TP 3| Listes et complexité

1. Wall-Street

Règle : pour évaluer correctement la complexité de vos algorithmes, il est nécessaire que vous n'utilisiez pas les fonctions min et max appliquées à des listes. Si on doit calculer un minimum ou un maximum, on le reprogrammera, au moins dans les premières questions.

On se donne une liste cot d'entiers naturels contenant n nombres cot_0, …, cot_{n-1} qui correspondent à la cotation d'une action en bourse aux jours 0,1,……n − 1.
On cherche à calculer le gain maximum possible que l'on peut réaliser à la bourse en achetant puis en vendant une action à deux jours à choisir pendant cette période.

1) On définit l'amplitude de la variation du cours comme le maximum de la valeur absolue de la variation de ce cours pendant la période observée, c'est-à-dire la quantité suivante :

$$amplitude = \max_{0 \le i \le j \le n-1} |cot_j - cot_i| \quad (1)$$

Noter qu'on a aussi :

$$amplitude = \max_{0 \le i \le n-1} cot_i - \min_{0 \le i \le n-1} cot_i \quad (2)$$

(i) Ecrire une fonction $amplitude1(cot)$ qui retourne comme résultat l'amplitude comme définie par la formule (2).
(ii) Ecrire une fonction $amplitude2(cot)$ qui retourne comme résultat l'amplitude comme définie par la formule (1).
(iii) Comparez la complexité de ces deux fonctions.

2) Le gain maximum est le gain maximum possible sur la période observée, c'est-à-dire la quantité suivante :

$$gain = \max_{0 \le i \le j \le n-1} (cot_j - cot_i)$$

(i) Donner un exemple où l'amplitude est différente du gain maximum. Que représente l'amplitude en termes de gain ou de perte ?
(ii) En suivant textuellement la définition du gain, écrire une fonction $gain(cot)$ qui retourne, en temps quadratique par rapport à n, le gain maximal possible sur le cours représenté par la liste cot.
(iii) Construire une procédure $gaindate(cot)$ en modifiant la procédure précédente afin de retourner la liste des couples formés des deux dates da et dv d'achat et de vente de l'action permettant d'obtenir le gain maximum sur la liste cot.

3) On se propose d'améliorer la fonction gain (2.ii), en une fonction $gainmieux$ qui renvoie le même résultat en temps linéaire par rapport à n. Pour cela, on doit n'utiliser qu'une seule boucle. Si on note i la variable qui parcourt les indices de cot, l'idée est d'utiliser une variable minimum, qui à chaque étape i contient la valeur de la plus petite cotation entre les temps 0 et i, ce qui permet de calculer le gain maximum qu'on peut faire en vendant au temps i. Ainsi, on peut en un seul parcourt de la liste déterminer le gain maximum.

2. Elections à deux tours (D'après X-ENS, PSI 2005)

On considère N candidats à une élection nationale, qui seront représentés chacun par une chaîne de caractères donnant leur nom, et une liste appelée vote qui donne (dans un ordre quelconque) la liste du vote de chaque habitant du pays pour un de ces candidats. On notera n le nombre d'habitants du pays.
Par exemple si N = 4, avec les candidats François, Nicolas, Alain, Olivier une liste vote avec n = 8 sera par exemple ["François", "Nicolas", "Olivier", "Nicolas", "Alain", "Alain", "Alain", "Olivier"] ce qui signifie que François a 1 voix, Nicolas 2 voix, Olivier 2 voix et Alain 3.

1) Le premier tour : pour qu'un candidat gagne au premier tour de scrutin, il faut et il suffit qu'il ait la majorité absolue des voix.

(i) Ecrire une fonction $winner(vote, candidat)$ qui prend en argument une liste de votes et un nom de candidat (chaîne de caractères) et qui renvoie $True$ si ce candidat a la majorité absolue et $False$ sinon.

(ii) A l'aide de la fonction précédente, écrire une fonction $gagne1(vote)$ qui renvoie le nom du candidat gagnant au premier tour s'il existe, et renvoie $None$ sinon. Cette fonction sera de complexité $O(n^2)$.

(iii) On suppose que des gentils assesseurs ont trié tous les bulletins de vote. Autrement dit, on suppose qu'on dispose d'une liste ordonnée des bulletins qu'on appelle encore vote.
Dans l'exemple précédent, avec l'ordre alphabétique décroissant, on aurait :

vote=["Olivier","Olivier","Nicolas","Nicolas","Francois","Alain","Alain","Alain"].

Ecrire alors une fonction $gagne1mieux(vote)$ de complexité linéaire, c'est-à-dire en O(n).

2) Une méthode plus maline pour compter les suffrages : on donne simplement un numéro aux candidats. La liste $vote$ des votes est simplement une liste de numéros, et on crée une fonction appelée $suffrage(vote)$ qui renvoie une liste S telle que S[i] contienne le nombre de voix obtenues par le candidat numéro i. Montrer qu'on peut faire ceci en temps linéaire. En déduire une façon d'avoir la même réponse que $gagne1$ en temps linéaire.

3) Pour le second tour, on suppose que le candidat ayant simplement le plus de suffrage gagne. Ecrire une fonction $gagne2(vote)$ qui donne le gagnant (ou la liste des ex-aequo).