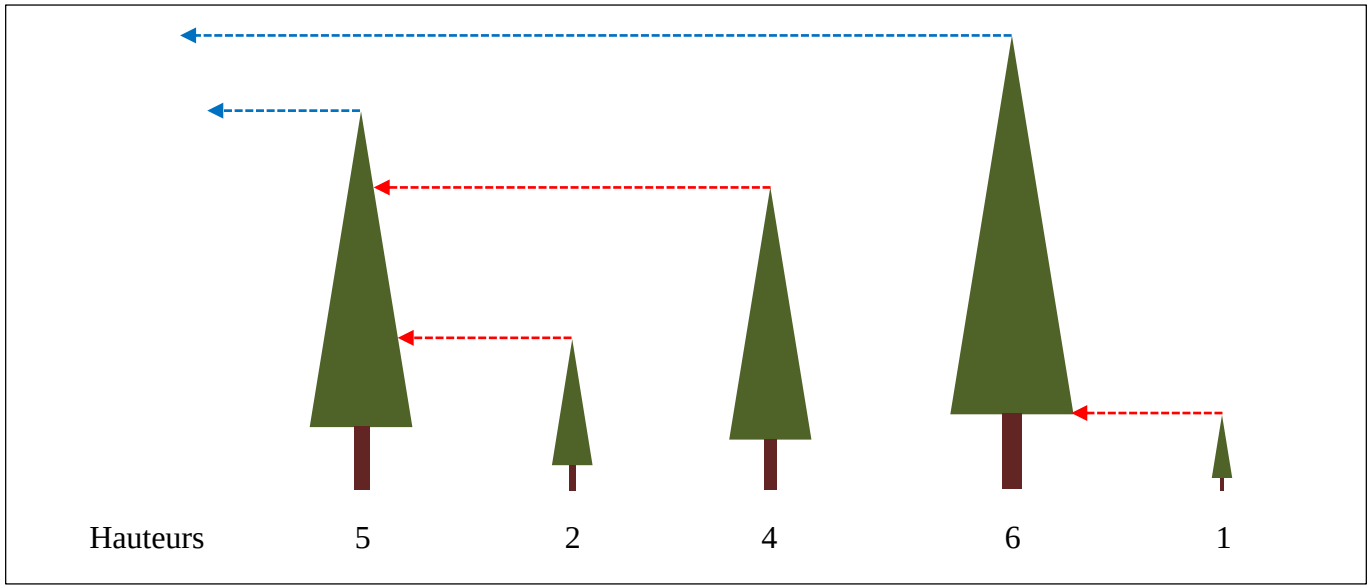
Concours de programmation 2022-2023

EPFC

L'arbre qui cache la forêt

Le problème :

Retrouver, pour chaque arbre aligné, celui qui lui cache la vue.

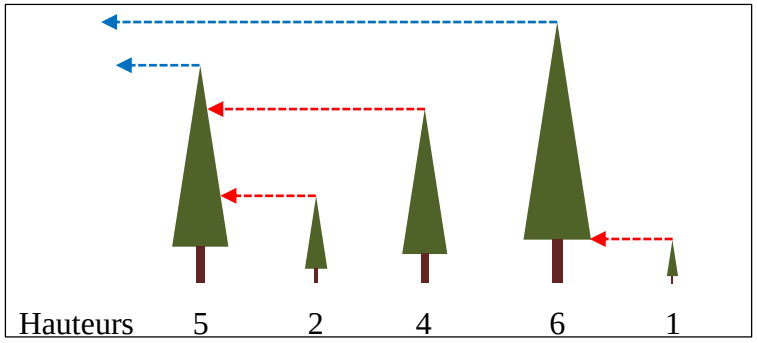


Algorithme (très) simple

On cherche le premier arbre qu'on rencontre (en revenant en arrière) plus grand que l'arbre courant:

Recherche séquentielle (cf. Prm2)

```
public static int hide(int i, int [] tab) {
  // -1 si pas trouvé
  int j = i - 1;
  while(j >= 0 && tab[j] <= tab[i])
    --j;
  return j;
}
```



Algorithme (très) simple

Pour chaque arbre ! Boucle for

Et on demande le numéro de l'arbre et pas son indice $\rightarrow$ +1

```
public static int[] align(int[] tab) {
  int[] res = new int[tab.length];
  for(int i = 0; i < tab.length; ++i)
    res[i] = hide(i, tab) + 1;
  return res;
}
```

Algorithme (très) simple

- Il faut comparer chaque arbre avec N (/2) autres arbres.
- Donc, ~ N x N comparaison !
- Par exemple pour N = 1.000.000 il faudra jusqu'à mille milliard de comparaisons

Temps ≈ 15 minutes

Et pour 10.000.000 d'arbres ce sera 25h !

Mieux ?

On peut facilement trouver une solution qui utilise un *TreeMap* et fait *N x log N* comparaisons d'arbres

Pour N = 1.000.000 il ne faudra que de l'ordre de 20 millions de comparaisons.

50.000 fois plus rapide !

Mais qu'en « pense » *ChatGPT* ?

```java
public static int[] align(int[] tab) {
  Stack<Integer> st = new Stack<>(); // Pile des indices
  int[] res = new int[tab.length];
  for(int i = 0; i < tab.length; ++i) {
    while(!st.isEmpty() && tab[st.peek()] <= tab[i])
      st.pop();
    if(!st.isEmpty())
      res[i] = st.peek() + 1;
    st.push(i);
  }
  return res;
}
```

Guidé par Benoît Penelle ☺

Qu'en « pense » *ChatGPT* ?

```
while(!st.isEmpty() && tab[st.peek()] <= tab[i])
  st.pop();
if(!st.isEmpty())
  res[i] = st.peek() + 1;
```

Qu'en « pense » *ChatGPT* ?

```
for(int i = 0; i < tab.length; ++i) {
  while(!st.isEmpty() && tab[st.peek()] <= tab[i])
    st.pop();
  if(!st.isEmpty())
    res[i] = st.peek() + 1;
  st.push(i);
}
```

- Nécessite un supplément de mémoire (Stack)
- La boucle *while* laisse penser que pour chaque arbre on risque de boucler (comme la 1re sol) !?
- Mais chaque indice n'est ajouté et retiré qu'une seule fois de la pile ! Le *while* ne compte pas !

Temps ≈ 1 sec

Et qu'en pense Olivier Pirson ?

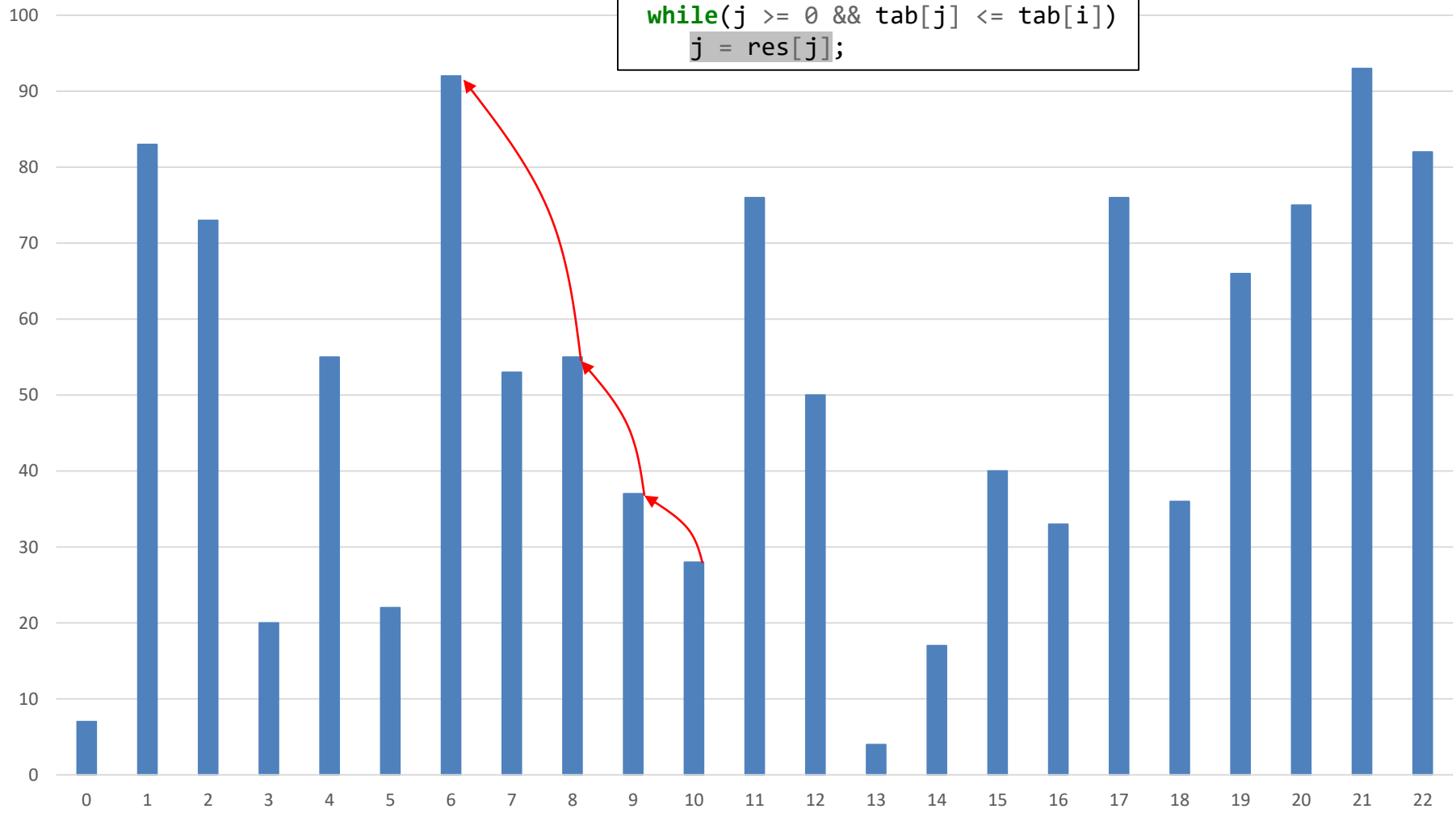
G→D : on dispose des réponses précédentes

Inutile de regarder si plus petits que la réponse d'un précédent

```
public static int hide(int i, int [] tab, int[] res) {
  // Indice du précédent qui cache l'arbre i, -1 sinon
  // res[0:i] contient la réponse pour ceux qui précèdent i
  int j = i - 1;
  while(j >= 0 && tab[j] <= tab[i])
    j = res[j];
  return j;
}
```

Et qu'en pense Olivier Pirson ?

```
while(j >= 0 && tab[j] <= tab[i])
    j = res[j];
```



Et qu'en pense Olivier Pirson ?

```java
public static int[] align(int[] tab) {
  int[] res = new int[tab.length];
  res[0] = -1;
  for(int i = 1; i < tab.length; ++i)
    res[i] = hide(i, tab, res);
  // Ajouter 1 partout
  return Arrays.stream(res).map(x -> x + 1).toArray();
}
```

- Pas de mémoire supplémentaire

Temps ≈ 0,1 sec

Clairement Olivier pense mieux que *ChatGPT* ☺