1) Recherche trichotomique…

a) Le principe de la recherche trichotomique est le suivant:

La recherche de l'élément `x` se fait sur une liste `λ` dont les `n` éléments sont compris entre deux bornes: une gauche `g` et une droite `d` initialisées respectivement aux valeurs `0` et `n-1`.

Le principe de recherche est récurrent:

Tant que `g+1<d` (au moins deux valeurs d'écart) on calcule deux valeurs de pivots `p1=(2g+d) div 3` et `p2=(g+2d) div 3`. On regarde alors si `x=ieme(λ,p1)` ou si `x=ieme(λ,p2)`.

Si c'est le cas, la recherche est positive et l'on retourne `P1` ou `p2` selon le cas.
Sinon, on poursuit la recherche de `x` sur un intervalle réduit au tiers des éléments.

Dans ce cas, la question est: Quel est cet intervalle?
Les réponses possibles sont les suivantes:

- si `x<ieme(λ,p1)` alors on recommence sur l'intervalle `[g,p1-1]`
- si `ieme(λ,p1)<x<ieme(λ,p2)` alors on recommence sur l'intervalle `[p1+1,p2-1]`
- si `ieme(λ,p2)<x` alors on recommence sur l'intervalle `[P2+1,d]`

Lorsque les bornes `g` et `d` se croisent, la recherche est négative (l'élément `x` n'existe pas dans la liste). Dans ce cas, on retourne `-1`.

Remarque: Nous pourrions continuer la récursion lorsque les bornes ont moins de deux valeurs d'écart, mais nous ferions des tours de plus et des calculs de pivots inutiles. L'idéal est, lorsque les bornes sont égales ou n'ont qu'une valeur d'écart, de tester la valeur de l'élément directement sur ces bornes. C'est cette solution qui est retenue pour l'algorithme et l'arbre d'exécution demandés.

b) L'algorithme:

Cette solution ne tient pas compte de l'aspect débranchant des `Retourne` et positionne à chaque fois les `sinon`. C'est bien évidemment optimisable.

```
Algorithme fonction Trichotomie : Entier
Paramètres locaux
    Element x
    Liste   l
    Entier  g,d
Variables
    Entier  p1,p2
Début
  Si (g+1<d) alors
     p1 ←(2*g+d) div 3
     p2 ←(g+2*d) div 3
     Si (x=ieme(l,p1) alors
        Retourne p1              /* Recherche positive sur p1 */
     Sinon
        Si (x=ieme(l,p2)) alors
           Retourne p2           /* Recherche positive sur p2 */
        Sinon
```

```
            Si (x<ieme(l,p1)) alors
               Retourne Trichotomie(x,l,g,p1-1)
            Sinon
               Si (x<ieme(l,p2)) alors
                  Retourne Trichotomie(x,l,p1+1,p2-1)
               Sinon
                  Retourne Trichotomie(x,l,p2+1,d)
               Finsi
            Fin si
         Fin si
      Fin si
   Sinon
      Si (x=ieme(l,g)) alors
         Retourne g           /* Recherche positive sur g ou d */
      Sinon
         Si (x=ieme(l,d) alors
            Retourne d        /* Recherche positive sur d */
         Sinon
            Retourne 0        /* Recherche négative */
         Fin si
      Fin si
   Fin si
Fin algorithme fonction Trichotomie
```

c) L'arbre d'évaluation de la recherche d'un élément `x` dans une liste λ de `20` éléments est celui de la figure 1.

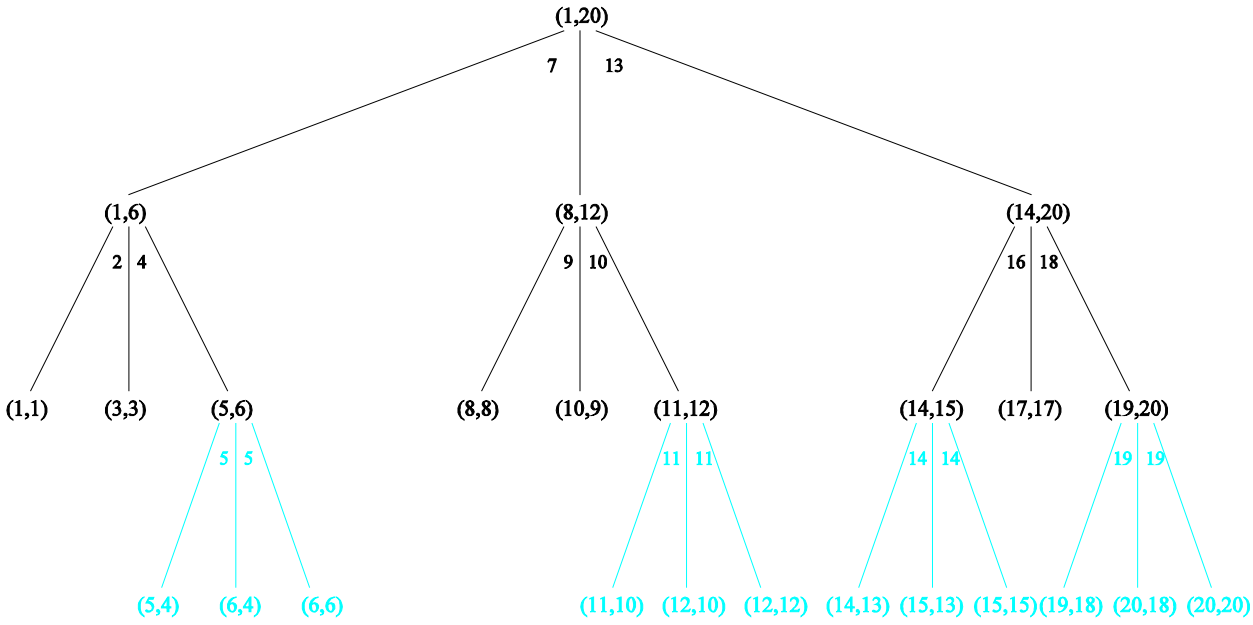

Figure 1 -Exécution d'une trichotomie sur 20 éléments

<u>Remarque:</u> Nous avons laissé la trichotomie visible lorsqu'il n'y a qu'une valeur d'écart entre g et d. Ceci afin de posséder toutes les valeurs de pivots. Mais notre algorithme s'arrête au niveau supérieur et teste les deux valeurs de borne.

d) $O(Log_3^n)$

e) Dans la mesure où la trichotomie offre trois choix à chaque évaluation, Le nombre d'éléments n de la liste est majoré par 3^x où x est le nombre de récursions nécessaires pour atteindre n'importe quel élément. Donc $x = O(\log_3^n)$

2) Liste bitonique…

Spécifications:
La fonction `bitonique(l)` teste si la liste `l` est bitonique et retourne l'index de son point culminant le cas échéant, la valeur -1 sinon.

Principe:
Si la liste n'est pas vide :

- A partir du premier élément, on avance dans la liste tant qu'on n'est pas arrivé au dernier élément et qu'elle est croissante (l'élément courant est inférieur ou égal à son suivant)
- On conserve l'index sur l'élément courant (celui sur lequel s'est arrêté le premier parcours): c'est le point culminant potentiel.
- A partir de là, on continue de parcourir la liste tant qu'elle est décroissante ou jusqu'à arriver au dernier élément.
- Si on est arrivé à la fin de la liste (dernier élément), alors la liste est bitonique.

```
algorithme fonction bitonique: entier
paramètres locaux
        liste l
variables
        entier pos,culm
debut
   si l= listevide alors
        retourne -1
   sinon
        pos ← 0
        tant que (pos <longueur(liste)) et (ième(l, pos) <=
ième(l, pos+1) faire
            pos ← pos+1
        fin tant que
        culm ← pos
        tant que (pos < longueur(liste)) et (ième(l, pos) >=
ième(l, pos+1)) \faire
            pos ← pos +1
        fin tant que
        si pos = longueur(l) -1 alors
            retourne culm
        sinon
            retourne -1
        fin si
    fin si
fin algorithme fonction bitonique
```

3) Graphes…

a) Le graphe est celui de la figure 2.

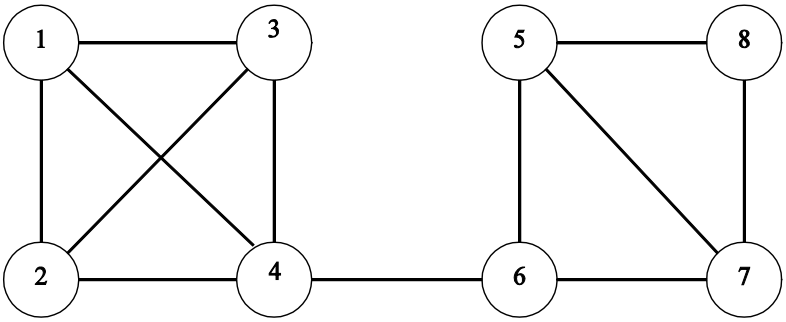

Figure 2 - Graphe non orienté G

b) Non il n'est pas complet. Oui il est connexe.

c) La séquence des sommets de `G` obtenus lors du parcours profondeur du graphe `G` est:
`1,2,3,4,6,5,7,8`

d) La séquence des sommets de `G` obtenus lors du parcours largeur du graphe `G` est:
`1,2,3,4,6,5,7,8`