

Exercice 1 (Commandes et système de fichiers) :

1. (a) `ls -l donnees.txt`
 (b) `chmod -w donnees.txt`
 (c) `rm ../../Autres/vieux_document.pdf`
 (d) `cd ../../MP2I/Physique/`
2. (a) `mkdir Data/`
 (b) `cp ../../Documents/Point_de_depart/donnees.txt Data/donnees_bis.txt`
 (c) `chmod +w Data/donnees_bis.txt`
 (d) `cd ../Info/Concours_Blanc/`
3. (a) `mv TP_exemple/tp.ml tp.ml`
 (b) `ln -s ../../Physique/Data/donnees_bis.txt donnees.txt`

Exercice 2 (Algorithme de tri) :

1. Dans le **while**, on aura toujours $0 \leq !i < n$, donc les `t.(!i)` ne causeront jamais une erreur d'indice. De plus, si on est dans la partie droite du `||` (par évaluation paresseuse) ou dans le **else**, on a $!i > 0$, donc les `t.(!i-1)` ne causeront jamais une erreur d'indice.
2.
 - Avant la boucle, on a $i = 0$ et $i_{\max} = 0$, donc les deux sous-tableaux $t[0:i]$ et $t[i:i_{\max}]$ sont vides, donc l'invariant est trivialement vérifié.
 - Notons t, i et i_{\max} les valeurs en haut du corps de la boucle, et t', i' et i'_{\max} les valeurs obtenues après un passage supplémentaire dans la boucle.
 - Montrons que $t'[0:i']$ est trié.
 - Si $i = 0$, alors on rentre dans le **if** et $i' = 1$, donc $t'[0:i']$ ne contient qu'un seul élément : OK.
 - Sinon, si $t.(i) \geq t.(i-1)$, alors on rentre dans le **if**, donc $i' = i + 1$, et $t'[0:i']$ est trié car, par hypothèse d'invariant (HI), $t'[0:i'-1] = t[0:i]$ est trié et l'élément qu'on lui rajoute est $t.(i) \geq t.(i-1)$ qui est le dernier élément de $t'[0:i'-1]$.
 - Sinon, on passe dans le **else**, donc $i' = i - 1$, donc $t'[0:i']$ est inclus dans $t[0:i-1]$, qui est trié par (HI).
 - Montrons que $t'[i':i'_{\max}]$ est trié.
 - Si on entre dans le **if**, alors :
 - soit $i'_{\max} = i_{\max}$, et dans ce cas $t'[i':i'_{\max}] = t[i+1:i_{\max}]$, qui est trié par (HI) ;
 - soit $i'_{\max} = i_{\max} + 1$, et ça ne peut se produire que parce que $i = i_{\max}$, donc $i' = i'_{\max}$, donc $t'[i':i'_{\max}] = t'[i':i']$ est vide, donc trié...
 - Sinon, on rentre dans le **else**, donc $i'_{\max} = i_{\max}$ et $i' = i - 1$, et on a $t.(i) < t.(i-1)$ avant d'échanger ces cases. Ainsi, à la fin de l'exécution du **begin ... end**, on a $t'.(i') = t.(i) < t.(i-1) = t'.(i'+1)$. De plus, $t'.(i'+1) = t.(i-1) \in t[0:i]$, donc, par (HI), est plus petit que les éléments de $t[i+1:i_{\max}] = t'[i'+2:i'_{\max}]$. Ainsi, $t'[i':i'_{\max}]$ est bien trié.
 - Montrons que les éléments de $t'[0:i'+1]$ sont plus petits que les éléments de $t'[i'+1:i'_{\max}]$.
 - Si $i'_{\max} > i_{\max}$, alors (même raisonnement que précédemment) $t'[i'+1:i'_{\max}]$ est vide : OK.
 - Sinon, $i'_{\max} = i_{\max}$, et qu'on est rentré dans le **if** :
 - $\forall a \in t'[0:i'] = t[0:i+1], \forall b \in t'[i'+1:i'_{\max}] \subset t[i+1:i_{\max}]$, on a $a \leq b$.
 - $t'.(i') = t.(i+1)$ est bien plus petit que tous les éléments de $t'[i'+1:i'_{\max}] = t[i+2:i_{\max}]$ car par (HI) $t[i:i_{\max}]$ est trié.

Donc OK.

- Sinon, on rentre dans le **else**, et $\forall a \in t'[0:i'] = t[0:i-1]$:
- on a $a \leq b$ pour $b \in t'[i'+2:i'_{\max}] = t[i+1:i_{\max}]$ par (HI) ;
- on a $a \leq t'.(i'+1) = t.(i-1)$ car par (HI) $t[0:i]$ est trié.

De plus, concernant la valeur $t'.(i') = t.(i)$:

- on a $t.(i) \leq b \in t'[i'+2:i'_{\max}] = t[i+1:i_{\max}]$ car par (HI) $t[i:i_{\max}]$ est trié ;
- on a $t'.(i') = t.(i) < t.(i-1) = t'.(i'+1)$ car on est rentré dans le **else**.

Donc OK.

Ainsi, tous les morceaux de l'invariant sont bien vrais pour t' , i' et i'_{\max} en bas de la boucle.

La propriété est donc bien un invariant de la boucle **while**.

- (a) D'après ce qu'on vient d'analyser, le programme va alterner entre des phases où l'on rentre dans le **if** à chaque passage successif de la boucle **while**, et des phases où l'on rentre dans le **else** à chaque passage successif de la boucle **while**.

En fait, le cas pire se produit quand le tableau est initialement trié à l'envers. Dans ce cas :

- chaque phase où l'on enchaîne les **if** dure i_{\max} passages dans la boucle **while**, et à la fin d'une telle phase, i_{\max} est incrémenté ;
- chaque phase où l'on enchaîne les **else** dure aussi i_{\max} passages (jusqu'à tomber sur $i = 0$).

Ainsi, on va effectuer au plus $\sum_{i_{\max}=0}^{n-1} 2 \times i_{\max} = n(n-1)$ passages dans la boucle **while**.

Donc la fonction **tri_gnome** termine toujours.

- (b) De plus, chaque passage dans la boucle **while** s'effectue en $\mathcal{O}(1)$, donc la complexité de **tri_gnome** \mathbf{t} est en $\mathcal{O}(n^2)$ où n est la longueur de \mathbf{t} .
- (c) Une fois qu'on est sorti de la boucle **while**, l'invariant est vrai pour $i = n$, donc en particulier, $\mathbf{t}[0:n] = \mathbf{t}$ est trié. Et les modifications effectuées sur le tableau ne sont que des échanges de cases (donc aucune valeur n'est perdue ou rajoutée par rapport au tableau de départ).

Donc l'algorithme est correct.

Exercice 3 (Fonction mystère) :

1. Les valeurs prises par la variable \mathbf{s} forment une suite strictement croissante à valeurs dans \mathbb{N} , la condition $!\mathbf{s} \leq \mathbf{n}$ ne peut donc rester vraie indéfiniment. Ainsi, la boucle **while** (et donc la fonction **f**) termine.
2. Conjecture : $\mathbf{f} \ \mathbf{n} = \lfloor \sqrt{\mathbf{n}} \rfloor$. On prend cette relation comme spécification de **f**.
3. Montrons que la propriété suivante est un invariant de boucle :

$$\mathbf{s} = (\mathbf{c} + 1)^2 \quad (1)$$

- Avant la boucle, on a $\mathbf{c} = 0$ et $\mathbf{s} = 1 = (\mathbf{c} + 1)^2$.
- Notons c et s les valeurs des variables en haut du corps de la boucle, et c' et s' leurs valeurs en bas du corps de la boucle. Supposons que $s = (c + 1)^2$, et montrons que $s' = (c' + 1)^2$. On a $c' = c + 1$ (ligne 5), et après la ligne 6 on a :

$$s' = \underbrace{s}_{(c+1)^2} + 2c' + 1 = \underbrace{(c+1)^2}_{c'} + 2c' + 1 = c'^2 + 2c' + 1 = (c' + 1)^2$$

Ainsi, la propriété (1) est bien un invariant de boucle.

Lorsqu'on sort de la boucle **while**, on a donc $(c + 1)^2 = s > n$. De plus, la valeur précédente de \mathbf{s} (qui vaut c^2) respectait la condition du **while**, donc $c^2 \leq n$. Au final, on a $c^2 \leq n \leq (c + 1)^2$.

Comme $x \mapsto \sqrt{x}$ est croissante, on a : $c \leq \sqrt{n} < c + 1$, donc **f** renvoie $c = \lfloor \sqrt{n} \rfloor$.

La fonction **f** est donc correcte.

4. On passe $c = \lfloor \sqrt{n} \rfloor$ fois dans la boucle **while**, et on effectue un nombre constant d'opérations élémentaires à chaque passage dans la boucle. On effectue de plus un nombre constant d'opérations élémentaires en dehors de la boucle. La complexité de **f** est donc un $\mathcal{O}(\sqrt{n})$.

Exercice 4 (Fonction de Hofstadter) :

1. $h\ 0 = 0$; $h\ 1 = 1$; $h\ 2 = 1$; $h\ 3 = 2$; $h\ 4 = 3$.
2. Montrons par récurrence forte que $\forall n \in \mathbb{N}$, $h\ n$ termine et $0 \leq h\ n \leq n$.

Initialisation : la propriété est vraie pour $n = 0$ d'après la question précédente.

Hérédité : Soit $n > 0$. Supposons que la propriété soit vraie pour $n' < n$. Alors, par HR, $h(n-1)$ termine, en renvoie un entier n' tel que $0 \leq n' \leq n-1 < n$. Donc, par HR, $h(n')$ termine, et renvoie un entier n'' tel que $0 \leq n'' \leq n' \leq n-1$. Donc $n \geq n - n'' \geq n - (n-1) = 1 \geq 0$. Ainsi, $h\ n$ termine, et renvoie $n - n''$, qui est bien dans l'intervalle voulu.

Ainsi, par principe de récurrence, la propriété est vraie pour tout $n \in \mathbb{N}$.

Exercice 5 (Problème de l'arrêt) :

1. `f1` termine sur les entiers naturels pairs. Montrons par récurrence sur $k \in \mathbb{N}$ que `f1` termine sur $2k$.

Initialisation : la propriété est vraie pour $k = 0$ (cas de base).

Hérédité : Soit $k > 0$. Supposons que la propriété soit vraie pour $k-1$. Alors, par HR, `f1`($2k-2$) termine. Donc `f1` termine sur $2k$.

Ainsi, par principe de récurrence, `f1` termine sur tous les entiers pairs.

2. La valeur de i est initialisée à n , et augmente de 1 à chaque passage dans la boucle. De plus, le programme termine lorsque i vaut 10. Ainsi, `f2` termine ssi $n \leq 10$ (et entre dans une boucle infinie sinon).

3.

```
let infini n =
  while true do
    print_string "INFINI !!!\n"
  done
```

4. (a) `string -> 'a -> bool`.

- (b)

```
let bizarre code_f x = if arret code_f x then infini 42
```

- (c)

```
let paradoxe code_f = bizarre code_f code_f
```

- (d) D'après la propriété demandée sur `paradoxe` à la question précédente, `paradoxe code_paradoxe` termine ssi `paradoxe code_paradoxe` ne termine pas : impossible.

Ainsi, par l'absurde, on a prouvé qu'une telle fonction `arret` n'existe pas.

Exercice 6 (Modélisation de l'inondation d'une grotte) :

1.

```
let verif_profondeur (grotte:profil) =
  let rec aux l p = match l with
    | [] -> p = 0
    | Bas::q -> aux q (p+1)
    | Haut::q -> aux q (p-1)
    | _::q -> aux q p
  in aux grotte 0
```

```
2. let verif_sans_rebroussement (grotte:profil) =
  let rec aux l = match l with
    | [] -> true
    | [Bas] -> false
    | [_] -> true
    | Gauche::Droite::_ -> false
    | Droite::Gauche::_ -> false
    | Haut::Bas::_ -> false
    | Bas::Haut::_ -> false
    | t::q -> aux q
  in
  match grotte with
  | [] -> true
  | Haut::_ -> false
  | _::l -> aux grotte
```

```
3. let verif_vallee (grotte:profil) =
  let rec montee l = match l with
    | [] -> true
    | Gauche::_ | Bas::_ -> false
    | _::q -> montee q
  in
  let rec descente l = match l with
    | [] -> true
    | Gauche::_ -> false
    | Bas::q | Droite::q -> descente q
    | Haut::q -> montee q
  in descente grotte
```

```
4. let fond (vallee:profil) =
  let rec parcours l (xf,yf) (xc,yc) =
    (* (xf,yf) est la position du fond,
       * (xc,yc) est la position courante *)
    match l with
    | [] -> (xf,yf)
    | Haut::q -> (xf,yf) (* c'est une vallée donc pas besoin de tester la suite *)
    | Gauche::q -> failwith "ce n'est pas une vallée"
    | Droite::q -> parcours q (xf,yf) (xc+1,yc)
    | Bas::q -> parcours q (xc,yc+1) (xc,yc+1)
  in parcours vallee (0,0) (0,0)
```

5. (a) (cf. page suivante)

- (b)
- Les fonctions `montee` et `descente` ne parcourent qu'une seule fois la liste `vallee`, et chaque case est traitée en $\mathcal{O}(1)$. Ainsi, l'appel `descente vallee (0,0)` s'effectue en $\mathcal{O}(n)$ où n est la longueur de la liste `vallee`.
 - De plus, la taille de la pile `p` et de la file `f` sont $\leq n$, on passe donc au plus n fois dans la boucle `while` de la fonction `fabrique_rectangles` au total cumulé de ses appels récursifs. Ainsi, l'appel `fabrique_rectangles ()` s'effectue en $\mathcal{O}(n)$.

Finalement, la fonction `decomposition_en_rectangles vallee` s'effectue bien en $\mathcal{O}(n)$.

```

let decomposition_en_rectangles (vallee:profil) =
  let p = creer_pile () in
  let f = creer_file () in
  let rec montee l (x,y) = match l with
    | [] -> ()
    | Gauche::_ | Bas::_ -> failwith "pas une vallée"
    | Droite::q -> montee q (x+1,y)
    | Haut::q -> enfiler f x ; montee q (x,y-1)
  in
  let rec descente l (x,y) = match l with
    | [] -> ()
    | Gauche::_ -> failwith "pas une vallée"
    | Bas::q -> empiler p x ; descente q (x,y+1)
    | Droite::q -> descente q (x+1,y)
    | Haut::q -> enfiler f x ; montee q (x,y-1)
  in
  descente vallee (0,0) ;
  let rec fabrique_rectangles () =
    if not (pile_vide p) then
      begin
        (* calcul du prochain rectangle *)
        let g = depiler p in (* extrémité gauche du rectangle *)
        let d = defiler f in (* extrémité droite du rectangle *)
        let h = ref 1 in (* hauteur du rectangle *)
        while not (pile_vide p)
          && g = sommet_pile p
          && d = prochain f
        do
          let _ = depiler p in
          let _ = defiler f in
          incr h
        done ;
        (d-g,!h) :: fabrique_rectangles ()
      end
    else []
  in fabrique_rectangles ()

```

6.

```

let hauteur_eau valee t =
  let rectangles = decomposition_en_rectangles valee in
  let rec traitement_rectangle l_rect t_restant = match l_rect with
    | [] -> 0.
    | (l,h)::q_rect ->
      let aire = l*h in
      if t_restant > aire then
        (float_of_int h) +. traitement_rectangle q_rect (t_restant - aire)
      else
        (float_of_int h) *. (float_of_int t_restant) /. (float_of_int aire)
  in traitement_rectangle rectangles t

```