# *Mi-session 2013*
# *Solution*

**Q1:**

- MyCode1(A) : Nombre d'operation est : $n * \sum_0^{200} i \dashrightarrow O(n)$
- MyCode2(A) : Nombre d'operation est : $201 * \sum_0^{n} i \dashrightarrow O(n^2)$

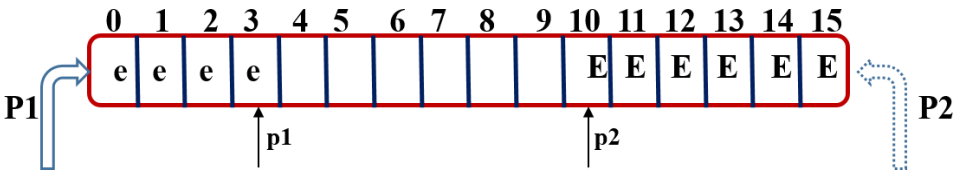
**Q2:**

- $O(n^3)$
- $n * \sum_0^{100} i + \log n \dashrightarrow O(n)$
- $n^2 + \text{n} \log n \dashrightarrow O(n^2)$

**Q3:**

- Les deux algorithmes retournent la position de l'élément K s'il existe dans le tableau a.
- S1 : Nombre de comparaisons effectues :
  - n Comparaisons dans la boucle While (if a[i]>=k)
  - 1 apres la boucle (if a[i-1]=k)
  - Nb total = n+1
  - O(n)
- S2 : Nombre de comparaisons effectues
  - 1 (if a[mid]=k)
  - 1 (if a[mid]>k)
  - n/2 dans la boucle while
  - Nb total = n/2 + 2
  - O(n)

**Q4:**




Example : N = 16

- Size1() {retourn p1+1}

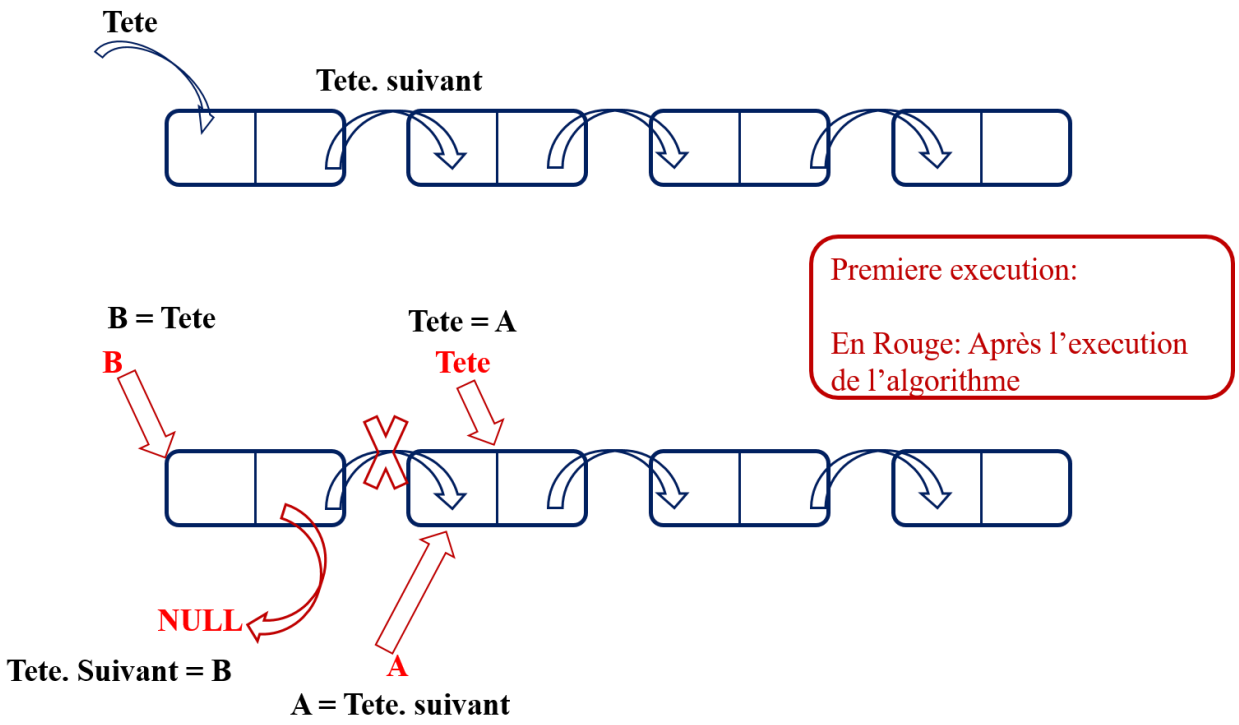
- Size2() {retourn N-p2}
- Estplein() {si p2-p1 =1 retourne true, sinon retourn false}
- Depile1() {si size1() >=1 {retourne A[p1], p1 = p1-1}, sinon retourne « P1 est vide »}
- Depile2() {si size2() >=1 {retourne A[p2], p2 = p2+1}, sinon retourne « P2 est vide »}

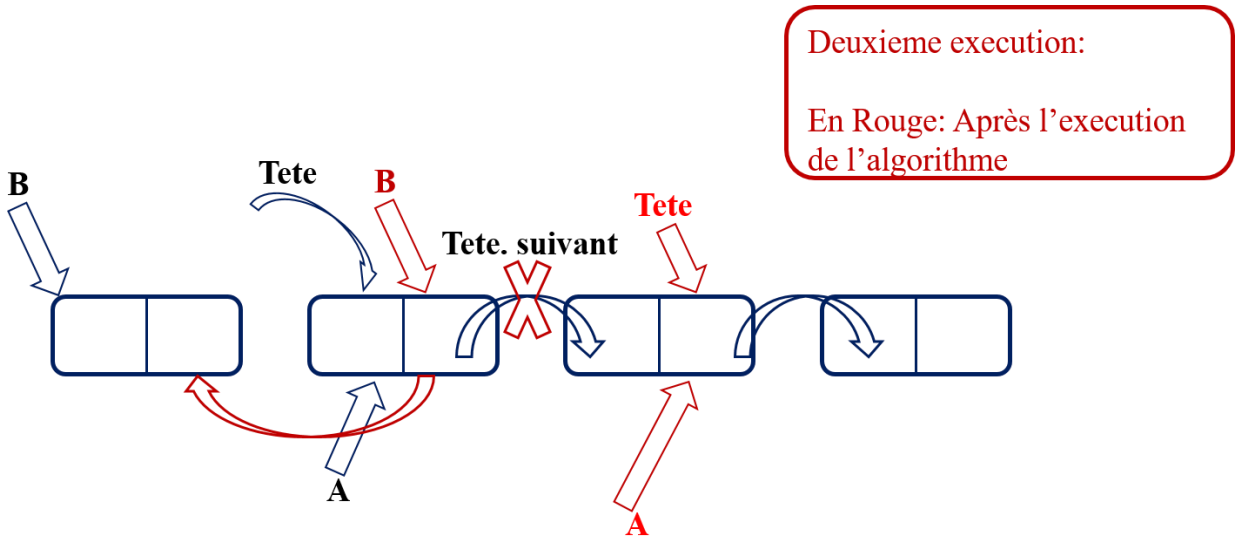
**Q5:**

On va utiliser trois pointeurs :

- tete : pointant au premier nœud de la liste
- A : pointeur intermediaire
- B : pointeur de la liste reversee

```
nœud * reverse (nœud * tete)

{

Nœud * a;

Noeud  * B = NULL;

Tadisque (tete != NULL)

        {

        A = tete.suivant;

        tete.suivant = B;

        B = tete;

        tete = A;

        }

}
```




**Q6:**

- ~~Reference au nœud suivant~~ : Reference au nœud Parent
- ~~Au maximum deux enfants :~~ 0 ou 2 enfants
- ~~$log_2 i$~~ : $2^i$

**Q7:**

- TAD Pile

| **Méthode Pile** | **Methode Dequeu (File a deux bouts)** |
|---|---|
| Size() | Size() |
| isEmpty() | isEmpty() |
| Top() | First (ou Last) |
| Push(e) | InsertFirst (ou InsertLast) |
| Pop() | removeFrist (ou removeLast) |

- TAD File

| **Méthode Pile** | **Methode Dequeu (File a deux bouts)** |
|---|---|
| Size() | Size() |
| isEmpty() | isEmpty() |
| front () | Last (ou First) |
| Enqueue(e) | InsertFirst (ou InsertLast) |
| Dequeue() | removeLast (ou removeFirst) |

**Q8:**

Arbre binaire complet de hauteur h :est une arbre binaire parfait de hauteur h-1 et une ou plusieurs feuilles au niveau h. Dans ce cas N = 29. Soit n et le nombre des nœuds de l'arbre binaire parafait de hauteur h-1 et m le nombre des nœuds restants. m+n = N.

Notre but est de déterminer m.

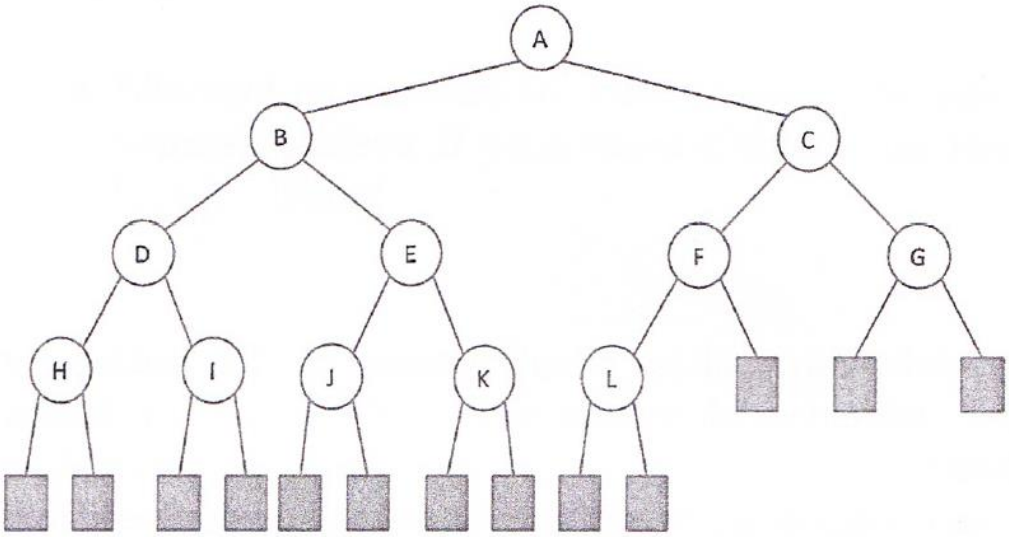
Dans un arbre binaire parfait (de hauteur i) le nombre de nœud est $n = 2^{(i+1)}$-1.

n doit etre plus petit que N donc : $2^{(i+1)} - 1 \leq 29 \Leftrightarrow 2^{(i+1)} \leq 30 \Leftrightarrow i + 1 = 4 \Leftrightarrow i = 3$

Donc

- n = 15
- m = N-n = 29-15 = 14

**Q9:**



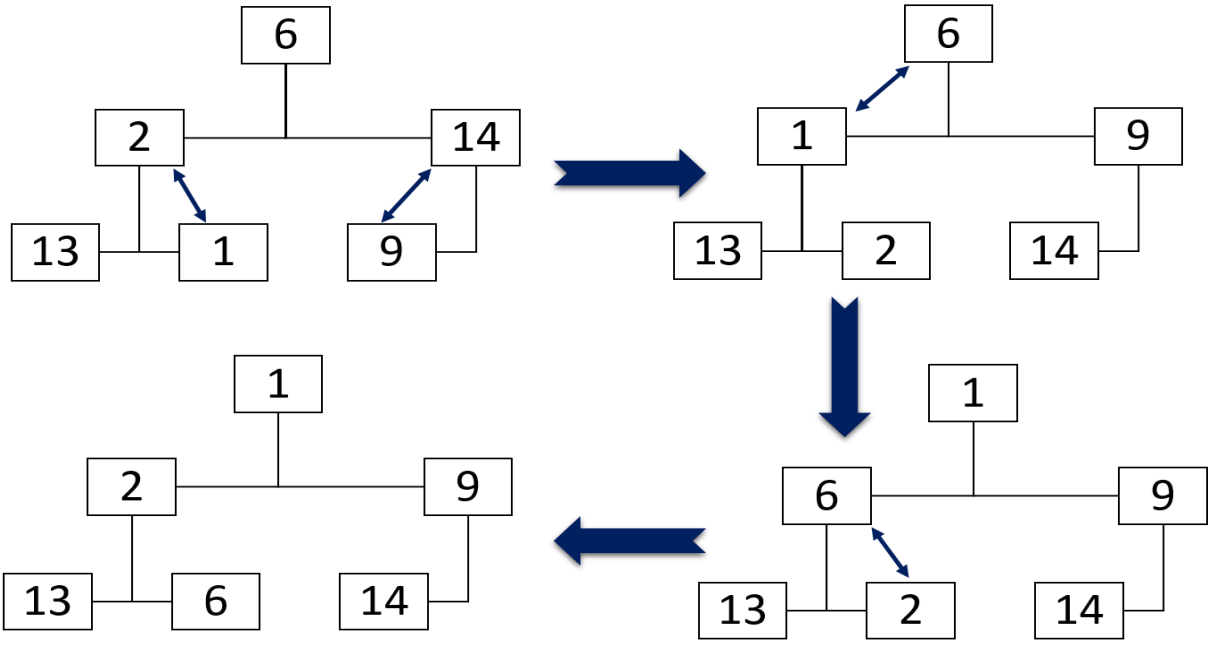

- Parcours pre-ordre (Parent avant enfants) : **Parent - enfant gauche - enfant droite**
  - A B D H I E J K C F L G

- Parcours Post-ordre (Enfants avant parent): **Enfant gauche - Enfant droite -Parent**
  - H I D J K E B L F G C A

- Parcours in-ordre (Enfant - Parent - Enfant): **Enfant gauche - Parent - Enfant droit**
  - H D I B J E K A L F C G

**Q10:**

- Phase 1 : Construction Ascendante




| Index | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Value | 6 | 2 | 14 | 13 | 1 | 9 |

| Index | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Value | 6 | 1 | 9 | 13 | 2 | 14 |

| Index | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Value | 1 | 6 | 9 | 13 | 2 | 14 |

| Index | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Value | 1 | 2 | 9 | 13 | 6 | 14 |

- Phase 2:

Après le 1er retrait

| Index | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Value | ~~1~~ | 2 | 9 | 13 | 6 | 14 |

| Index | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Value | 14 | 2 | 9 | 13 | 6 | |

| Index | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Value | 2 | 14 | 9 | 13 | 6 | |

| Index | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Value | 2 | 6 | 9 | 13 | 14 | |

Après le 2eme retrait

| Index | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Value | ~~2~~ | 6 | 9 | 13 | 14 | |

| Index | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Value | 14 | 6 | 9 | 13 | | |

| Index | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Value | 6 | 14 | 9 | 13 | | |

| Index | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Value | 6 | 13 | 9 | 14 | | |

Apres le retrait du dernier element

| Index | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Value | 6 | 13 | 9 | ~~14~~ | | |

| Index | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Value | 6 | 13 | 9 | | | |

**Q11:**

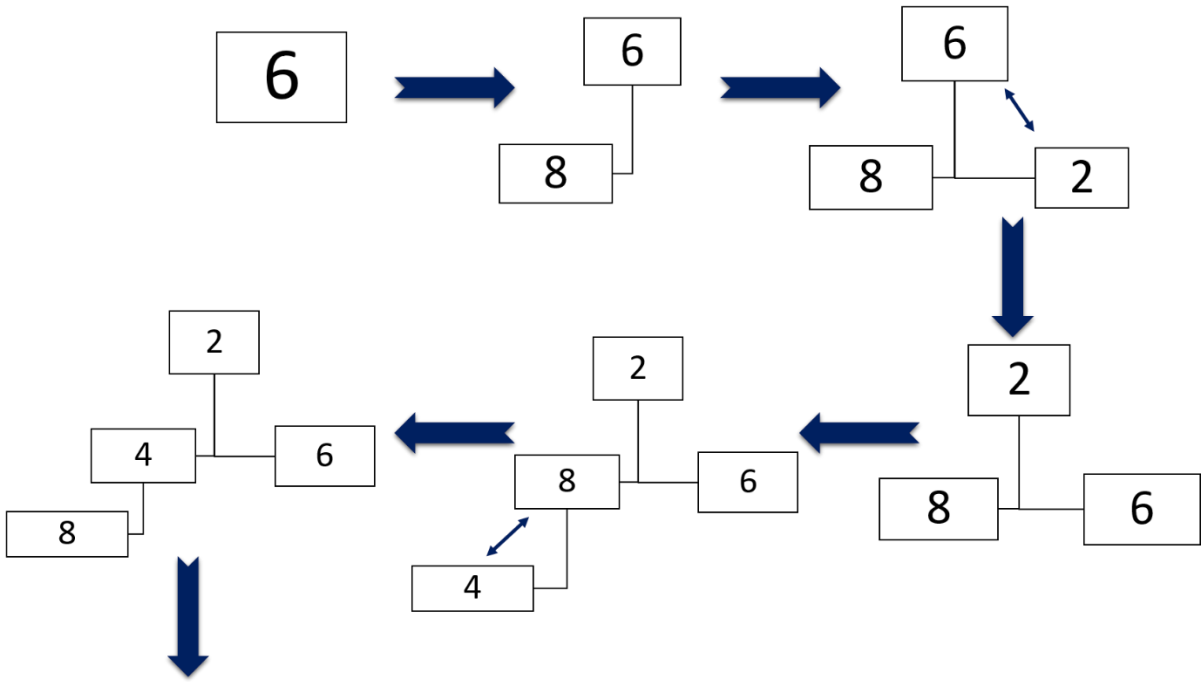
- Faux
- Vrai
- Vrai

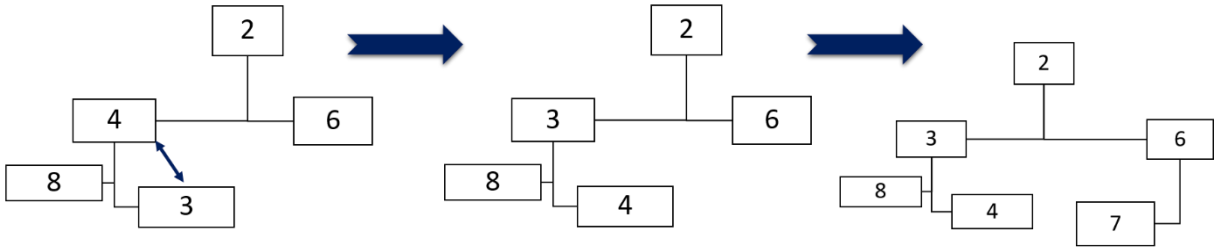
**Q12:**

- Meilleur du cas : racine toute seule : O(1)
- Pire du cas : il y a  n/2 élément qui suit la racine par parcours infixe, donc O(n).

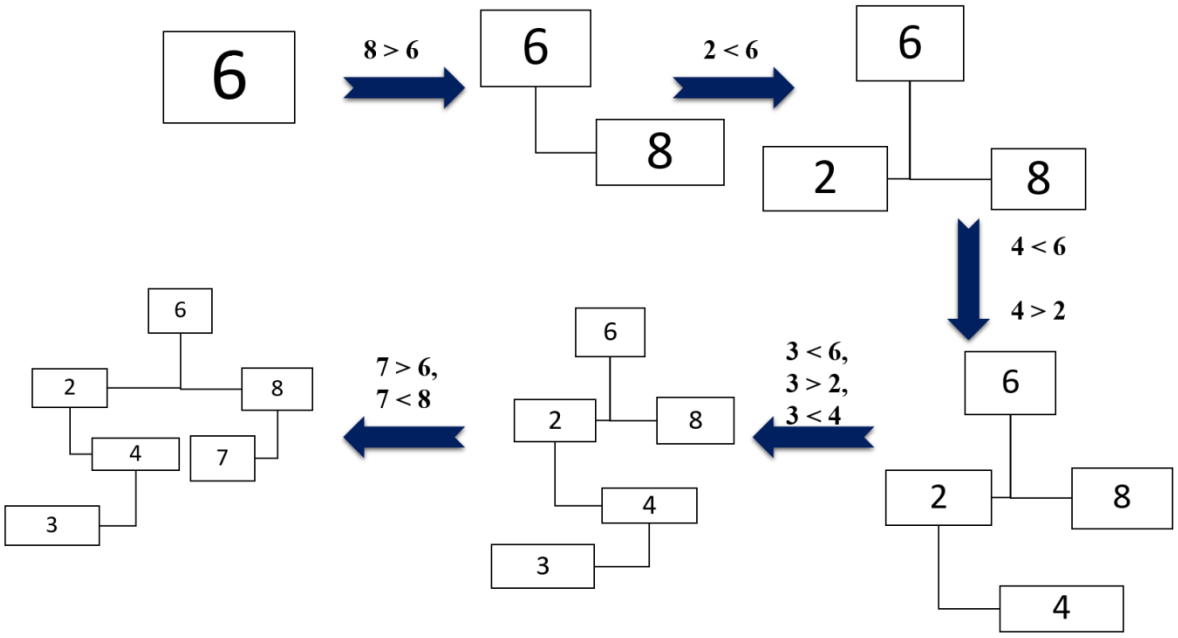
**Q13:**

- Monceau-Min






- Arbre Binaire de Recherche




**Q14:**

- C'est une arbre binaire de recherche. Le nœud 27 est ajoute comme l'enfant droite de 25 puisque 27 > 19 et 27 < 30 et 27 > 25.
- On remplace 10 par 11