TD2: validité et complexité

Notations asymptotiques

1. Caractériser les fonctions suivantes en terme de notation θ :
   - $f(n) = 4n^3 + n$
   - $f(n) = \frac{n(n+1)}{2}$
   - $f(n) = n * log_2(n)$
2. Parmi les fonctions suivantes, quelles sont celles qui ont le même ordre de grandeur ?
   - $f_1(n) = 4n^3 + n$
   - $f_2(n) = n^2 + log_2(n)$
   - $f_3(n) = n^2 * log_3(n) + 6n^3$
   - $f_4(n) = \frac{n(n^2+1)}{2}$
3. En utilisant la définition de θ montrer que $f(n) + g(n) = \theta(max(f(n), g(n)))$

Complexité

4. Exprimer en fonction de n le nombre de fois où est effectué $res+ = c$ dans les méthodes `String repete1(char c,int n)` et `String repete2(char c,int n)`. Ces deux méthodes ont-elles la même complexité ?

```
String repete1(char c,int n) {
        String res="";
        for (int i=0;i<n;i++) {
            for (int j = 0; j < n; j++)
                res += c;
        }
        return res;
    }
String repete2(char c,int n) {
        String res="";
        for (int i=0;i<n;i++) {
            for (int j = 0; j < i; j++) {
                res += c;
            }
        }
        return res;
    }
```

5. On s'intéresse à la méthode de tri appelée "tri par sélection" vue au 1er semestre et fournie dans la classe "TableauGenerique". Compléter les propriétés P1 et P2 et expliquer comment fonctionne ce tri. Calculez la complexité dans le pire et le meilleur des cas de la méthode "triSelection".

```
/** PRE :  leTableau est un tableau de K
  *  POST : leTableau est trié par ordre croissant */
 public void triSelection() {
     for (int i=0; i<this.leTableau.length;i++){
         //P1 : ..... (prop. sur leTableau et i)
         int indiceMin = i;
         for (int j = i+1; j < this.leTableau.length;j++)
             // P2 : ..... (prop. sur leTableau, indiceMin et i
             if (this.leTableau[indiceMin].compareTo(this.leTableau[j])>0) {
                 indiceMin = j;
             }
         K aux = this.leTableau[i];
         this.leTableau[i]= this.leTableau[indiceMin];
         this.leTableau[indiceMin] = aux;
     }
 }
```

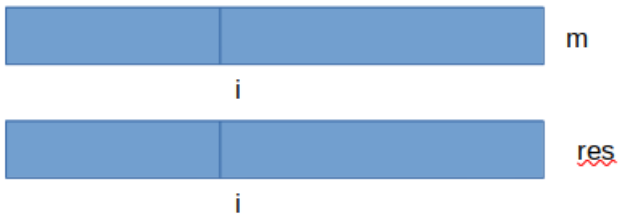
6. Calculer la complexité dans le pire des cas de la méthode "rechercheVite" du cours (compter le nombre de passages dans la boucle while).

Validité

7. On considère la méthode "mystere" de la classe "ValiditeComplexite" :

```
    // fonction mystere
    // POSTCONDITION: ..............................
    String mystere(String m) {
        String res = "";
        int i=0;
        while (i<m.length()) {
            // P1: propriété sur m, i, et res pour assurer la validité
            // ..............................
            res=m.charAt(i)+res;
            // P2: propriété sur i pour assurer la terminaison
            // ..............................
            i++;
        }
        return res;
    }
}
```

- Que vaut $mystere(\text{“}Bonjour\text{”})$?
- Donner les propriétés $P1$, $P2$ et la post-condition. Pour $P1$, vous pouvez utiliser le schéma ci-dessous.
- Calculer la complexité de la méthode "public int mystere(int x)".



8. Complétez la méthode suivante qui détermine si c est un palindrome. Un palindrome est un mot qui se lit aussi bien de gauche à droite que de droite à gauche. Par exemple, “rever” et “ressasser” sont des palindromes, “carotte” n’est pas un palindrome, “ravir” non plus.

   Pour accéder aux caractères d’une chaine vous pouvez utiliser la méthode “charAt(int i)” de la classe String. Par exemple si `s = ‘‘carotte’’` alors `s.charAt(3)` renvoie ’o’.

   Attention : à chaque étape de la boucle, la propriété P doit être respectée. L’exécution de “palindrome(”ressasser”)” doit afficher successivement **`rr reer resser`** et l’exécution de “palindrome(”épater retapé”)” doit afficher successivement **`éé éppé épaapé épattapé épateetapé épaterretapé`**. En respectant P, vous écrivez l’algorithme le plus efficace qui compare exactement une fois chaque élément.

```
public boolean palindrome(String c) {
    int i = 0;
    int j = c.length()-1;
    while ((i<j) && ..................)) {
        //  P  : si C1 est la sous-chaine de c allant de l’indice 0 à l’indice i-1
        //     : si C2 est la sous-chaine de c allant de l’indice j+1 à l’indice c.length()-1
        //     : alors C1C2 est un palindrome
        System.out.println(c.substring(0,i)+c.substring(j+1));
        ......................
        ......................
    }
    return ..............;
}
```

9. Calculer la complexité de la méthode “public boolean palindrome(String c)”.

10. On a modifié la ligne 9 de la méthode “rechercheVite”. Trouver un exemple de tableau pour lequel cette nouvelle méthode ne termine jamais.

```
1    public int rechercheVite(int[] tab, int x) {
2        int gauche = 0;
3        int droite = tab.length - 1;
4        int milieu  ;
5        while (gauche <= droite) {
6            milieu  = (gauche + droite) / 2 ;
7            if (x==tab[milieu]) return milieu;
8            if (x<tab[milieu])
9               droite = milieu;     // ERREUR (dans le cours : droite = milieu - 1)
10            else gauche = milieu + 1;
11       }
12       return -1;
13   }
```