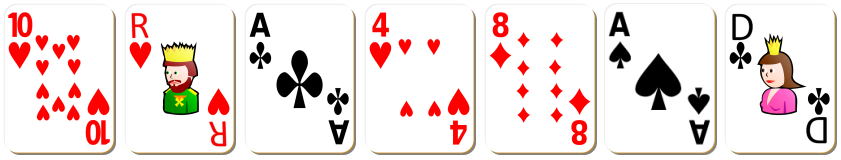
TD 6 : Tris

Dans ce TD, on testera les différents tris sur le jeu J suivant :



Une carte est caractérisée par sa couleur (C,K,P,T pour respectivement « cœur », « carreau », « pique » et « trèfle ») ainsi que sa valeur (de 2 à 10 puis 11 pour « valet », 12 pour « dame », 13 pour « roi » et 14 pour « as »). Une carte sera donc un doublet (valeur, 'couleur'). Par exemple l'as de pique sera représenté par (14, 'P').
La relation d'ordre est la suivante : on ordonne d'abord par valeur croissante, puis à valeur égale par couleur dans l'ordre alphabétique. Ainsi `(2, 'P')` < `(5,'K')` < `(5, 'T')` < `(12, 'C')`.

Tri fusion itératif

On suppose que l'on dispose d'une fonction `Fusion(L1,L2)` qui renvoie la liste ordonnée des éléments de `L1` et `L2`, deux listes déjà ordonnées.
Soit une liste `L` de longueur n quelconque. Pour lui appliquer un tri fusion itératif, on va à l'étape 1 travailler sur des listes de taille 1 (elles ne contiennent qu'un seul élément) et commencer par appliquer `Fusion` à `L[0:1]` et `L[1:2]`, puis à `L[2:3]` et `L[3:4]` et ainsi de suite jusqu'à ce que tous les éléments de `L` aient été examinés. On stocke progressivement les éléments triés dans une liste temporaire par laquelle `L` est remplacée en fin d'étape.
À l'étape 2, la taille des listes a doublé, et on applique `Fusion` à `L[0:2]` et `L[2:4]`, puis à `L[4 :6]` et `L[6 :8]`, et ainsi de suite.
On poursuit les étapes tant que la taille des listes que l'on fusionne est strictement inférieure à n.

1. On s'entraîne « à la main » sur J. Donner les différentes étapes du tri fusion itératif.
2. Que se passe-t-il si la longueur n de `L` n'est pas une puissance de 2 ? Combien d'étapes il y a-t-il ?
3. Programmer `TriFusion_iter(L)` et tester cette fonction sur le jeu de cartes proposé en affichant l'état de `L` à chaque étape.

Tri rapide

Le tri rapide vu en cours fait appel à une fonction `partition` qui permet de placer respectivement à gauche et à droite du pivot les éléments plus petits et plus grands que le pivot, ce tri se faisant *sur place*. Si on ne cherche pas a effectuer le tri sur place, on peut programmer plus facilement le tri rapide, toujours de façon récursive.
On utilise pour cela l'algorithme suivant :

- On choisit comme pivot le premier élément de la liste.
- On crée deux listes vides que l'on remplit respectivement avec les éléments plus petits et plus grands que le pivot.
- On trie récursivement ces deux listes que l'on concatène (dans le bon ordre) avec le pivot.

1. On s'entraîne « à la main » sur le jeu J. Donner les différentes étapes du tri rapide décrit ci-dessus.
2. Écrire une fonction `quicksort(L)` qui renvoie une liste triée des éléments de `L`.

Tri à bulles

Le principe du tri à bulles (dans l'ordre croissant) sur une liste `L` de longueur n est le suivant :

- On compare les deux premiers éléments de la liste `L[0]` et `L[1]` ; s'ils ne sont pas dans le bon ordre, on les permute.
- On fait de même avec le deuxième et le troisième élément, puis avec le troisième et le quatrième, jusqu'aux deux derniers éléments de la liste. À ce stade, le plus grand élément de la liste est `L[n-1]` et c'est sa position définitive.
- On recommence le processus pour la sous-liste `L[:n-1]`, puis pour `L[:n-2]`,..., et enfin pour `L[:2]`. La liste est alors triée sur place.

1. On s'entraîne « à la main » sur le jeu J. Donner les différentes étapes du tri à bulles décrit ci-dessus.
2. Écrire une fonction `tri_bulles(L)` qui effectue le tri sur place de `L`.
3. Quelle est la complexité du tri à bulles ? Montrer que l'on peut améliorer la complexité dans le meilleur des cas en interrompant le tri lorsqu'aucune permutation n'a été nécessaire. Déterminer cette complexité. Écrire la fonction `tri_bulles_optimise(L)` correspondante.