Ordre Supérieur

1 Les obligatoires

1.1 Générateur de liste

Écrire une fonction `gen_list` qui va générer une liste de n elements quelconques.
Conseil : utiliser `Random.int`.

Exemples :

```
 val gen_list : int -> int -> int list = <fun>

# gen_list 10 5;;
 - : int list = [1; 4; 4; 4; 0; 0; 3; 0; 0; 4]
```

1.2 Fonction `map`

Écrire la fonction `map` qui prend en arguments une fonction et une liste et construit une nouvelle liste avec l'application de la fonction prise en argument sur tous les éléments de la liste.

```
 val map : ('a -> 'b) -> 'a list -> 'b list = <fun>

# map (function x -> x * x) [1; 2; 3; 4];;
 - : int list = [1; 4; 9; 16]
```

1.3 Fonction `iter`

Écrire la fonction `iter` qui prend en arguments une fonction et une liste et applique cette fonction à tous les éléments de la liste.

```
 val iter : ('a -> 'b) -> 'a list -> unit = <fun>

# iter (print_int) [1; 2; 3; 4];;
 1234- : unit = ()
```

1.4 Fonction `for_all`

Écrire la fonction `for_all` qui prend en arguments une fonction booléenne et une liste et renvoie vrai si pour tous les éléments X de la liste, f X = true.

```
 val for_all : ('a -> bool) -> 'a list -> bool = <fun>

# for_all (function x -> x = 0) [0; 0; 0; 0];;
 - : bool = true
```

2 Chiffrement

2.1 Outils

Fonction `explode`

Écrire la fonction `explode` qui transforme une chaine en une liste de caractères.

```
val explode : string -> char list = <fun>

# explode "Chiffrement";;
- : char list = ['C'; 'h'; 'i'; 'f'; 'f'; 'r'; 'e'; 'm'; 'e'; 'n'; 't']
```

Fonction `implode`

Écrire la fonction `implode` qui transforme une liste de caractères en une chaine.

```
val implode : char list -> string = <fun>

# implode ['C';'h';'i';'f';'f';'r';'e';'m';'e';'n';'t'];;
- : string = "Chiffrement"
```

2.2 Chiffre de César

"Le ROT13 (rotate by 13 places) est un cas particulier du chiffre de César, un algorithme simpliste de chiffrement de texte. Comme son nom l'indique, il s'agit d'un décalage de 13 caractères de chaque lettre du texte à chiffrer. Son principal aspect pratique est que le codage et le décodage se font exactement de la même manière."
Source Wikipédia.

Écrire la fonction `s_rotn` qui applique la méthode du chiffrement de texte sur un caractère avec la valeur n.

```
val s_rotn: int -> char -> char = <fun>

# s_rotn 13 'Y';;
- : char = 'L'
```

Écrire la fonction `s_rot13` à partir de la fonction `s_rotn` qui force la valeur à 13.

```
val s_rot13: char -> char = <fun>

# s_rot13 'Y';;
- : char = 'L'
```

En utilisant les fonctions précendantes, écrire la fonction rot13 qui applique le chiffrement à une phrase.

```
val rot13: string -> string = <fun>

# rot13 "Chiffrement";;
- : string = "Puvsserzrag"
```

2.3 Vigenère

"Le chiffre de Vigenère est un système de chiffrement polyalphabétique, c'est un chiffrement par substitution, mais une même lettre du message clair peut, suivant sa position dans celui-ci, être remplacée par des lettres différentes, contrairement à un système de chiffrement monoalphabétique comme le chiffre de César (qu'il utilise cependant comme composant). Ce chiffrement introduit la notion de clé. Une clé se présente généralement sous la forme d'un mot ou d'une phrase. Pour pouvoir chiffrer notre texte, à chaque caractère nous utilisons une lettre de la clé pour effectuer la substitution. Évidemment, plus la clé sera longue et variée et mieux le texte sera chiffré."
Source Wikipédia.

Écrire la fonction `encode` qui encode une lettre du message en clair avec la bonne lettre de la clef.

```
val encode : char -> char -> char = <fun>
```

Écrire la fonction `decode` qui décode une lettre du message chiffré avec la bonne lettre de la clef.

```
val decode : char -> char -> char = <fun>
```

Écrire la fonction `vigenere` qui prend en paramètre la fonction d'encodage ou de décodage, le message à chiffrer ou déchiffrer et la clef.

```
val vigenere : (char -> char -> char) -> string -> string -> string = <fun>

# vigenere encode "J ADORE FAIRE DU CAML TOUTE LA JOURNEE" "VIGENERE";;
 - : string = "E IJSEI WEDZK HH GRQG BUYGI CE EWAVAIV"

# vigenere decode "WWRVVHXW OMI EVX XHIPUMEV RX DSKTI RRW TE XN FSVV" "SECRETKEY";;
 - : string = "............................................."
```

3 Les tris

3.1 Tri à bulles

"Le tri à bulles ou tri par propagation est un algorithme de tri qui consiste à faire remonter progressivement les plus grands éléments d'une liste, comme les bulles d'air remontent à la surface d'un liquide."
Source Wikipédia.

Écrire la fonction `bubble_sort` qui effectue un tri à bulle sur une liste passée en paramètre en utilisant une fonction de comparaison.

```
val bubble_sort : ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>
```

Principe

On va parcourir notre liste autant de fois qu'il y a d'élements dedans (ou tant que nous effectuons des changements) et comparer les élements deux à deux grace à la fonction de comparaison. Si la relation d'ordre est bien respectée, on continue sur le reste de la liste, sinon on permutte les deux éléments et on continue sur le reste de la liste.

3.2 Tri par insertion

"Le tri par insertion est un algorithme de tri classique dont le principe est très simple. C'est le tri que la plupart des personnes utilisent naturellement pour trier des cartes : prendre les cartes mélangées une à une sur la table, et former une main en insérant chaque carte à sa place."
Source Wikipédia.

Écrire la fonction `insertion_sort` qui effectue un tri par insertion sur une liste passée en paramètre en utilisant une fonction de comparaison.

```
val insertion_sort : ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>
```

Principe

On va prendre les éléments de notre liste de départ et les insérer dans une nouvelle liste en respectant la relation d'ordre donnée par la fonction de comparaison.

3.3 Tri fusion

"Le tri fusion est un algorithme de tri stable par comparaison. Ce tri est basé sur la technique algorithmique diviser pour régner. L'opération principale de l'algorithme est la fusion, qui consiste à réunir deux listes triées en une seule."
Source Wikipédia.

Écrire la fonction `merge_sort` qui effectue un tri fusion sur une liste passée en paramètre en utilisant une fonction de comparaison.

```
val merge_sort : ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>
```

Principe

Il peut être intéressant de décomposer le problème en sous problèmes plus simple :

— une fonction `merge` qui fusionne deux liste triée selon une relation d'ordre en une seule (triée).

— une fonction `split` qui éclate une liste en deux sous listes (de tailles semblables).

— une fonction `merge_sort` qui fera appel aux deux autres pour faire le tri demandé.

De plus on sait que les listes ne contenant qu'un seul élément sont toujours triées.

3.4 Tri rapide

"Le tri rapide (en anglais quicksort) est un algorithme de tri inventé par C.A.R. Hoare en 1961 et fondé sur la méthode de conception diviser pour régner. La méthode consiste à placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié."
Source Wikipédia.

Écrire la fonction `quick_sort` qui effectue un tri rapide sur une liste passée en paramètre en utilisant une fonction de comparaison.

```
val quick_sort : ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>
```

Principe

Comme pour l'exercice précédent, il peut être intéressant de décomposer le problème :

— une fonction `pivot` qui éclate une liste en deux sous listes selon le respect de la relation d'ordre.

— une fonction `quick_sort` qui va s'occuper de trouver le pivot et de lancer la récursion sur les deux listes résultantes.

3.5 La fonction qui mesurait les fonctions

En utilisant la fonction `Sys.time`, écrire une fonction `time` qui permet de mesurer le temps qu'une fonction (à un paramètre) met à donner son résultat.

```
val time : ('a -> 'b) -> 'a -> float = <fun>

# time (bubble_sort (<)) (gen_list 10000 10000);;
 - : float = 15.908999999999878
```