Correction du devoir à faire pour le Lundi 3 Novembre

Exercice 1 (Files de priorité représentées en arbres binaires)

1. Voici les instructions demandées :

```
type arbre_bin =
  | Vide
  | Noeud of int*(arbre_bin)*(arbre_bin)
;;

let borne_sup = 1000000;;
let borne_inf = -1000000;;

let ex_1 =
  Noeud (34,
         Noeud (30,
                Noeud(18,
                      Noeud (10, Vide, Vide),
                      Noeud (8, Vide, Vide)),
                Noeud (20,Noeud (11, Vide, Vide),
                       Noeud (14, Vide, Vide))),
         Noeud (32,
                Noeud(30, Vide, Vide),
                Noeud (26, Vide, Vide)))
;;
```

2. Voici la fonction `etiquette` demandée :

```
let etiquette arb= match arb with
  | Vide -> borne_inf
  | Noeud(e,ag,ad) -> e
;;
```

3. Voici la fonction `test_file` demandée :

```
let rec test_file arb = match arb with
  | Vide -> true
  | Noeud(e,ag,ad) -> (e >= etiquette ag) && (e >= etiquette ad)
      && (test_file ag) && (test_file ad) ;;
```

4. Voici la fonction `echange_etiquette` demandée :

```
let echange_etiquette (arb:arbre_bin) nouvelle_etiquette = match arb with
  | Vide -> failwith "pas d echange possible"
  | Noeud(e,ag,ad) -> Noeud(nouvelle_etiquette,ag,ad)
;;
```

5. Voici la fonction `percolation` demandée :

```
let rec percolation arb = match arb with
  | Vide -> Vide
  | Noeud(e,ag,ad) ->
```

```
      let e_g = etiquette ag and e_d = etiquette ad in
        if e = max (max e_g e_d) e then Noeud(e,ag,ad)
        else
          if e_g > e_d then
            begin
              let percole_ag = percolation (echange_etiquette ag e) in
                Noeud(e_g,percole_ag,ad)
            end
          else
            begin
              let percole_ad = percolation (echange_etiquette ad e) in
                Noeud(e_d, ag, percole_ad)
            end
;;
```

6. Voici la fonction `supp_feuille` demandée :

```
let rec supp_feuille arb = match arb with
  | Vide -> failwith "arbre vide"
  | Noeud(e,Vide,Vide) -> (e,Vide)
  | Noeud(e,Vide,ad) -> let (e_finale,nouvel_arb) =
                              supp_feuille ad in
      (e_finale,Noeud(e,Vide,nouvel_arb))
  | Noeud(e,ag,ad) -> let (e_finale,nouvel_arb) = supp_feuille ag in
      (e_finale,Noeud(e,nouvel_arb,ad))
;;
```

7. Voici la fonction `supp_racine` demandée :

```
let supp_racine arb =
  let (e,nouvelle_arb) = supp_feuille arb in
    let abr_2 = echange_etiquette nouvel_arb e in
      percolation abr_2
;;
```

8. Voici la fonction `insere` demandée :

```
let rec insere e arb = match arb with
  | Vide -> Noeud(e,Vide,Vide)
  | Noeud(e_arb,ag,ad) when e>e_arb ->
      let hasard = Random.int 2 in
        if hasard = 0 then
          let nouvel_arb = insere e_arb ag in Noeud(e,nouvel_arb,ad)
        else
          let nouvel_arb = insere e_arb ad in Noeud(e,ag,nouvel_arb)
  | Noeud(e_arb,ag,ad) ->
      let hasard = Random.int 2 in
        if hasard = 0 then
          let nouvel_arb = insere e ag in Noeud(e_arb,nouvel_arb,ad)
        else
```

```
            let nouvel_arb = insere e ad in Noeud(e_arb,ag,nouvel_arb)
;;
```

9. Voici la fonction `file_of_liste` demandée :

```
let rec file_of_liste l = match l with
  | [] -> Vide
  | a::reste -> insere a (file_of_liste reste)
;;
```

Exercice 2 (Files de priorité représentées en tableaux (tas))

1. On utilise une fonction auxiliaire qui prend en argument l'indice à partir duquel on recopie.

```
let arbre_de_tas t =
  let rec aux k=
    if k > t.(0)
    then
      Vide
    else
      Noeud (t.(k), aux (2*k), aux (2*k+1))
  in aux 1
;;
```

2. On commence par déterminer la taille de l'arbre (fonction classique), puis on utilise une fonction auxiliaire qui prend en argument l'indice à partir duquel on doit recopier le tas.

```
let rec taille a = match a with
  | Vide -> 0
  | Noeud (e,ag,ad) -> 1 + taille ag + taille ad
;;

let tas_d_arbre a =
  let n = taille a in
  let t = Array.make (n+1) n in
  let rec aux k sous_arbre = match sous_arbre with
    | Vide -> ()
    | Noeud(e,Vide,Vide) -> t.(k) <- e
    | Noeud(e, ag ,ad ) -> t.(k) <- e; aux (2*k) ag; aux (2*k+1) ad in
  aux 1 a ;
  t
;;
```

3. On parcourt le tableau à partir de l'élément d'indice 2 (le premier qui n'est pas la racine) et on teste si le nœud est bien plus petit que son père. On rappelle que la taille du tas est le premier élément du tableau (qui doit donc être strictement inférieure à la taille du tableau) et que le père du nœud d'indice i est le nœud d'indice $\lfloor \frac{i}{2} \rfloor$.

```
let verification_tas t =
  if t.(0) >= Array.length t
  then
    false
  else
    let boo = ref true in
    for k =2 to t.(0) do
      boo := !boo && t.(k) <= t.(k/2)
    done;
    !boo
;;
```

4. (a) Voici la fonction `echange` demandée :

```
let echange t i j =
  let aux=t.(i) in
    t.(i) <- t.(j);
    t.(j) <- aux
;;
```

(b) Voici la fonction `remontee` demandée :

```
let remonte n t =
  let k=ref n in
    while (!k > 1) && ( t.(!k) > t.(!k/2) ) do
      echange t !k (!k/2);
      k:=!k/2
    done
;;
```

(c) Voici la fonction `insertion` :

```
let insertion x t =
  let n = t.(0)+1 in
  t.(n) <- x;
  t.(0) <-n;
  remonte n t
;;
```

5. Voici la fonction `creation` demandée :

```
let creation tab =
  let n = Array.length tab in
    let tas = Array.make (n+1) 0 in
      for k = 0 to (n-1) do
        insertion tab.(k) tas
    done;
    tas
;;
```

6. (a) Voici la fonction `fils_max` demandée :

```
let fils_max k t =
  let fg = 2*k and fd = 2*k+1 in
    if fg > t.(0)
    then
      k
    else
      if fg = t.(0)
      then
        fg
      else
        if t.(fg) >t.(fd)
        then
          fg
        else
          fd
;;
```

(b) Voici la fonction `descente` demandée :

```
let descente t =
  let k = ref 1 in
    let f = ref (fils_max !k t) in
      while t.(!k) < t.(!f) do
        echange t !k !f;
        k:= !f;
        f:=fils_max !k t
      done
;;
```

(c) Voici la fonction `suppression` demandée :

```
let suppression t =
  if t.(0) = 0
  then
    failwith "file vide"
  else
    begin
      echange t 1 t.(0);
      t.(0) <- t.(0) - 1;
      descente t
    end
;;
```

7. • Notons $C_c(n)$ le nombre de comparaisons et $C_a(n)$ le nombre d'affectations dans un tableau effectués au pire à l'appel de `remontee n t`.
On remarque que la valeur de la référence `k` vaut initialement n puis est divisée par deux à chaque passage dans la boucle jusqu'à atteindre la valeur 1 : il y a au pire $\log_2(n)$ passages dans la boucle.
A chaque passage dans la boucle, on effectue 2 comparaisons et deux affectations

dans un tableau (car un échange) donc $C_a(n) = C_c(n) = 2\log_2(n) = O(\log_2(n))$.

- Notons n la taille du tas `t`, $T_c(n)$ le nombre de comparaisons et $T_a(n)$ le nombre d'affectations dans un tableau effectués au pire à l'appel de `insertion x t`.
  On a $T_c(n) = C_c(n+1) = O(\log_2(n))$ et $T_a(n) = 2 + C_a(n+1) = O(\log_2(n))$.

- Notons n la taille du tas `t` , $C'_c(n)$ le nombre de comparaisons et $C'_a(n)$ le nombre d'affectations dans un tableau effectués au pire à l'appel de `descente t`.
  On remarque que la valeur de la référence `k` vaut initialement 1 puis est multipliée par deux à chaque passage dans la boucle jusqu'à atteindre au maximum la valeur n : il y a au pire $\log_2(n)$ passages dans la boucle.
  A chaque passage dans la boucle, on effectue 3 comparaisons (dans `fils_max`) et deux affectations dans un tableau (car un échange) donc $C'_c(n) = 3\log_2(n) = O(\log_2(n))$ et $C'_a(n) = 2\log_2(n) = O(\log_2(n))$.

- Notons n la taille du tas `t` , $T'_c(n)$ le nombre de comparaisons et $T'_a(n)$ le nombre d'affectations dans un tableau effectués au pire à l'appel de `suppression t`.
  On a $T'_c(n) = C_c(n-1) = O(\log_2(n))$ et $T'_a(n) = 3 + C_a(n-1) = O(\log_2(n))$.

8. (a) La fonction `tri_tas` renverra :

```
- : int array = [|9; 8; 4; 2|]
```

   (b) Voici la fonction `tri_tas` :

```
let tri_tas t =
  let tas = creation t in
    for k = 0 to tas.(0)-1 do
      t.(k) <- tas.(1);
      suppression tas
    done;
  t
;;
```

   (c)
   - Notons n la taille de `tab`, $T2_c(n)$ le nombre de comparaisons et $T2_a(n)$ le nombre d'affectations dans un tableau effectués au pire à l'appel de `creation tab`.
     La fonction effectue n boucles avec un appel à la fonction `insertion` sur un tas de taille n à chaque boucle donc $T2_c(n) = n \times T_c(n) = O\left(n\log_2(n)\right)$ et $T2_a(n) = n \times T_a(n) = O\left(n\log_2(n)\right)$.
   - Notons n la taille de `t`, $T3_c(n)$ le nombre de comparaisons et $T3_a(n)$ le nombre d'affectations dans un tableau effectués au pire à l'appel de `tri_tas t`.
     La fonction effectue n boucles avec une affectation et un appel à la fonction `suppression` sur un tas de taille $\leqslant n$ à chaque boucle donc $T3_c(n) \leqslant T2_c(n) + n \times T'_c(n) = O\left(n\log_2(n)\right)$ et $T3_a(n) \leqslant T2_a(n) + n \times (1 + T'_a(n)) = O\left(n\log_2(n)\right)$.