TD n°12 – Recherche dans un tableau trié

1. *Implémenter en Ocaml une fonction* `recherche : int array -> int -> int -> int -> int`.

```
let recherche tab x g d =
  let gg = ref g in
  let dd = ref d in
  while !gg+1<!dd do
    let  m = (!gg+!dd)/2 in
    if tab.(m) >= x then dd:=m
    else gg:=m
  done;
  if t.(!dd)=x then !dd
  else -1
```

2. *Quelle valeur donner à g et d pour rechercher dans le tableau entier ?*
   $g = -1$, d vaut la longueur du tableau moins 1.
3. *Justifier soigneusement la terminaison de cette fonction.*
   m est le milieu entre gg et dd. Dans la boucle, on sait que $gg + 1 < dd$, donc m n'est ni gg, ni dd. De plus, comme c'est un milieu, on a $gg < m < dd$.
   À chaque tour de boucle, on a soit gg qui prend la valeur m, soit dd qui prend la valeur m. Dans les deux cas, $dd - gg$ diminue d'au moins 1.
   $dd - gg$ est également un entier et la condition de fin de la boucle nous assure que cette quantité reste positive.
   En conclusion, $dd - gg$ est un variant de la boucle, donc la boucle finit et la fonction aussi.
4. *Démontrer que la propriété suivante est un invariant : "Si x apparait dans le tableau, alors il apparait pour la première fois entre les indices $g + 1$ et d".*
   On le montre par récurrence sur les appels.
   Au début de l'algorithme, en considérant qu'on nous a bien donné $g = -1$ et $d = n - 1$, la propriété est évidente.
   Supposons la propriété vraie au début d'une itération. On fait une disjonction de cas :
   - Si $tab.(m) \geq x$, alors, puisque le tableau est trié, on sait que tous les éléments à droite de m sont plus grands que x. Ainsi si x est dans le tableau, c'est à gauche de m (inclus, car potentiellement x se trouve en m). Or le prochain appel est bien fait sur la partie du sous-tableau siuté à gauche de m.
   - Pour l'autre cas la situation est symétrique.

   La propriété est bien un invariant.
5. *Conclure quant à la correction de la fonction* `recherche` *écrite.*
   À la fin de la boucle, on a $gg + 1 = d$, donc le tableau est réduit à 1 élément. L'invariant indique que si x est dans le tableau, il est en dd. La fonction vérifie $tab.(dd)$. Si c'est x, elle renvoie dd, qui est une bonne réponse. Sinon, elle renvoie -1, qui est une bonne réponse puisque x ne peut être ailleurs qu'en dd et n'est pas en dd.

Pour étudier la complexité, on va plutôt regarder la version récursive suivante, qui fonctionne selon le même principe :

```
let rec recherche_rec t x g d =
 if g+1=d then
   if t.(d) = x then d
   else -1
 else let m = (g+d)/2 in
      if t.(m)>=x then recherche_rec t x m d
      else recherche_rec t x g m;;
```

6. *De quoi va dépendre la complexité de cette fonction ?*
   De la taille du sous-tableau étudié, soit $d - g$.
7. *On note $n = d - g$ la taille du sous tableau entre les indices* **g+1** *et* **d**. *Soit $m = \dfrac{g+d}{2}$ le milieu entre* **g** *et* **d**. *Quelle est la taille du sous-tableau d'indices $]g, m]$? Quelle est la taille du sous-tableau d'indices $]m, d]$?*
   On a $m = \lfloor \dfrac{g+d}{2} \rfloor$.
   Le sous-tableau d'indices $]g, m]$ est de taille $m - g$ (car g est exclus).
   On fait une disjonction de cas selon la parité de $g + d$, qui est aussi la parité de $d - g$.
   - Si $g + d$ est pair, $m - g = \dfrac{g+d}{2} - g = \dfrac{g+d-2g}{2} = \dfrac{d-g}{2} = \dfrac{n}{2} = \lfloor \dfrac{n}{2} \rfloor$.
   - Si $g + d$ est impair, $m - g = \dfrac{g+d-1}{2} - g = \dfrac{g+d-2g-1}{2} = \dfrac{d-g-1}{2} = \lfloor \dfrac{n}{2} \rfloor$.

Le sous-tableau d'indices $]m, d]$ est de taille $d - m$.
On fait une disjonction de cas selon la parité de $g + d$, qui est aussi la parité de $d - g$.

- Si $g + d$ est pair, $d - m = d - \frac{g+d}{2} = \frac{2d-g-d}{2} = \frac{d-g}{2} = \frac{n}{2} = \lceil \frac{n}{2} \rceil$.
- Si $g + d$ est impair, $d - m = d - \frac{g+d-1}{2} = \frac{2d-g-d+1}{2} = \frac{d-g+1}{2} = \lceil \frac{n}{2} \rceil$.

8. *En déduire la formule de récurrence de la complexité de la fonction* `recherche`.
On a finalement :

$$C(1) = 1$$
$$C(n) = C(\lfloor \frac{n}{2} \rfloor) + 1 \text{ si } t.(m) \geq x$$
$$C(n) = C(\lceil \frac{n}{2} \rceil) + 1 \text{ sinon}$$

On suppose que la taille n du tableau initial est une puissance de 2.

9. *Trouver la complexité de la fonction sur un tableau de taille* $n = 2^k$.
Si n est une puissance de 2, alors les deux cas de la formule précédente s'unissent en un seul :

$$C(1) = 1$$
$$C(n) = C(\frac{n}{2}) + 1$$

Montrons par récurrence sur $l \in [|0, k|]$, $C(n) = C(n/2^l) + l$.
Pour $l = 0$, on a bien $C(n) = C(n)$.
Supposons que la formule est vraie pour un certain l : $C(n) = C(n/2^l) + l$.
Alors en appliquant la formule de récurrence de la suite C sur $C(n/2^l)$, on obtient $C(n/2^l) = C(n/2^{l+1}) = 1$.
Finalement, $C(n) = C(n/2^{l+1}) + l + 1$.
Par principe de récurrence, la formule est vraie pour tout $l \in [|0, k|]$ et en particulier en $l = k$ on a : $C(n) = C(1) + k$.
Or $n = 2^k$ implique $k = log_2(n)$, donc $C(n) = 1 + log_2(n)$.

Il faudrait revenir à n qui n'est pas une puissance de 2, mais à priori la recherche dichotomique dans un tableau a une complexité logarithmique en la taille du tableau.