PARIS DESCARTES IUT

DUT Informatique - 1ère année
SDA - Structures de Données et Algorithmes
Equipe pédagogique
Marie-José Caraty, Julien Rossit, Camille Kurtz,
Jacques Alès-Bianchetti, Van Cuong Kieu

DST - 16 janvier 2017

STRUCTURES DE DONNEES ET ALGORITHMES

Durée : 3 heures

Tout document autorisé – Sans calculatrice

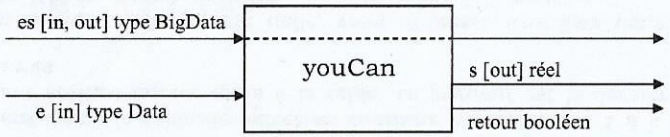
Les parties I et II sont indépendantes et les barèmes donnés sont indicatifs.
Toutes les spécifications données qui ne seront pas suivies seront pénalisées.
La documentation de code sera explicitement demandée dans la question.
La programmation demandée est impérative en Langage C++.
Prenez le temps de lire tout le sujet avant de commencer.

Partie I. Contrôle de connaissance [11 points]

(1)	(2)	(3)	(4)	(5)
1 pt	1 pt	2 pts	3 pts	4 pts

(1) Prototypage

Suivant les bonnes règles du prototypage en C++, prototypez la fonction `youCan` dont l'interface est la suivante :



Spécification : les allocations en mémoire des variables de type `Data` et `BigData` seront considérées de grande taille.

bool youCan (const Data& e, BigData& es, float& s);

(2) Récursivité

La fonction `h` présente un défaut de type contrat concepteur-utilisateur. Modifiez en conséquence la partie de code concernée.

```
int h(int a, int b) {
     if (a==0)
          return b;
     else
          return (h(a-1, a+b));
}
```

Rappels : 1. Une fonction récursive doit avoir un critère d'arrêt. 2. le contrat concepteur-utilisateur doit garantir une bonne utilisation de la fonction. Le domaine de définition de la fonction doit être bien délimité : les moyens pour le concepteur sont un typage adéquat des paramètres formels du prototype associé éventuellement à une/des précondition(s) dont l'objectif est de prévenir l'utilisateur des conditions de bonne utilisation de la fonction.

(3) Schéma mémoire

Soient les deux extraits de programme P1 et P2 suivants :

P1)

```
int *r, *s, a[3];

s = a;
r = a + 1;
a[0] = 20;
a[1] = 21;
a[2] = 22;
r= r+1;

cout << "r=" << *r
     << " s=" << *s;
```

P2)

```
struct Bzz {
     int m, n;
};
Bzz b, c;
b.m = 19; b.n = 37;
c.m = 6; c.n = 92;
Bzz* d;
d = &b;
d->m = 365;
d = &c;
cout << b.m << ' ' << b.n << endl;
cout << d->m << ' ' << d->n ;
```

Dans les deux cas, schématisez l'allocation des variables dans la pile d'exécution et visualisez les pointeurs.

Donnez les traces d'exécution des deux extraits de programme P1 et P2.

(4) Comptine numérique

`m` participants sont assis autour d'une table virtuelle de jeu dans l'ordre horaire de leur numéro d'inscription au jeu (de `1` à `m`). On tire au hasard un entier `k` tel que `0 < k ≤ m` et le participant `p` par lequel le jeu commencera. On commence à compter de `1` à `k` à partir du participant `p` et en tournant dans le sens horaire. Le `k`ème participant quitte la table de jeu et on continue à compter de `1` à `k` à partir du participant suivant encore assis à la table. Le gagnant est le dernier participant restant assis.

Pour un nombre `m` de grande taille, vous utiliserez une liste pour résoudre le problème. Vous trouverez en annexe un composant de Liste (cf. p. 7).

Codez un programme principal implémentant ce jeu.
Spécification : l'utilisateur du programme choisira la valeur de m par sa saisie au clavier. Les valeurs `m`, `k`, `p` et le numéro du gagnant seront affichées à l'écran.

Rappel : Les fonctions à utiliser pour la génération d'un nombre aléatoire

```
srand((unsigned) time(NULL)); // initialise le générateur
rand() % k // renvoie un nombre compris entre 0 et k-1
```

(5) Chaînage

La chaîne considérée est doublement chaînée, la définition des types Chaine et Maillon où Item est le type des données contenues dans la chaîne est la suivante :

```
struct Chaine {
   Maillon* tete;
   Maillon* queue
   unsigned int longueur;
};
```

```
struct Maillon {
   Item data;
   Maillon* suiv;
   Maillon* pred;
};
```

5.1. Faites le schéma d'une chaîne `ch` constituée de quatre maillons successifs d'adresse `a`, `b`, `c` et `d` que vous instancierez respectivement aux adresses `@10`, `@20`, `@30` et `@40`.

5.2. Dans notre exemple, on illustrera le `swap` du maillon `b`. La fonction `swap` permet à partir de l'adresse d'un maillon (dans notre exemple le maillon `b`) d'échanger les deux maillons consécutifs (`b` et `c`) dans la chaîne `ch`. Donnez la précondition de la fonction `swap`.

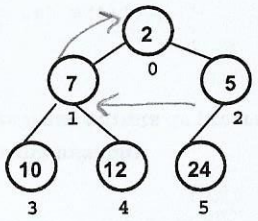
5.3. Dans l'exemple donné, quels sont les maillons que doit modifier la fonction `swap` et quelles sont les modifications à faire ?

5.4. Prototypez, documentez et codez la fonction `swap` d'un maillon `m`.

Partie II. Le tri par tas (heapsort) [9 points]

Un arbre binaire est composé de nœuds, tel que tout nœud père admet un fils gauche et un fils droit. Dans le cas où le dernier nœud père n'a qu'un fils, celui-ci est considéré comme un fils gauche et l'arbre binaire est dit incomplet. Le premier nœud père est appelé la racine. Les nœuds sans fils sont des feuilles. Chaque nœud est valué (valeur inscrite dans le nœud) par une valeur de type quelconque (dans notre cas, un entier).
En numérotant les nœuds de `0` à `n`, un arbre binaire peut être stocké dans un tableau ou une liste en stockant pour chacun des nœuds (de la racine au dernier nœud) : le nœud père, son fils gauche et son fils droit.

L'arbre suivant est un exemple d'arbre binaire incomplet dont chaque nœud est valué par un entier. Le stockage de l'arbre est donné dans la liste `l` :



Numérotation et indice de stockage dans la liste `l` :
a) la racine de l'arbre, nœud numéroté `0` est stockée à l'index `0` de la liste

b) un nœud père n° `i` stocké à la position `i` a son fils gauche stocké en `2*i + 1` et son fils droit `i` stocké en `2*i + 2`

l	2	7	5	10	12	24
	0	1	2	3	4	5

1. La fonction `echanger` vue dans le cours sur les tris est utile au tri par tas.
On rappelle que la fonction `echanger`, à partir de deux indices `i` et `j` dans une liste `l`, échange les valeurs correspondantes dans la liste.
Documentez, prototypez et codez la fonction `echanger`.

2. On étudiera un tri par tas par ordre croissant. Une fonction `tamiser` est introduite, son prototype est le suivant :

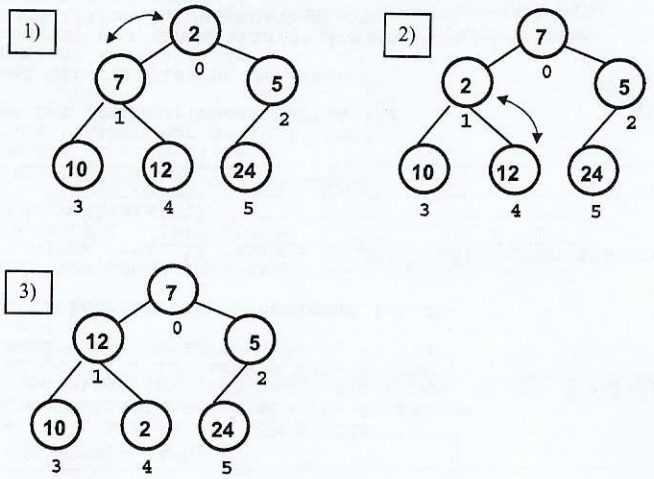
```
void tamiser (Liste& l, int indNoeud, int n);
```

Son rôle est de « tamiser » dans la liste `l` un nœud de numéro `indNoeud` jusqu'au nœud numéro `n`.

L'algorithme de la fonction `tamiser` est le suivant :

```
Naturel nPere <- indNoeud
Naturel fG <- 2*nPere+1
Tant que (fG est inférieur strictement à n) Faire
   fMax <- fils de nPere de plus grande valeur (fG ou fG+1 s'il existe)
   Si (valeur de nPere est inférieure à la valeur de fMax)
       Echanger valeur de nPere avec valeur de fMax
       nPere <-  fMax
       fG <-  2*nPere+1
   fin Si
   Sinon Terminer
fin TantQue
```

L'exemple suivant correspond à l'arbre résultant de l'appel :
`tamiser(l, 0, longueur(l))`



2.1. Documentez et codez la fonction `tamiser`.

3. Codage de la fonction tri par tas `triParTas` de prototype :

```
void triParTas(Liste& l);
```

Pour un tri par ordre croissant, un tas est un arbre tel que tout nœud est de valeur supérieure ou égale à celles de ses fils (voire à son fils si l'arbre est incomplet).
Le tri par tas d'une liste se fait en deux étapes.

Etape 1. Construction du tas à partir de la liste

On commence par tamiser l'arbre correspondant à la liste, de bas en haut et de droite à gauche. Ce tamisage se fait ainsi dans l'ordre suivant des nœuds : du nœud numéro (longueur(l)/2)-1 à 0 par pas de -1, avec pour valeur limite longueur(l).

L'arbre alors obtenu est un tas (tout nœud père a une valeur supérieure ou égale à ses nœuds fils). A la racine se trouve le maximum de la liste.

3.1.1. Donnez l'arbre organisé en tas résultant au traitement de cette étape ainsi que la liste résultante.

3.1.2. Codez cette première étape de la fonction `triParTas`

Etape 2. Tri des éléments de la liste correspondant à un tas

La liste est organisée en tas, le maximum de la liste se trouve à la racine à l'indice 0. L'algorithme de tri est le suivant :

```
   Pour i allant de longueur(l)-1 à 1 par pas de -1 debut_Pour
[A1]   échanger les valeurs de liste entre i et en 0 // i est trié
       //le nœud racine (0) est alors le seul nœud mal organisé en tas
[A2]   tamiser la racine jusqu'au numéro i
   fin_Pour
```

3.2.1. Donnez pour tout `i` de l'indice de boucle : l'arbre résultant de l'action `A1` et de l'action `A2`. Donnez la liste résultant de l'étape 2.

3.2.2. Codez la deuxième étape de la fonction `triParTas`.

3.2.3. Donnez la complexité temporelle et spatiale du `triParTas`.

4. Codez un programme de test du tri par tas appliqué à l'exemple donné. Spécification : Le programme devra afficher la liste correspondant à l'exemple et la liste triée résultant du tri par tas.

```
#ifndef LISTE_H
#define LISTE_H
/**
 * @file Liste.h
 * @brief Liste en mémoire dynamique à capacité paramétrée */
#include "TableauMemDyn.h"

struct Liste {
   TableauMemDyn tab; // tableau mémorisant les éléments de la liste
    unsigned int nb; // nombre d'éléments stockés dans la liste
};

/**
 * @brief Initialiser une liste vide
 * la liste est allouée en mémoire dynamique
 * @see detruire, la liste est à désallouer en fin d'utilisation
 * @param[out] l : la liste à initialiser
 * @param[in] c : capacité (>0) de la liste
 * @pre c>0 */
void initialiser(Liste& l, unsigned int c);
/**
 * @brief Désallouer une liste
 * @see initialiser, la liste a déjà été allouée en mémoire dynamique
 * @param[out] l : la liste */
void detruire(Liste& l);
/**
 * @brief Longueur de liste
 * @param[in] l : la liste
 * @return la longueur de la liste*/
unsigned int longueur(const Liste& l);
/**
 * @brief Lire un élément de liste
 * @param[in] l : la liste
 * @param[in] pos : position de l'élément à lire
 * @return l'item lu en position pos
 * @pre 0<=pos<longueur(l) */
Item lire(const Liste& l, unsigned int pos);
/**
 * @brief Ecrire un item dans la liste
 * @param[in,out] l : la liste
 * @param[in] pos : position de l'élément à écrire
 * @param[in] it : l'item
 * @pre 0<=pos<longueur(l) */
void ecrire(Liste& l, unsigned int pos, const Item& it);
/**
 * @brief Insérer un élément dans une liste
 * @param[in,out] l : la liste
 * @param[in] pos : la position à laquelle l'élément est inséré
 * @param[in] it : l'élément inséré
 * @pre 0<=pos<=longueur(l)
 * l'insertion est faite avant la position pos */
void inserer(Liste& l, unsigned int pos, const Item& it);
/**
 * @brief Supprimer un élément dans une liste
 * @param[in,out] l : la liste
 * @param[in] pos : la position de l'élément à supprimer
 * @pre longueur(l)>0 et 0<=pos<longueur(l) */
void supprimer(Liste& l, unsigned int pos);

#endif /*LISTE_H */
```

Annexe de code – Composant de Liste

UNIVERSITÉ PARIS DESCARTES
IUT
DÉPARTEMENT INFORMATIQUE

NOM : RIFKI (en capitales) Écrire très lisiblement
Prénom : Kenza

DISCIPLINE : SDA
Date de l'épreuve : 16/01/2017
Année : 1ère Groupe : 110

NOTE DE 0 À 20	APPRÉCIATIONS
16/20 mtc	I. 8.5 II 7.5

Ne rien écrire dans cette marge

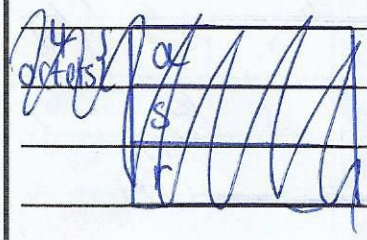
Partie I :

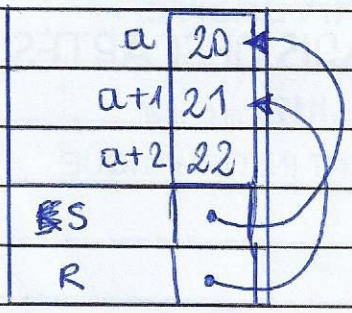
(1) bool youCan(const Data& e, BigData& es, float& s);

(2)
```
int h(unsigned int a, int b) {
    assert(a>=0);
    if (a==0)
        return b;
    else
        return (h(a-1, a+b));
}
```

(3)

P1) Pile :





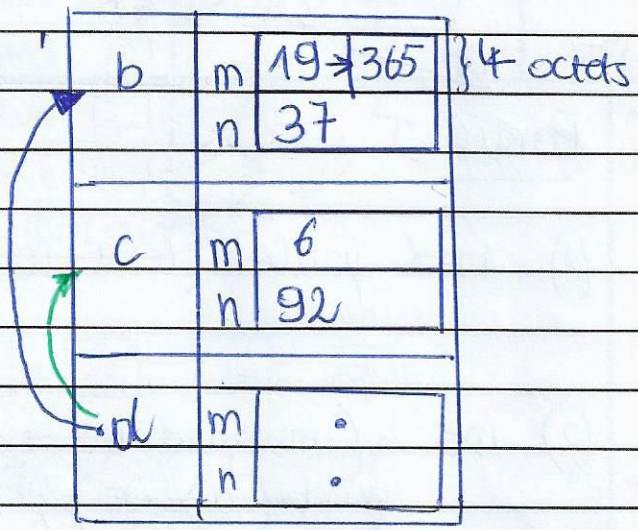
1

Traces d'exécution : S = &a
R = &a[1]
R = &a[2]

Affichage : r = 22
s = 20

TP

P2) Pile :



1

en vert est représenté le pointeur quand d pointe sur c. A ce moment, d ne pointe plus sur b (flèche bleue).

Affichage : 365 37
6 92

TP

(4)

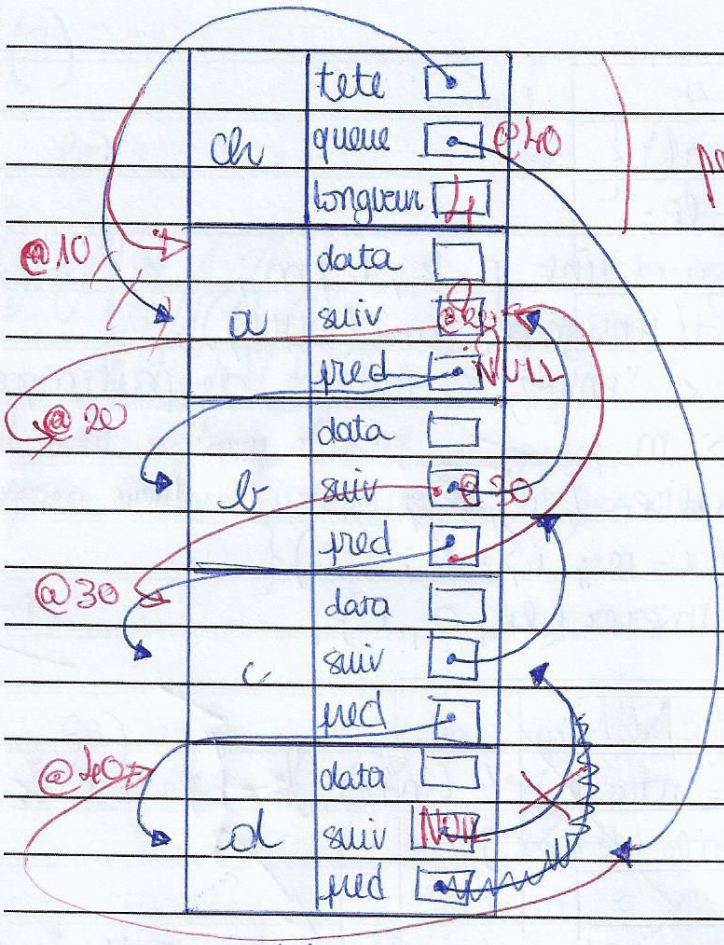
```
#ifndef LISTE_H
#define LISTE_H

#include "liste.h"
#include <iostream>
using namespace std;
```

```
int main(){
  liste li;
  unsigned int p, k, i, m;
  srand((unsigned) time(NULL));
  cout << "Entrez le nombre de participants" << endl;
  cin >> m;
  initialiser(li);
  for(i=m; i>=1; --i){
      inserer(li, 0, i);
  }
  do{
    k = rand() % (m+1);
  }while(k==0);
  do{
    p = rand() % (m+1);
  }while(p==0);
  while(longueur(li) != 1){
      supprimer(li, (p-1)+(k-1));
  }
  cout << lire(li, 0);
  detruire(li);
  system("pause"); return 0;
}
```

3.25

(5)

5.1. voir page suivante.

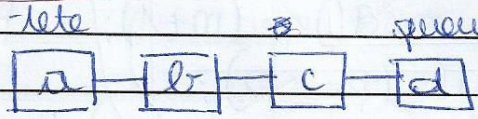


pas spécialement corrigé !

ce n'est pas le chaînage demandé !

0.5

de plus, donnez 0?



5.2. assert (b != c); ? b -> suiv != NULL ;

oui mais dans l'échange on donne juste b

5.3. Dans l'exemple donné, tous les maillons (échange de 2 maillons consécutifs !)
oui subissent des modifications. En effet,
suiv pred du maillon a ne pointera plus sur
b mais sur c, et suiv pred du maillon d
? ne pointera plus sur c, mais sur b.
les champs suiv et pred des maillons b
et c seront également modifiés.

0.25

5.4. /**

* @brief Echange de deux maillons consecutifs d'une chaîne.

UNIVERSITÉ PARIS DESCARTES

IUT

DÉPARTEMENT INFORMATIQUE

Écrire très lisiblement

NOM (en capitales) : RIFKI

Prénom : Kenza

DISCIPLINE : SDA

Date de l'épreuve : 16/01/2017

Année : 1ère Groupe : 110

NOTE DE 0 À 20	APPRÉCIATIONS

Ne rien écrire dans cette marge

0,25

* @ param[in-out] la chaîne ch
* @ param[in-out] le maillon m
* @ pre ? l'adresse du premier maillon de l'échange.
*

0,25

```
void swap(Chaine& ch, Maillon* m) {
    assert (m >= ch.tête && m <= ch.queue);
```

?

Partie II :

1.

```
/**                                   [in, out]
 * @brief echange deux elements d'une liste
 * @param[in] li: la liste
 * @param[in] i: l'indice 1
 * @param[in] j: l'indice 2
 * @pre i>=0 et j>=0 et i<longueur(l)
 *      et j<longueur(l) et i != j
 */

void echanger (liste& li, const unsigned int i,
               const unsigned int j) {
    unsigned int tmp;
    assert(i>=0 && j>=0 && i!=j);
    assert(i<longueur(li) && j<longueur(li));
    tmp = lire(li, i);
    ecrire(li, i, lire(li,j));
    ecrire(li, j, tmp);
}
```

1.5

2.

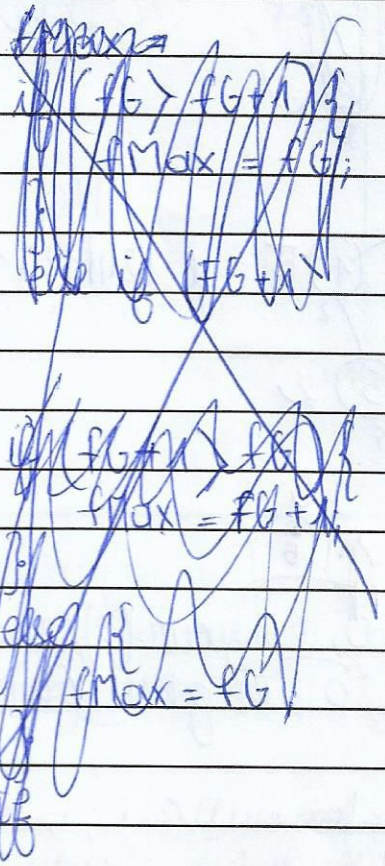
2.1.

```
/**                                   [un noeud dans]
 * @brief tamiser une liste d'un noeud à l'autre
 * @param ? [in,out] la liste
 * @param ? [in] l'index du noeud à tamiser dans la liste
 * @param ? [in] la position de fin de tamisage dans la liste.
 * @pre indNoeud>=0 et n>0 et indNoeud<n
 */

void tamiser (liste& l, int indNoeud, int n){
    assert(indNoeud>=0 && n>0 && indNoeud<n);
    int nPere, fG, fMax;
```

0,25

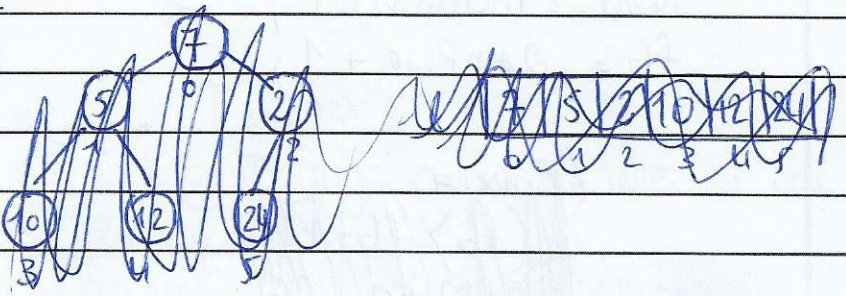
0,25

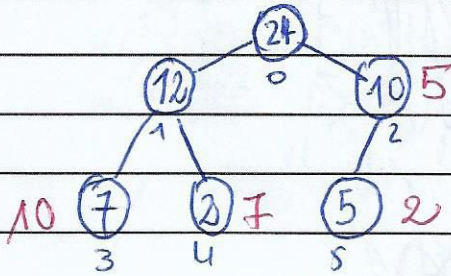
```
nPere = indNoeud ;
fG = 2*nPere + 1 ;
```



```
while (fG < n) {
    if (lire(l, fG+1) > lire(l, fG)) {
        fMax = fG+1 ;
    }
    else {
        fMax = fG ;
    }
    if (lire(l, nPere) < lire(l, fMax)) {
        echanger(l, nPere, fMax);
        nPere = fMax;
        fG = 2*nPere + 1 ;
    }
    else break ;
}
```

3.1.1.

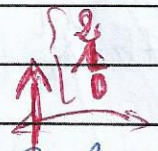




l	24	12	10	7	2	5
	0	1	2	3	4	5
			5	10	7	2

0

3.1.2.



tamiser (l, 0, longueur(l)/2 - 1);
tamiser (l, 0, longueur(l));

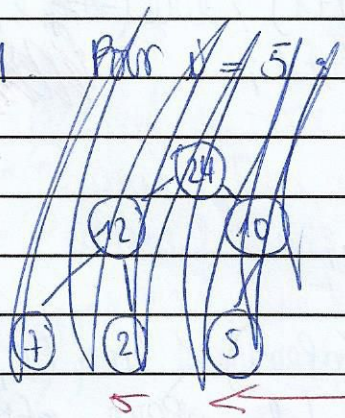
1 tamiser pour i compris entre 2 et 0

1 crée la boucle de tamisage

for (int i = longueur(l)/2 - 1 ; i >= 0 ; --i) // données de l'étape 1
    tamiser (l, i, longueur(l));

3

3.2.1. Pour i = 5 //



l	24	12	10	7	5	2
	0	1	2	3	4	5

0.25 - la liste est triée par ordre croissant.

B0 Il fallait donner tous les autres ou les listes correspondantes. -

UNIVERSITÉ
PARIS DESCARTES
IUT
DÉPARTEMENT INFORMATIQUE

NOM (en capitales) : RIFKI
Prénom : Kenza
Écrire très lisiblement

DISCIPLINE : SDA
Date de l'épreuve : 16/01/2017
Année : 1ère **Groupe** : 110

NOTE DE 0 À 20	APPRÉCIATIONS

Ne rien écrire dans cette marge

3.2.2.

Désolée je reprends votre copie, je me suis trompée de question — $i \geq 0$

```
for (unsigned int i = longueur(l)-1 ; i>=1 ; --i) {
    echanger (l , i , 0) ;
    tamiser (l, 0, i) ;
}
```

TB — 0.75

3.2.3. Complexité spatiale : $1 = \Theta(0)$
Complexité temporelle : $\Theta(n \log_2(n))$

Meilleur cas : $\Theta(\text{longueur}(l))$
Pire des cas : $\Theta(\text{longueur}(l) \times \log_2 \text{longueur}(l))$

TB — 1

4.
```
# include "Liste.h"
# include <iostream>
using namespace std;
# include "main.h"  //admettons que les fonctions codées précédemment se trouvent ici
```

OK

```
int main(){

    Liste l;
    initialiser(l);
    inserer(l,0,24);
    inserer(l,0,12);
    inserer(l,0,10);
    inserer(l,0,5);
    inserer(l,0,7);
    inserer(l,0,2);

    for(unsigned int i=0; i<longueur(l); ++i){
        cout << lire(l,i); // affiche la liste d'exemple
    }

    triParTas(l);

    for(unsigned int i=0; i<longueur(l); ++i){
        cout << lire(l,i); // affiche la liste triée
    }

    system("pause"); return 0;
}
```

Affichage ! et Liste initiale ! Liste triée !