

Algorithmes et structures de données Travail Écrit 2

15.06.2021

Nom :

Prénom :

Note :

|

Durée du test : 1h30

Documents autorisés : 6 feuilles de notes personnelles (recto/verso : 12 pages)

Modalités d'évaluation : Le travail écrit est noté sur **52** points. La répartition des points par exercice est communiquée dans le tableau ci-dessous.

Exercice	Points
1	8
2	7
3	10
4	8
5	9
6	10

IMPORTANT : veuillez indiquer vos nom et prénom sur chacune des feuilles.

1. Qu'affiche chaque ligne ? [8 points] 6,5

Soient les headers et la définition de la classe *Cours* suivante :

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>

using namespace std;
class Cours {
    string nom;
    double note;

public:
    Cours() : nom(""), note(1) {
        cout << "Cd";
    }
    Cours(string _nom, double _note) : nom(_nom), note(_note) {
        cout << "Cp";
    }
    Cours(const Cours& _p) : nom(_p.nom), note(_p.note) {
        cout << "Cc";
    }
    Cours(Cours&& _p) noexcept : nom(_p.nom), note(_p.note) {
        _p.nom = ""; _p.note = 0;
        cout << "Cm";
    }
    Cours& operator= (const Cours& _p) {
        if(&_p == this) return *this;
        nom = _p.nom; note = _p.note;
        cout << "Ac";
        return *this;
    }
    Cours& operator= (Cours&& _p) noexcept {
        if(&_p == this) return *this;
        nom = _p.nom; note = _p.note;
        _p.nom = ""; _p.note = 0;
        cout << "Am";
        return *this;
    }
    ~Cours() { cout << "D"; }
};
```

Indiquez à coté de chaque ligne du programme de la page suivante ce qu'elle affiche.

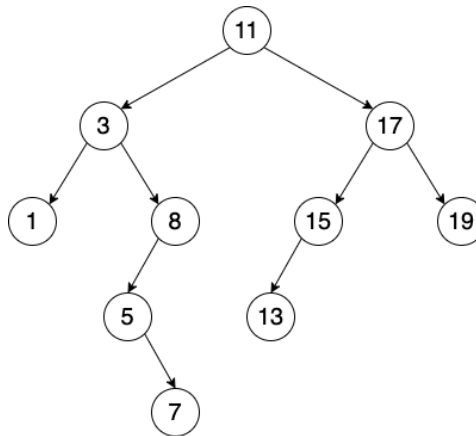
Nom :

Prénom :

Code	Affichage	
int main() {		
vector<Cours> v(1);	Cd	Cd ✓
v.reserve(8);	Cm D	x
Cours c1("ASD", 6);	Cp	Cp ✓
v.push_back(c1);	Cc	Cc ✓
{		
Cours c2;	Cd	cd ✓
v.push_back(c2);	Cc	Cc ✓
}	D	D ✓
Cours* c3;		
{		
c3 = new Cours;	Cd	Cd ✓
v.push_back(*c3);	Cc	Cc ✓
}		D x
v.push_back(Cours("PRG2", 5.5));	Cp Cm D	CpCmD ✓
v.emplace_back("ISD", 5.7);	Cp	Cp ✓
v.resize(4);	D D	D D x
{		
swap(v[1], v[2]);	Cm Am Am D	CmAmAmD ✓
v[0] = Cours("MAT2", 5);	Cp Am D	CpAmD ✓
}		
delete c3;	D D	✓
return 0;		
}	D D D D D (c1 supprimé également) D D D D D x	

2. Arbres binaires [7 pts] 7

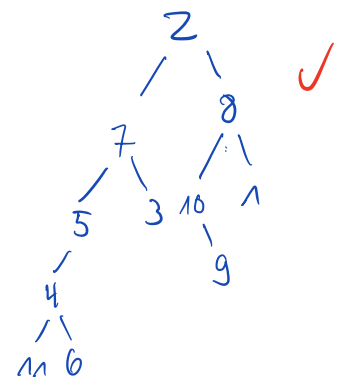
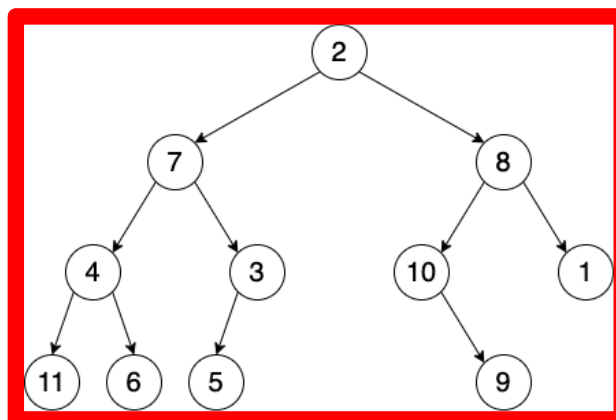
a. Pour l'**arbre binaire de recherche** ci-dessous, veuillez indiquer les différents parcours possibles.



- Parcours pré-ordonné : 11 3 1 8 5 7 17 15 13 19 11 3 1 8 5 7 17 15 13 19 ✓
- Parcours symétrique : 1 3 5 7 8 11 13 15 17 19 1 3 5 7 8 11 13 15 17 19 ✓
- Parcours post-ordonné : 17 5 8 3 13 15 19 17 11 17 5 8 3 13 15 19 17 11 ✓
- Parcours en largeur : 11 3 17 1 8 15 19 5 13 7 11 3 17 1 8 15 19 5 13 7 ✓

b. Dessinez l'**arbre binaire** dont on connaît les résultats des parcours suivants :

- Parcours post-ordre : 11 - 6 - 4 - 5 - 3 - 7 - 9 - 10 - 1 - 8 - 2
- Parcours symétrique : 11 - 4 - 6 - 7 - 5 - 3 - 2 - 10 - 9 - 8 - 1



Nom :

Prénom :

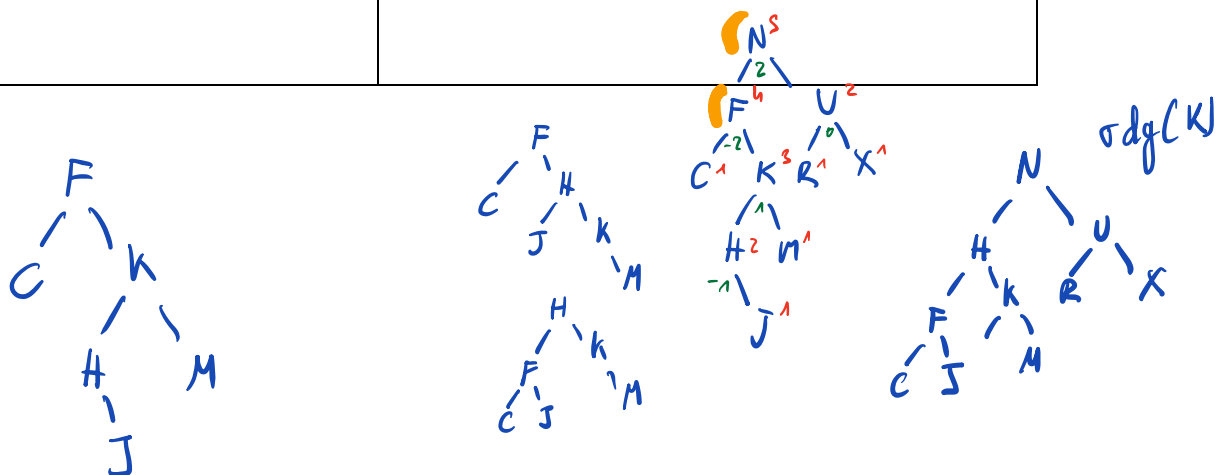
3. Arbres AVL [10 points] 1

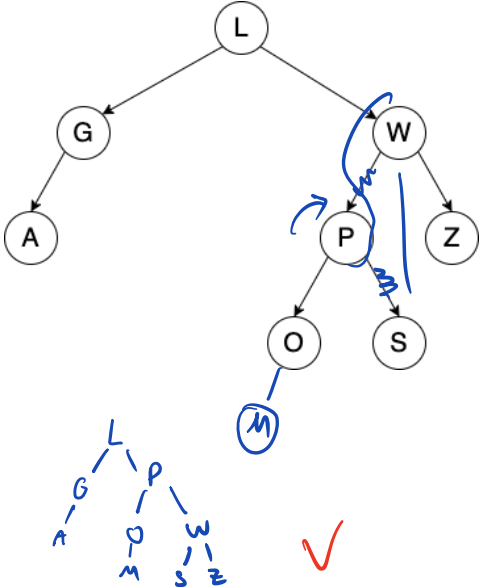
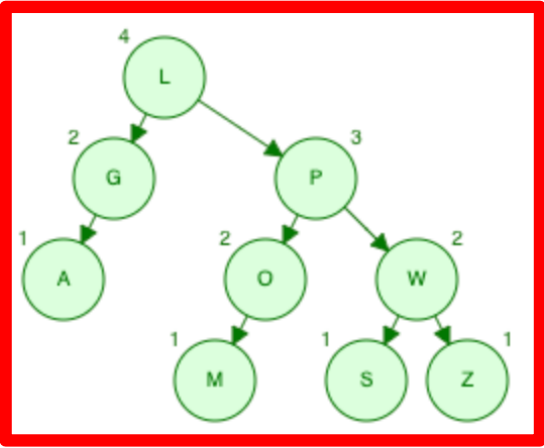
Pour chacun des cinq arbres AVL suivants, dessiner le résultat final de l'opération correspondante, en précisant les rotations effectuées sur quel nœud : $rg(nœud)$, $rd(nœud)$, $rgd(nœud)$, $rdg(nœud)$. (2pts pour chaque)

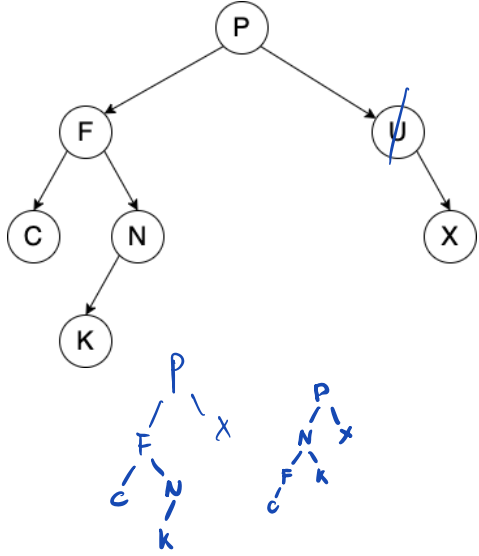
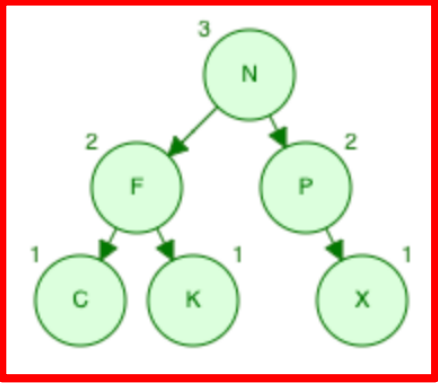

S'il faut supprimer un nœud avec 2 fils, privilégiez la recherche du remplaçant dans le sous-arbre gauche.

Pour rappel, voici l'alphabet français dans l'ordre : ABCDEFGHIJKLMNOPQRSTUVWXYZ

Insertion de J	
	Rotations : $rdg(F)$
	Résultat final :



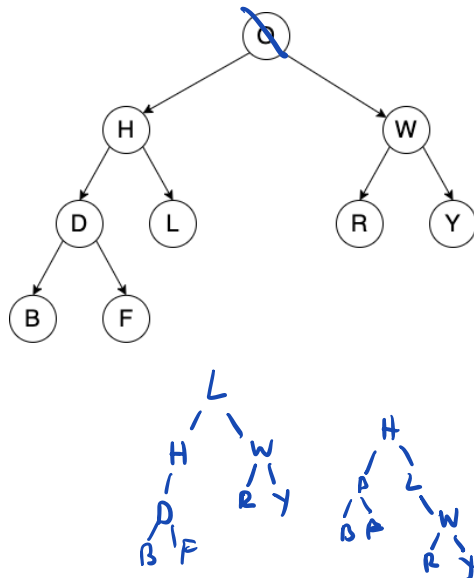
Insertion de M	
	<p>Rotations :</p> <p>rd(W) rd(P) ✗</p> <p>Résultat final :</p> 

Suppression de U	
	<p>Rotations :</p> <p>rgd(P) rgd(N)</p> <p>Résultat final :</p>  

Nom :

Prénom :

Suppression de O

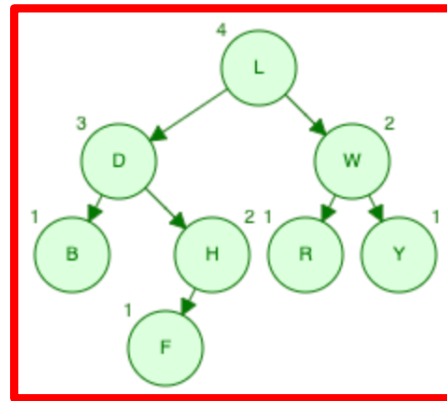


Rotations :

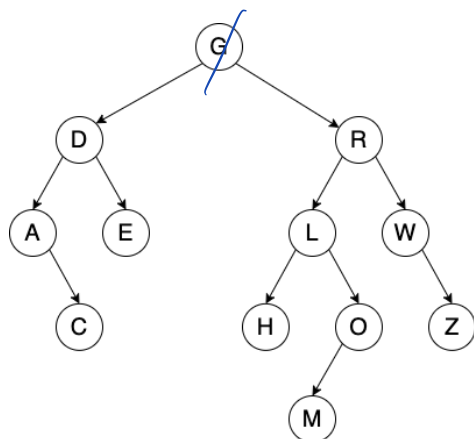
rd(H)

rd C #)

Résultat final :



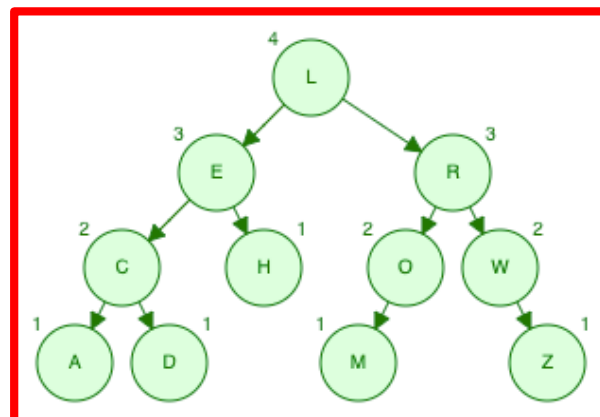
Suppression de G



Rotations :

rgd(D) et rdg(E)

Résultat final :



4. Tas [8 pts] 8

Effectuer le tri par tas selon l'ordre alphabétique sur le tableau de 8 éléments suivant est effectuant en premier un `make_heap` puis un `sort_heap` :

1 2 3 4 5 6 7 8

Indiquer l'état du tableau après chaque descente d'éléments, ainsi que le résultat final.

make_heap

1 2 3 4 5 6 7 8

1 2 3 8 5 6 7 4

1 2 3 8 5 6 7 4 ✓

1 2 7 8 5 6 3 4

1 8 3 2 5 6 7 4

1 8 7 2 5 6 3 4

1 8 7 2 5 6 3 4

8 1 7 2 5 6 3 4

1 8 7 4 5 6 3 2

8 5 7 2 1 6 3 4

8 1 7 4 5 6 3 2

8 5 7 4 1 6 3 2 ✓

8 5 7 4 1 6 3 2

sort_heap

8 5 7 4 1 6 3 2

2 5 7 4 1 6 3 8

2 5 7 4 1 6 3 8

7 5 6 4 1 2 3 8

7 5 2 4 1 6 3 8

3 5 6 4 1 2 7 8

7 5 6 4 1 2 3 8

6 5 3 4 1 2 7 8

3 5 6 4 1 2 7 8

2 5 3 4 1 6 7 8

6 5 3 4 1 2 7 8

5 4 3 2 1 6 7 8

2 5 3 4 1 6 7 8

1 4 3 2 5 6 7 8

5 2 3 4 1 6 7 8

4 2 3 1 5 6 7 8

5 4 3 2 1 6 7 8

1 2 3 4 5 6 7 8

1 4 3 2 5 6 7 8

3 2 1 4 5 6 7 8

4 1 3 2 5 6 7 8

1 2 3 4 5 6 7 8

4 2 3 1 5 6 7 8

2 1 3 4 5 6 7 8

1 2 3 1 4 5 6 7 8

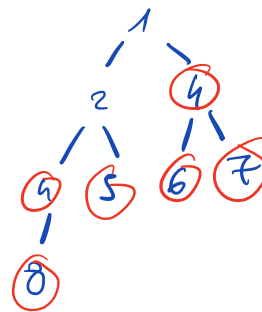
1 2 3 4 5 6 7 8

3 2 1 1 4 5 6 7 8 ✓

1 2 1 3 4 5 6 7 8

2 1 1 3 4 5 6 7 8 ✓

1 2 1 3 4 5 6 7 8 ✓



4 pts pour le `make_heap`, 4 pour le `sort_heap`

Nom :

Prénom :

5. Complexités [9 pts] 5

Indiquez la complexité moyenne des fonctions suivantes en fonction des paramètres N et éventuellement M cités dans le code et / ou les commentaires.

<pre>vector<int> f1(int N) { vector<int> v; for(int i = 0; i < N; ++i) v.push_back(i); return v; }</pre>	$O(N)$ N ✓
<pre>set<int> f2(int N) { set<int> s; for(int i = 0; i < N; ++i) { for (int j = 0; j < N; ++j) { s.insert(i + j); } } return s; }</pre>	$O(n^2 \log(n))$ $N^2 \log(N)$ ✓
<pre>map<string,int> f3(string const& livre) { // Livre contient N mots issus d'un // vocabulaire contenant M mots // différents, avec $N \gg M$. istringstream in(livre); map<string,int> wordCounter; string word; while(in >> word) { ++wordCounter[word]; } return wordCounter; }</pre>	$O(N * \log(M))$ $\max(N, M)$ ✗
<pre>void f4(vector<int>& v, int M) { size_t N = v.size(); // on suppose N suffisamment grand // pour que le code suivant ne vide // pas le vecteur for(int i = 1; i < M; i*= 2) { v.erase(v.begin()); } }</pre>	$O(N * \log(M))$ $N^x \log(M)$ ✓

<pre>void f5(list<int>& v, int M) { size_t N = v.size(); // on suppose N suffisamment grand // pour que le code suivant ne vide // pas la liste for(int i = 1; i*i < M; ++i) { v.erase(v.begin()); } }</pre>	<p>$O(\sqrt{M})$</p> <p>\sqrt{M} ✓</p>
<pre>void f6(vector<int>& v) { size_t N = v.size(); make_heap(v.begin(), v.end()); $O(N)$ for(int i = N-1; i > 0; --i) N pop_heap(v.begin(), v.begin()+i); $O(\log(N))$ }</pre>	<p>$O(N \cdot \log(N))$</p> <p>$N^2 \log(N)$ ✗</p>
<pre>void f7(vector<int>& v) { size_t N = v.size(); if(N > 1) { vector<int> v1(v.begin(), v.begin() + N/2); f7(v1); vector<int> v2(v.begin() + N/2, v.end()); f7(v2); std::merge(v1.begin(), v1.end(), v2.begin(), v2.end(), v.begin()); } }</pre>	<p>$O(N \cdot \log(N))$</p> <p>$N/2$ ✗</p>
<pre>vector<int> f8(list<int>& in) { size_t N = in.size(); vector<int> out; out.reserve(N); for(int i = 0; i < N; ++i) out.push_back(*next(in.begin(), i)); return out; }</pre>	<p>$O(N^2)$</p> <p>N^2 ✓</p>
<pre>list<int> f9(vector<int>& in) { size_t N = in.size(); list<int> out; for(int i = 0; i < N; ++i) out.push_back(*next(in.begin(), i)); return out; }</pre>	<p>$O(N)$</p> <p>N^2 ✗</p>

Notation: 1 point par bonne réponse. Pas de points partiels

Nom :










Prénom :







6. Coût en mémoire [10 pts]

Pour chacune des fonctions suivantes, indiquez la taille mémoire utilisée - en **octets** - par la structure retournée, y compris les données stockées, en fonction de N . Supposez pour ce faire qu'un `int` utilise 4 octets, tandis que les pointeurs et les `size_t` utilisent 8 octets.

La réponse est toujours de la forme $A \cdot N^2 + B \cdot N + C$. Indiquez les valeurs des coefficients A , B et C , respectivement.

Pour `std::set`, vous pouvez ignorer le bit de couleur pour l'équilibrage rouge-noir. Certaines versions de la STL mettent en œuvre ces structures moins efficacement que possible. Utilisez ici la taille minimale qui permet de mettre en œuvre ces structures et toutes leurs méthodes avec les complexités garanties par le standard.

Fonction	A	B	C
<pre>vector<vector<int>> make1(size_t N) { vector<vector<int>> m(N); for(size_t i = 0; i < N; ++i) { m[i].reserve(N); for(size_t j = 0; j < N; ++j) m[i].push_back((i+j) % 2); } return m; }</pre>			
<pre>list<list<int>> make2(size_t N) { list<list<int>> m; for(size_t i = 0; i < N; ++i) { m.push_back(list<int>()); for(size_t j = 0; j < N; ++j) m.back().push_front(i*j); } return m; }</pre>			
<pre>forward_list<forward_list<int>> make3(size_t N) { forward_list<forward_list<int>> m; m.push_front(forward_list<int>()); for (size_t i = 1; i < N; ++i) { // copie de la liste précédent m.push_front(m.front()); // la ième liste est plus longue // de 1 que la (i-1)ème m.front().push_front(i); } return m; }</pre>			

<pre>template<size_t N> array<array<int,N>,N> make4() { array<array<int, N>, N> m; for (size_t i = 0; i < N; ++i) { m[i, i] = 1; } return m; }</pre>			
<pre>vector<set<int>> make5(size_t N) { vector<set<int>> m(N); for (size_t i = 0; i < N; ++i) { for (size_t j = 0; j < N; ++j) { m[i].insert((i+j) % 2); } } return m; }</pre>			

Notation: 1 point si A est bon, 2 si A et B sont bons, 3 si A, B et C sont bons. Noté sur 10 même si le total fait 15.