INF3105 – Structures de données et algorithmes
Été 2024 – Examen de mi-session

Jaël Champagne Gareau
Département d'informatique
Université du Québec à Montréal

Mardi 25 juin 2024 – 13h30 à 16h30 (3 heures) – Local PK-1705

Instructions

1. Aucune documentation n'est permise, excepté l'aide-mémoire C++ disponible à la dernière page.
2. Les appareils électroniques, incluant les téléphones et les calculatrices, sont strictement interdits.
3. Répondez directement sur le questionnaire à l'intérieur des endroits appropriés.
4. Pour les questions demandant l'écriture de code :
   — le fonctionnement correct, l'efficacité (temps et mémoire), la clarté, la simplicité du code et la robustesse sont des critères de correction à considérer ;
   — vous pouvez scinder votre solution en plusieurs fonctions ;
   — vous pouvez supposer l'existence de fonctions et de structures de données raisonnables.
5. **Aucune question ne sera répondue durant l'examen**. Si vous croyez qu'une erreur ou qu'une ambigüité s'est glissée dans le questionnaire, indiquez clairement la supposition que vous avez retenue pour répondre à la question.
6. Vous pouvez détachez les annexes à la fin du questionnaire. Évitez de détacher les autres feuilles.
7. Dans l'entête de l'actuelle page, si vous encerclez le numéro de local où a lieu le cours, vous aurez un point boni pour avoir lu les instructions.
8. À l'exception de la question 4a, vous devez répondre à l'aide d'un crayon non rouge.

Identification

Nom : _____

Code permanent : _____

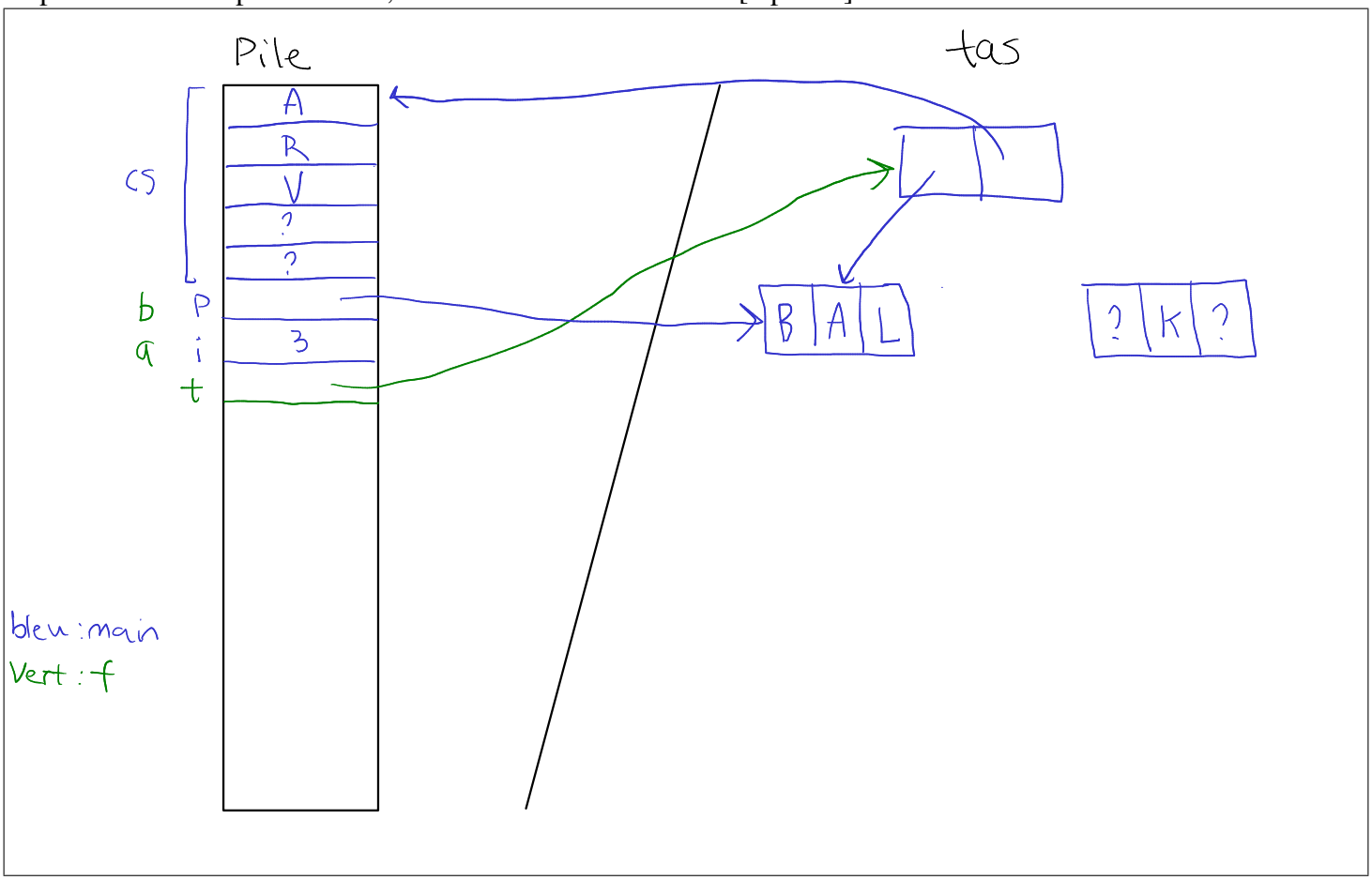
Signature : _____

Résultat

Q1		/ 15
Q2		/ 10
Q3		/ 20
Q4		/ 15
Q5		/ 10
Q6		/ 10
Q7		/ 20
Total		/ 100

1 C++ [15 points]

(a) Faites un dessin mémoire de l'état du programme présenté à l'Annexe A lorsque l'exécution est rendue à la ligne 17. Identifiez bien la séparation entre les variables stockées sur la pile d'exécution et le tas (*heap*). Si un emplacement n'est pas initialisé, mettez ' ?' comme contenu. [5 points]



(b) Qu'affiche le programme ? [5 points]

ABR
AVL
3

(c) Est-ce que ce programme contient une fuite mémoire ? Si oui, dites combien d'octets et expliquez le(s) changement(s) à apporter pour corriger le problème. Si non, décrivez où chacune des ressources allouée dynamiquement est libérée. [5 points]

Oui, 3 octets de fuite mémoire. On peut régler le problème en ajoutant `delete[] t[1].z;` entre les lignes 11 et 12 de la fonction `f`.

2 Questions à choix multiples [10 points]

(a) Pour chacun des types d'arbre de n nœuds suivants, indiquez le choix le plus près de sa hauteur maximale.

(i) Arbre binaire

- ☒ $n-1$
- ☐ n
- ☐ $n+1$
- ☐ $\log_2(n)$
- ☐ $\lfloor 1.44\log_2(n+1)-1.328 \rfloor$
- ☐ $\lfloor 2\log_2(n) \rfloor$

(ii) Arbre AVL

- ☐ $n-1$
- ☐ n
- ☐ $n+1$
- ☐ $\log_2(n)$
- ☒ $\lfloor 1.44\log_2(n+1)-1.328 \rfloor$
- ☐ $\lfloor 2\log_2(n) \rfloor$

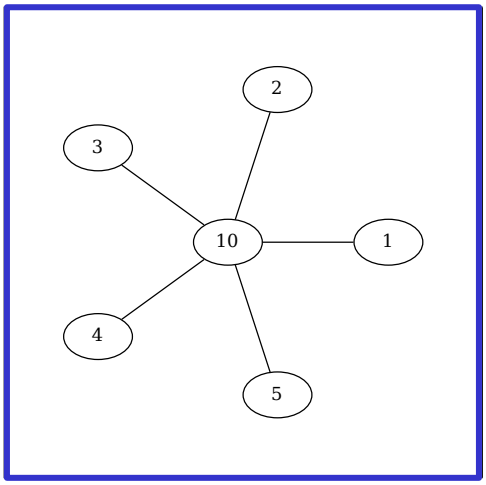
(iii) Arbre Rouge-Noir

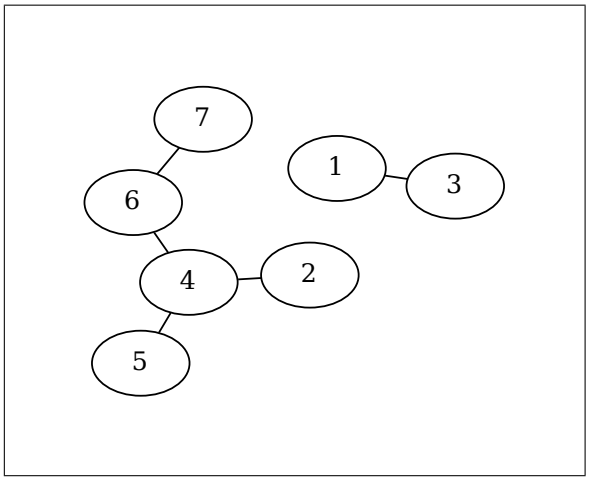
- ☐ $n-1$
- ☐ n
- ☐ $n+1$
- ☐ $\log_2(n)$
- ☐ $\lfloor 1.44\log_2(n+1)-1.328 \rfloor$
- ☒ $\lfloor 2\log_2(n) \rfloor$

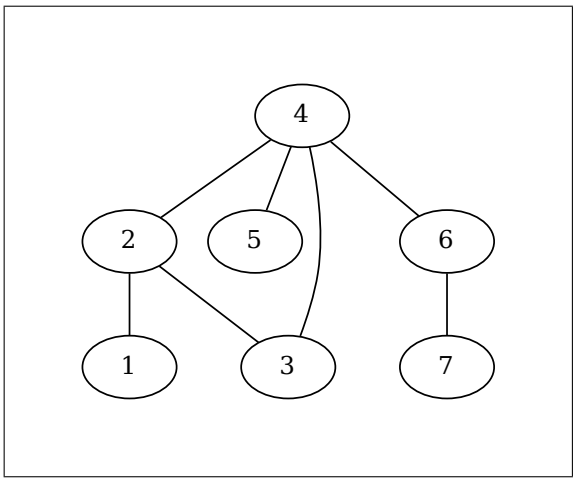
(d) Dans quel contexte un arbre AVL est-t-il meilleur qu'un arbre Rouge-Noir ?

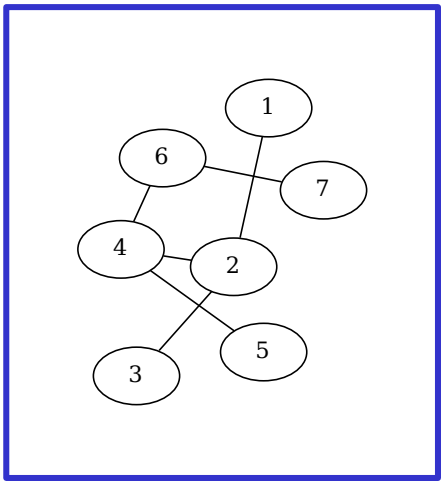
- ☐ Lorsqu'on fait beaucoup d'insertions par rapport au nombre de recherches.
- ☒ Lorsqu'on fait beaucoup de recherches par rapport au nombre d'insertions.
- ☐ Lorsqu'on fait beaucoup de suppressions par rapport au nombre d'insertions.
- ☐ Les arbres AVL sont toujours plus lent que les arbres Rouge-Noir.
- ☐ Les arbres AVL et Rouge-Noir ont la même performance empirique dans tous les cas.

(e) Encerclez, parmi les figures suivantes, celles qui représentent des arbres.



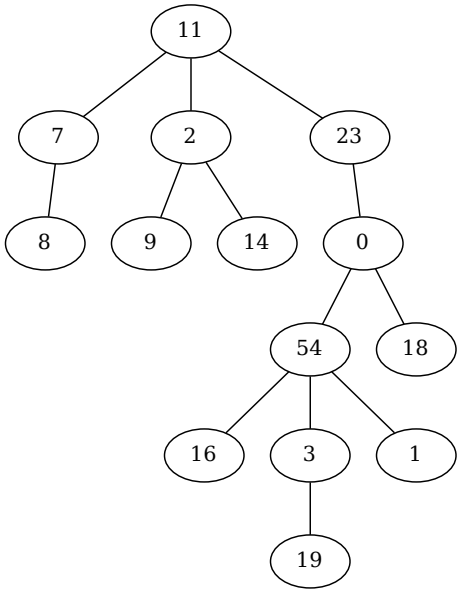






3 Arbres généraux [20 points]

Considérez l'arbre suivant pour répondre aux questions ci-dessous. Chaque question vaut 2 points.



(a) Quelle serait la hauteur de l'arbre si la racine était le nœud 8 (hauteur d'un arbre vide = -1) : 7

(b) Quelle est la profondeur du nœud 1 : 4

(c) Quels sont les ancêtres du nœud 16 : 54, 0, 23, 11

(d) Quels sont les descendants du nœud 23 : 0, 54, 18, 16, 3, 1, 19

(e) Lesquelles des opérations suivantes *devraient* être implémentées à l'aide d'un parcours postordre ? Encerclez toutes les bonnes réponses.

- [x] (i) Calculer la hauteur de l'arbre.
- [ ] (ii) Recherche d'un élément dans l'arbre.
- [x] (iii) Vider l'arbre.
- [ ] (iv) Affichage des éléments en ordre croissant.

Écrivez l'ordre de traitement des nœuds de l'arbre ci-dessus pour chacun des parcours suivants. Si un parcours ne s'applique pas pour l'arbre donné, écrivez simplement "impossible".

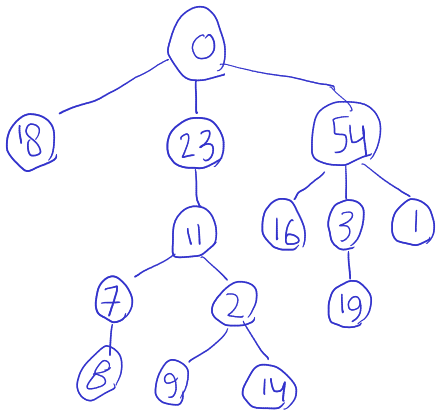
(f) Parcours en largeur : 11, 7, 2, 23, 8, 9, 14, 0, 54, 18, 16, 3, 1, 19

(g) Parcours préordre : 11, 7, 8, 2, 9, 14, 23, 0, 54, 16, 3, 19, 1, 18

(h) Parcours inordre : impossible

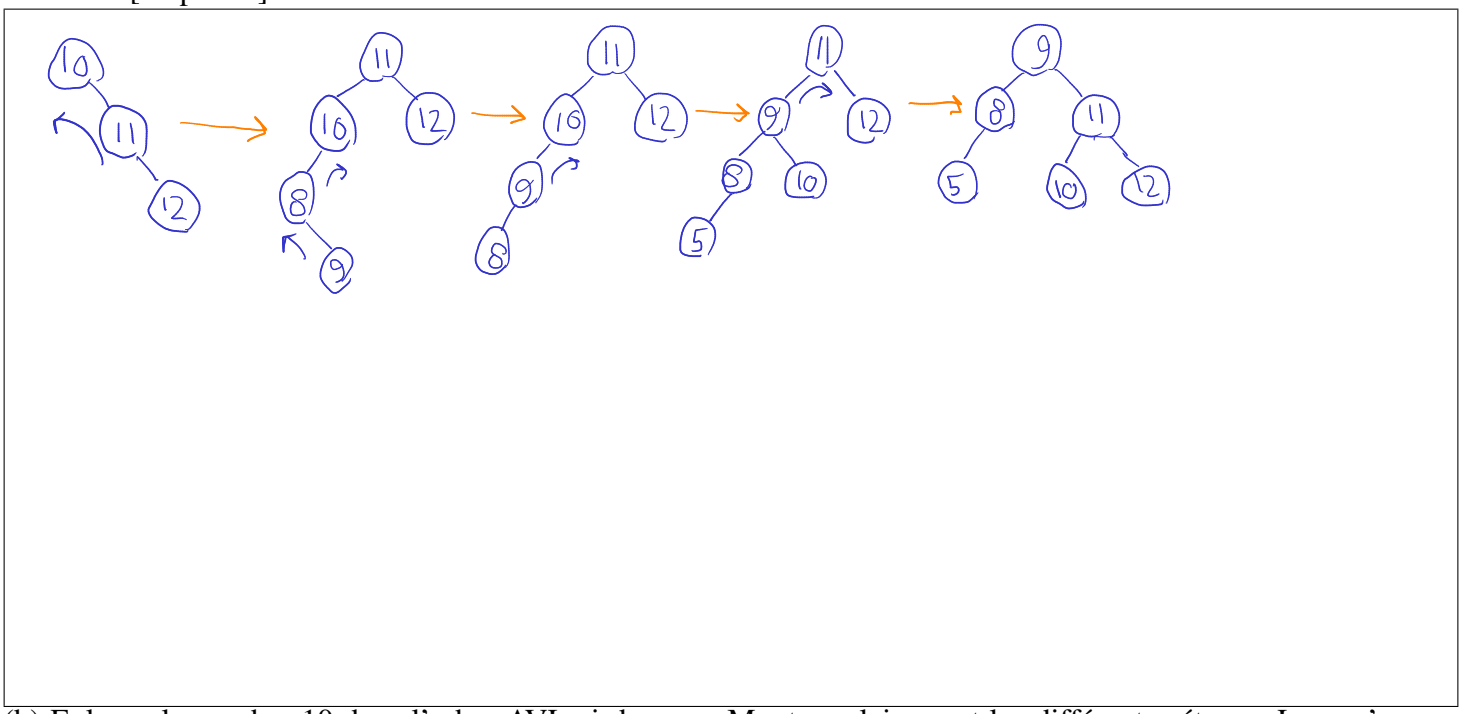
(i) Parcours postordre : 8, 7, 9, 14, 2, 16, 19, 3, 1, 54, 18, 0, 23, 11

(j) Redessiner l'arbre ci-dessus en l'enracinant sur le nœud 0 plutôt que 11.

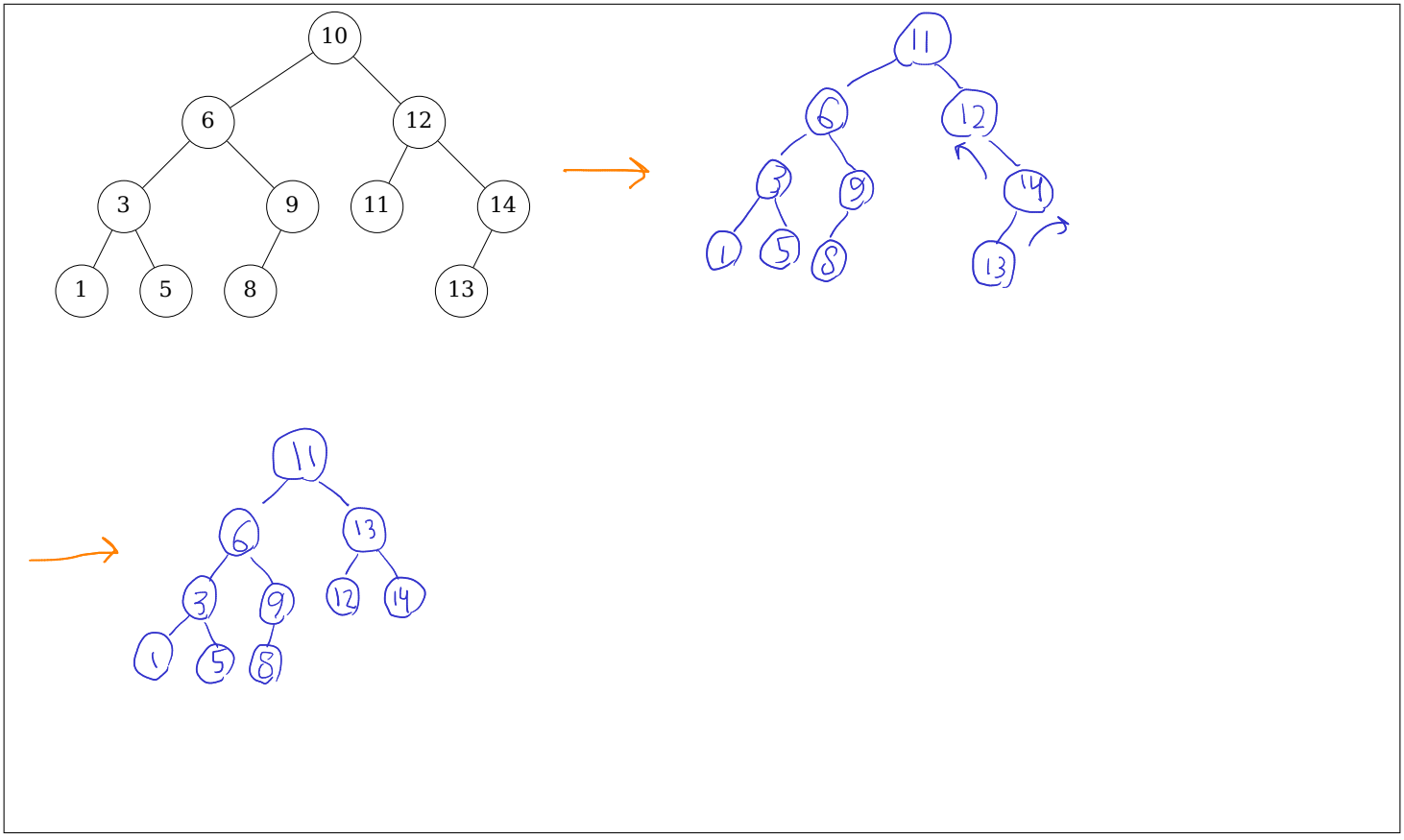


4 Arbres AVL [15 points]

(a) Insérez les nombres $10, 11, 12, 8, 9, 5$ (dans cet ordre) dans un arbre AVL initialement vide. Montrez clairement les différentes étapes. Lorsqu'une rotation est requise, dessinez une flèche et redessinez le nouvel arbre résultant. [10 points]

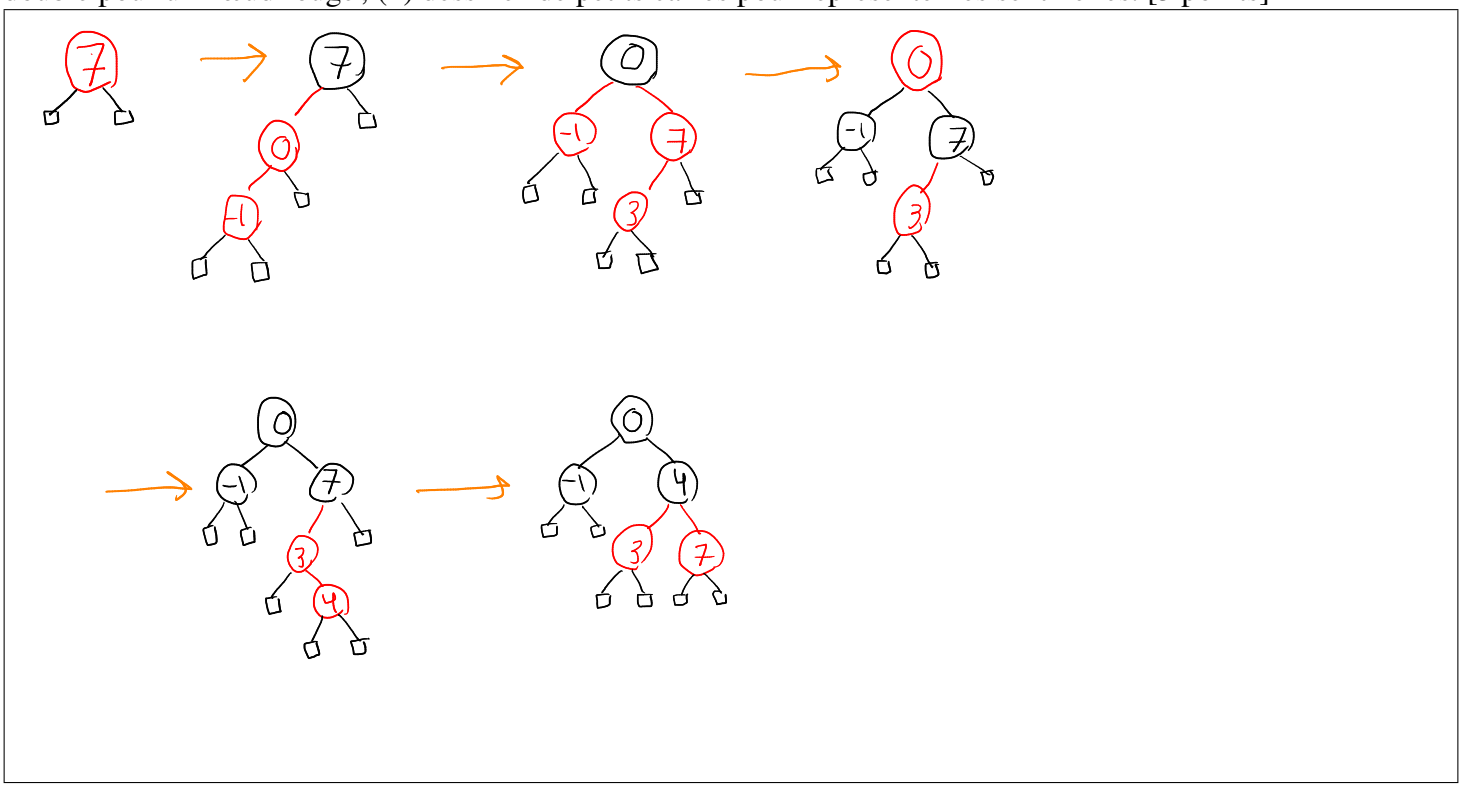


(b) Enlevez le nombre 10 dans l'arbre AVL ci-dessous. Montrez clairement les différentes étapes. Lorsqu'une rotation est requise, dessinez une flèche et redessinez le nouvel arbre résultant. Lorsqu'un nœud ayant deux enfants est supprimé, utilisez le minimum du sous-arbre droit pour le remplacer. [5 points]

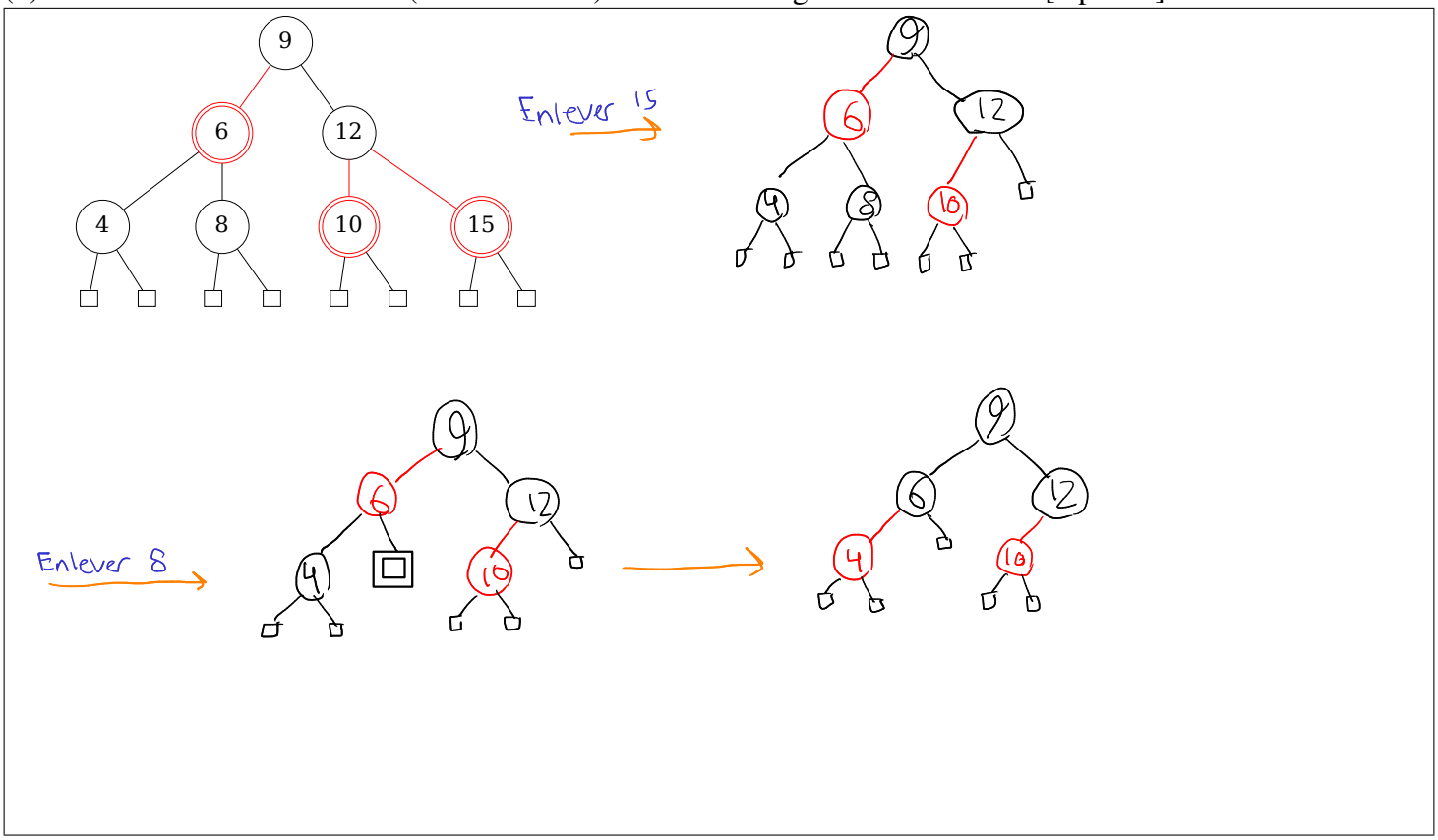


5 Arbres Rouge-Noir [10 points]

Insérez les nombres 7, 0, -1, 3, 4 dans un arbre rouge-noir vide. Lorsqu'une réorganisation/recoloration de nœud(s) est requise, redessinez l'arbre afin de bien montrer les étapes intermédiaires. Considérez les directives suivantes : (1) si vous n'avez pas de crayon rouge, dessinez un cercle simple pour un nœud noir et un cercle double pour un nœud rouge ; (2) dessinez de petits carrés pour représenter les sentinelles. [5 points]



(b) Enlevez les nombres 15 et 8 (dans cet ordre) de l'arbre Rouge-Noir ci-dessous. [5 points]



6 Listes chaînées [10 points]

Vous devez compléter la fonction `inverser` de la classe `Liste` ci-dessous, qui retourne une nouvelle liste contenant les éléments de la liste originale, mais dans l'ordre inverse. Vous devez utiliser la représentation de liste simplement chaînée vu en classe et donnée ci-bas.

```
1  template <class T>
2  class Liste {
3   public:
4    Liste() : premiere(nullptr) {}
5    ~Liste();
6
7    Liste<T> inverser() const;
8   private:
9    struct Cellule {
10     Cellule(const T& v, const Cellule* s) : valeur(v), suivante(s) {}
11     T valeur;
12     Cellule* suivante;
13   };
14   Cellule* premiere;
15 };
16
17 template <class T>
18 Liste<T> Liste<T>::inverser() const {
19   Liste<T> resultat;
20   Cellule* courante = premiere;
21   while (courante != nullptr) {
22     resultat.premiere = new Cellule{courante->valeur, resultat.premiere};
23     courante = courante->suivante;
24   }
25   return resultat;
26 }
```

7 Complexité asymptotique [20 points]

(a) Pour chacune des deux fonctions suivantes, indiquez sa complexité asymptotique. [2 points chacun]

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

void q7a(int n) {
  for(int i = 0; i < n; ++i) {
    for(int j = 0; j < i; ++j) {
      if(j < i)
        continue;

      for(int k = 0; k < n; ++k)
        cout << i << j << k << endl;
    }
  }
}
```

Complexité `q7a` : $\underline{O(n^2)}$

```cpp
#include "tableau.h"

void q7b(int n) {
  Tableau<int> t;
  for(int i = 0; i < n; ++i)
    t.ajouter(i);
  q7b_2(t, n);
}

void q7b_2(Tableau<int>& t, int n) {
  if(n == 0) return;
  q7b_2(t, n - 1);
  t.inserer(n, 0); // insère n à pos. 0
}
```

Complexité `q7b` : $\underline{O(n^2)}$

(b) Simplifiez chacune des six complexités asymptotiques suivantes. [1 point chacun]
Si une des complexités est déjà simplifiée, écrivez "déjà simplifiée".

1. $O(1000n + 0.1n^2)$: $\underline{O(n^2)}$
2. $O(n! + n^n)$: $\underline{O(n^n)}$
3. $O((n^2 - n)^2)$: $\underline{O(n^4)}$
4. $O(n^3 \log(n) + n^4)$: $\underline{O(n^4)}$
5. $O(\log(n^{16}))$: $\underline{O(\log(n))}$
6. $O(50 + 0.001 \log(n))$: $\underline{O(\log(n))}$

(c) Considérez le code suivant. On suppose que les bibliothèques nécessaires sont incluses.

```cpp
struct Etudiant {
  Etudiant(int i, const std::string& n) : id(i), nom(n) {}
  bool operator<(const Etudiant& e) const { return id < e.id; }
  int id; std::string nom; // Les id sont uniques
};

void q7c(int n, int k) {
  Tableau<Etudiant> tab; int id; std::string nom;
  for (int i = 0; i < n; ++i) {
    std::cin >> id >> nom;
    tab.ajouter(Etudiant(id, nom));
    if (i % k == 0) {
      tab.trier(); // Tri Rapide (QuickSort)
      std::cout << tab[0].nom << std::endl;
    }
  }
}
```

(i) Quel est la complexité de la fonction `q7c` ? [2 points] $\frac{n^2}{k}\log(n)$

(ii) Que fait la fonction `q7c` ? [2 points]

À chaque k éléments lus, la fonction affiche le nom de l'étudiant ayant le id le plus petit parmi ceux lus jusqu'à cet instant.

(iii) Écrivez en C++ une fonction `q7c_rapide` affichant la même chose que `q7c`, mais étant asymptotiquement plus rapide. Vous pouvez utilisez toutes les structures de données vues jusqu'ici en cours. [6 points]

```
1  struct Etudiant {
2    Etudiant(int i, const std::string& n) : id(i), nom(n) {}
3    bool operator<(const Etudiant& e) const { return id < e.id; }
4    int id; std::string nom; // Les id sont uniques
5  };
6
7  void q7c(int n, int k) {
8    ArbreMap<int, std::string> d; int id; std::string nom;
9    for (int i = 0; i < n; ++i) {
10     std::cin >> id >> nom;
11     d[id] = nom;
12     if (i % k == 0)
13       std::cout << d.debut().valeur() << std::endl;
14   }
15 }
```

(iv) Quelle est la complexité de votre fonction `q7c_rapide` ? [2 points] $O(n\log n)$

Annexe A

```
1  #include <iostream>
2
3  struct K {
4    K(int n = 3) { z = new char[n]; }
5    char* z;
6  };
7
8  void f(int& a, char*& b) {
9    b[0] = 'A';
10   K* t = new K[2];
11   t[1].z[1] = 'K';
12   t[1].z = b;
13   t[0].z[a++] = 'B';
14   t[0].z[a++] = t[1].z[0];
15   t[0].z[a++] = 'L';
16   b = t[0].z;
17   // <---- Dessin mémoire rendu ici
18   delete[] t;
19 }
20
21 int main() {
22   char cs[5] = {'Z', 'R', 'V'};
23   char* p = &cs[0];
24   int i = 0;
25   f(i, p);
26
27   std::cout << cs[0] << p[0] << cs[1] << '\n';
28   std::cout << p[1] << cs[2] << p[2] << '\n';
29   std::cout << i << std::endl;
30   delete[] p;
31   return 0;
32 }
```

/**** Aide-mémoire sur C++ pour les examens d'INF3105 ****/

1 Types, déclarations, initialisations et opérateurs

Type (taille octets) : `bool` (1), `char` (1), `short` (2), `int` (4), `long` (8), `float` (4), `double` (8), `void*` (8).
Opérateurs arithmétiques : `*`, `/`, `+`, `-`.
Opérateurs logiques : `&&`, `||` (ou).
Opérateurs d'affectation sans/avec opération : `=`, `+=`, `++`, `-=`, `-`, `*=`, `/=`, `,|=`, `,&=`, etc.

```
int a;    // a est non initialisé
int b();  // b est initialisé à 0
int c=2;  // c est initialisé à 2
int d(3); // d est initialisé à 3
a = b++;  // post-incrementation  (a=0, b=1)
a = ++c;  // pre-incrementation   (a=3, c=3)
```

2 Entrées et sorties

```
#include <iostream>
using namespace std; /* std:: est le namespace standard */
int main(){
    int a, b;
    cin >> a >> b; // std::cin >> a >> b;
    std::cout << "La somme est : " << (a + b) << std::endl;
}
```

3 Pointeurs, déférencement, arithmétique des pointeurs et références

- Un pointeur est un objet (une variable) qui contient une adresse mémoire (un entier) de X bits (selon plateforme).
- On peut manipuler un pointeur de façon similaire à un entier (incrémenter, additionner, etc.).
- Le déférencement d'un pointeur doit être explicite à l'aide de l'opérateur de déférencement `*`.
- Une référence est un objet (une variable) qui contient une adresse mémoire, mais se manipule comme un objet.
- Le déférencement d'une référence est implicite. On ne peut manipuler l'adresse mémoire contenu dans une référence.

```
int x; // déclare un entier x
int* p; // déclare un pointeur d'entier p
int* q = &x; // q prend pour valeur l'adresse mémoire de x
p = q; // copie l'adresse mémoire contenu dans q vers p
int y = *q; // accède au contenu pointé par q, c.-à-d. x.
int tab[4]; // tab est un pointeur sur le premier élément d'un bloc de 4 entiers
p = tab + 2; // p = (l'adresse tab) + (2*sizeof(int)). Idem p=&(tab[2]).
p++; // p pointe sur l'entier suivant, c'est-à-dire tab[3]
cout << tab[2]; // équivaut : cout << *(tab+2).
int& r = x; // crée une référence r sur x
r = 3; // affecte 3 à l'entier référé par r, c.-à-d x=3
```

4 Instructions de contrôle d'exécution et blocs d'énoncés

- `if(condition) enonce; [else enonce;]`
- `for(expression_init;expression_condition;expression_increment) enonce;`
- `while(condition) enonce;`
- `do enonce; while(condition);`
- `condition ? expression_sivrai : expression_sifaux;`
- `break` permet d'arrêter et de sortir de la bouche en cours.
- `continue` permet d'arrêter l'itération en cours et d'aller directement à la prochaine itération.
- Les accolades permettent de définir des blocs : `{ enonce1; enonce2; ...; enonceN;}`.

5 Classes, constructeurs, destructeurs, fonctions et appels de fonctions

```
// Déclaration (point.h)
#include <iostream>
using namespace std;
class Point {
 public:
  Point();
  Point(double x_,
        double y_);
  ~Point();

  double dist(const
    Point& p) const;

 private:
  double x, y;

friend istream& operator
    >> (istream& is,
    Point& p);
friend ostream& operator
    << (ostream& os, const
    Point& p);
};
```

```
// Définition (point.cpp)
#include <math.h>
#include "point.h"
Point::Point(){}
Point::Point(float x_, float y_)
 : x(x_), y(y_)
{ /* Alternative: x=x_; y=y_; */ }
Point::~Point(){}

float Point::dist(const Point& p) const {
  float dx = x - p.x, dy = y - p.y;
  return sqrt(dx * dx + dy * dy);
}
istream& operator>>(istream& is,Point& p) {
  char po, vir, pf;
  is >> po >> p.x >> vir >> p.y >> pf;
  return is;
}
ostream& operator<<(ostream& os, const
    Point& p) {
  os << "(" << p.x << "," << p.y << ")";
  return os;
}
```

```
// Utilisation (main.cpp)
#include "point.h"
#include <math.h>

void f1(
Point a // par valeur (copie),
Point* b // par pointeur,
Point& c // par référence,
const Point& d // réf. const
)
{
  float d1 = b->dist(a);
  float d2 = d.dist(c);
  *b = c = d; // p2 = p3 = p4
}

int main() {
  Point p1(0,0), p2(3,4), p3;
  float d = p1.dist(p2);
  Point* p4 = new Point();
  f1(p1, &p2, p3, *p4);
  delete p4;
}
```

Un constructeur (dans l'ordre) :

1. appelle le constructeur de la ou des classes héritées ;
2. appelle le constructeur de chaque variable d'instance ;
3. exécute le code dans le corps du constructeur.

Un destructeur (dans l'ordre) :

1. exécute le code dans le corps du destructeur.
2. appelle le destructeur de chaque variable d'instance ;
3. appelle le destructeur de la ou des classes héritées ;

6 Allocation de la mémoire automatique et dynamique

```
int main(){
  Point p1(0,0), p2(10,10), p3; // alloue automatiquement sur la pile d'exécution
  float d = p1.dist(p2);
  Point* p = new Point();    Point* points = new Point[10];
  delete p;                  delete[] points;
}
```

7 Classes génériques

```
template <class T> class Tableau {
  T*  elements;
  int capacite, nbElements;
public:
  Tableau(int capacite_initiale = 4);
  ~Tableau();
  int taille() const {return nbElements;}
  void ajouter(const T& item);
  T& operator[] (int index);
  const T& operator[] (int index) const;
};
```

```
template<class T> Tableau<T>::Tableau(int ci)
  : capacite(ci), nbElements(0),
    elements(new T[capacite]) {}
template<class T> Tableau<T>::~Tableau() {
  delete[] elements; // elements = nullptr;
}
template <class T>
T& Tableau<T>::operator[] (int index) {
  assert(index < nbElements);
  return elements[index];
}
```