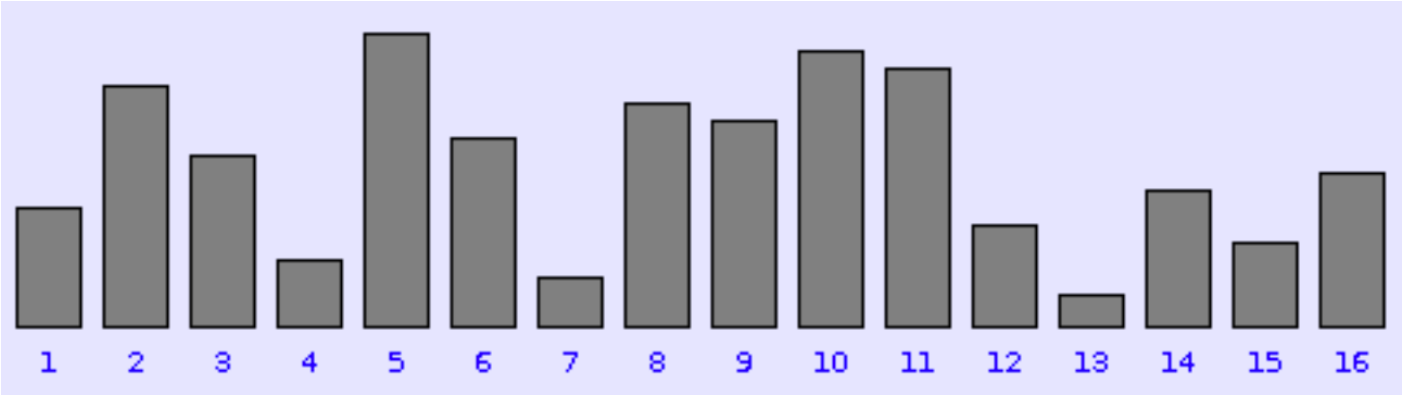
Algorithmique 2 - Les algorithmes de tris simples

Les ordinateurs stockent beaucoup d'informations et ont besoin de les analyser rapidement. Les moteurs de recherche sur Internet sont confrontés à l'un des plus grands problèmes de recherche au monde puisqu'ils doivent balayer des milliards de pages en une fraction de seconde. Les données qu'un ordinateur doit rechercher, que ce soit un mot, un code-barres ou le nom d'un écrivain, sont appelées *clés de recherche.*

Exemple : le tableau suivant contient 16 éléments de poids (représenté par la longueur) différents.



Question a : mesurer ces poids. Indiquer les valeurs sur le document.

Question b : Ecrire les données de ces poids dans un tableau T (une liste python). Dans le même ordre.

Trier des éléments d'un tableau est fondamental car la recherche d'un élèment dans une table est plus rapide si elle est ***ordonnée****.*

Remarques :

- **ordonnée** signifie …
- pour le tableau L suivant : [3.0, 1.9, 0.4, 9.5], la clé est un nombre … … …
- pour le tableau D suivant : ["19/03/2020", "01/04/2020", "10/06/2020"], la clé est une … … …
- pour le tableau M suivant : [("Paulette", 93), ("Marie", 62), ("Thomas", 45), ("Jean", 50)], la clé pour trier est l'information de l'age. Il s'agit de M[0][…], M[1][…], M[…

1 / Tri par insertion

Le tri par insertion est le tri que la majorité des joueurs de cartes occasionnels pratiquent intuitivement.

Il consiste à « traiter » toutes les cartes dans l'ordre découlant de la donne, le « traitement » se résumant, pour chaque carte, à l'insérer au bon endroit dans l'ensemble des cartes déjà triées.

Matériellement, cette opération peut être réalisée selon deux variantes :

Soit avec deux tas de cartes, l'un sur la table, résultat de la donne, l'autre dans la main du joueur, contenant le jeu trié. L'opération de tri commence alors avec la totalité des cartes sur la table, et se termine avec la totalité des cartes dans la main du joueur.

Soit directement dans la main du joueur, celle-ci étant partagée mentalement en un côté « trié » et un côté « pas encore trié ». L'opération de tri consiste alors à faire passer les cartes de l'un à l'autre en les insérant au bon endroit.

Dans les deux cas l'algorithme utilisé peut être le suivant :

```
Pour chaque carte de la donne :
    Regarder à la fin de la main triée
    Mémoriser la clé de cette carte
    Tant que la nouvelle carte va avant la carte de la main triée :
        Avancer le regard d'une carte vers la gauche dans la main triée
    Fin tant que
    Insérer la nouvelle carte à gauche de la carte de la main triée qu'on vient de regarder
Fin pour chaque
```

Remarques:

- le paquet de la main triée commence avec une seule carte.

– En pratique, le tri par insertion « humain » n'est pas aussi primitif, la recherche du point d'insertion se faisant plutôt par dichotomie que par balayage élément par élément.

Question c : Réaliser le tri par insertion pour le tableau **T** ci-dessus. Représenter le tableau après chaque opération de tri, en indiquant l'index du ***regard*** où se place la valeur (la carte dans la main triée), ainsi que l'index du début de la partie non triée.

Question d : Est-il nécessaire d'arriver jusqu'au dernier élément du tableau pour trier celui-ci en entier ?

Question e : Créer une fonction **`tri_ins`** à partir de l'algorithme vu plus haut. Ecrire le script de la fonction en python. Spécifier la fonction.

Question f : On souhaite contrôler au debut de la fonction que la liste à trier est non vide. Ecrire une pré-condition.

Question g : L'algorithme a bien traité toutes les données une fois sorti des boucles. Ecrire une post-condition. On suppose qu'il existe une fonction **`est_triee`**, dont on ne donne pas le code ici.

Question h : Programmer la fonction à l'aide d'un IDE python. Ajouter un compteur du nombre de boucles effectuée.

- Tester avec l'exemple initial.
- Puis tester avec la liste initiale déjà triée.
- Et enfin, tester avec la liste initiale, mais triée en sens inverse.

Conclure.

2 / Tri par sélection

On donne l'algorithme et sa spécification pour le tri par sélection :

```
Procédure tri_sélection (tab: tableau [1...n] de Element)
Précondition: les éléments de tab sont initialisés
Post-condition: les éléments de tab sont triés (plus petits à gauche)
Paramètre en mode donnée-résultat: tab
Variables locales:
      i,j,indmin : entiers
      min : Element
Début
      pour i allant de 1 à n-1 par pas de 1 faire
            indmin ← i
            pour j allant de i+1 à n par pas de 1 faire
                  si tab[j] < tab[indmin] alors
                        indmin ← j
                  finsi
            finpour
            min ← tab[indmin]
            tab[indmin] ← tab[i]
            tab[i] ← min
      finpour
Fin tri_sélection
```

Question i : Expliquer avec des mots le principe du tri par sélection.

Question j : Utiliser l'algorithme pour trier les éléments du tableau donné ci-dessus. Indiquer à chaque étape l'état du tableau, ainsi que la valeur de **`i`**, et celle de **`indmin`**.

Question k : Le nombre d'instructions exécutées par cet algorithme, dépend-il de la nature du tableau (déjà trié, non trié, trié à l'envers) ?

Question l : Programmer la procédure de tri par sélection. Adapter cette procédure pour trier la liste suivante selon la clé correspondant à l'age (le nombre entier):

```
M = [("Paulette", 93), ("Marie", 62), ("Thomas", 45), ("Jean", 50)]
```

Correction tri insertion version itérative

```
def tri_ins(t):
    for k in range(1,len(t)):
        temp=t[k]
        j=k
        while j>0 and temp<t[j-1]:
            t[j]=t[j-1]
            j-=1
        t[j]=temp
    return t
```

ENTREE : le tableau **t** contenant les éléments à trier.
SORTIE : aucune, **t** est modifié sur place .
ALGORITHME :

```
POUR i variant de 1 à (longueur – 1)
    cle ← t[i]
    j ← i – 1
    TANT QUE j ≥ 0 et que t[j] > cle
        t[j+1] ← t[j]
        j ← j – 1
    Fin TANT QUE
    t[j+1] ← cle
Fin POUR
Renvoyer VIDE (⌀)
```

Correction version recursive

```
def insere(t,j):
    if j>0 and t[j]<t[j-1]:
        t[j-1],t[j]=t[j],t[j-1]
        insere(t,j-1)
```

```
def tri_ins(t,j=1):
    if j<len(t):
        insere(t,j)
        tri_ins(t,j+1)
```

DOCUMENT :