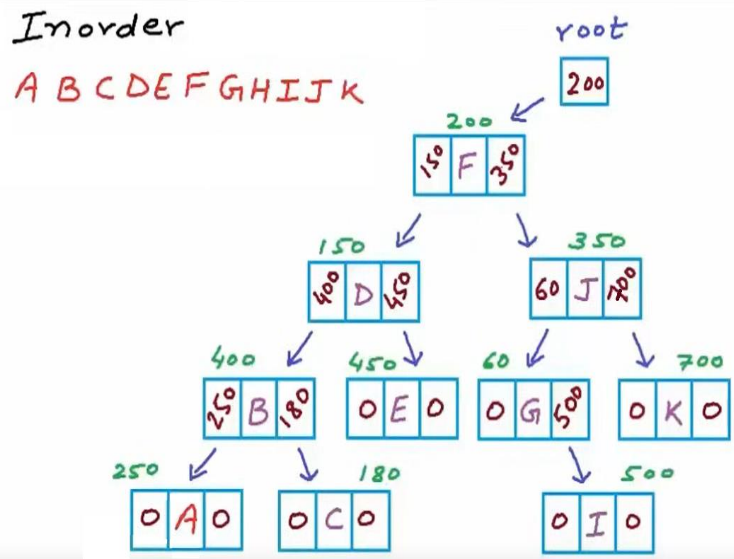
Approche récursive minimaliste usuelle



Les parcours les plus efficaces sont les parcours dé-récursifiés.

40

```c
void Inorder(Node *root)
{
    if(root == NULL) return;

    Inorder(root->left);
    printf("%c ",root->data);
    Inorder(root->right);
}
```



Pre-order ?

Post-order ?

Mycodeschool.com

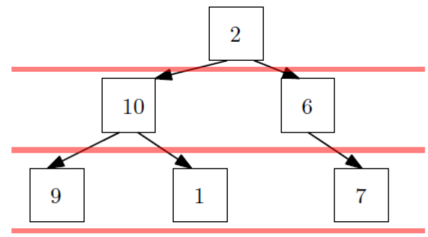
Attention à ne pas oublier de tester la racine.

Une possibilité existe d'avoir un double test pour éviter un appel récursif de fonction (évaluer l'impact de la redondance)

```c
void printTree(node *tree)
{
    if(!tree) return;

    if(tree->left)  printTree(tree->left);

    printf("Cle = %d\n", tree->key);

    if(tree->right) printTree(tree->right);
}
```

Rappel L1-S2 : parcours d'arbre binaire de recherche en largeur avec un file.

```
Proc parcoursEnLargeur(nœud racine):
    file F:=NewFile()
    enfiler(F, racine)
    nœud N
    Tant que non(estVide(F)):
        N:=valeurDebut(F)
        print(N.data)
        defiler(F)
        Si N.filsG=! -1: /* si N a un fils gauche */
            enfiler(F, N.filsG)
        Si N.filsD!=-1:
            enfiler(F, N.filsD)
```

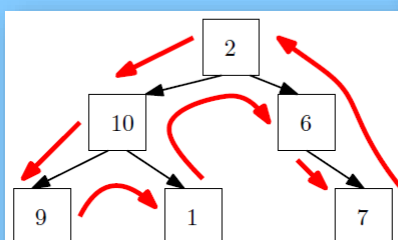


Parcours voulu: **2, 10, 6, 9, 1, 7.**

43

Idem pour parcours d'arbre binaire en profondeur (dé-récursifié). Cette fois il faut une pile.

```
Proc parcoursEnProfondeur(nœud racine):
    pile P:=NewPile()
    empiler(P, racine)
    nœud N
    Tant que non(estVide(P)):
        N:=valeurTop(P)
        print(N.data)
        depiler(P)
        Si N.filsG=! -1: /* si N a un fils gauche */
            empiler(P, N.filsG)
        Si N.filsD!=-1:
            empiler(P, N.filsD)
```



Parcours voulu: **2, 10, 9, 1, 6, 7.**

44

MERCI DE VOTRE ATTENTION

David Hill
+33 (0) 473 40 50 19
David.Hill@uca.fr
www.isima.fr/~hill/AAC

45