TP : reconnaissance de classes Θ (Fin)

Question 1

Dans l'espace pédagogique cours *FLIN408 - Analyse d'algorithme (Hervé Dicky)* dans la partie *Documents et liens* $\rightarrow$ *TP1* téléchargez les fichiers

- fonctionsMysterieuses.h
- fonctionsMysterieuses.o
- SolutionsFonctionsMysterieuses.cpp

Le fichier fonctionsMysterieuses.o est la compilation des six fonctions mystérieuses déclarées dans fonctionsMysterieuses.h.
Chacune de ces fonctions mystérieuses renvoie pour résultat le produit par une constante (à chaque fois différente) d'une des fonctions

$$log(n),\ \sqrt{n},\ n^2,\ n^5,\ 2^n,\ 3^n$$

Il s'agit de déterminer la classe du résultat (**pas** la classe de complexité) renvoyé par chacune de ces fonctions mystérieuses.

indications pour trouver la solution :
Le fichier SolutionsFonctionsMysterieuses.cpp vous fournit un *main* qui vous permet de tester ces fonctions mystérieuses à la main.
Son rôle est seulement de vous rappeler l'usage du *switch* et de vous aider pour vos ordres de compilation.

Il est conseillé de travailler fonction mystérieuse après fonction mystérieuse.

Pour chaque fonction mystérieuse, commencez par trouver la plage significative dans lequel faire varier le paramètre n (que l'exécution ne soit pas instantanée, mais ne prenne pas non plus un temps démesuré).
Par exemple, voici deux essais possibles pour tester la fonction mysterieuse fx.

```
int main(){
  for (int n=100000; n < 1100000; n+=1000)// deuxieme essai  (int n=1000; n < 1010; n++)
    std::cout<< fx(n) <<"\n";
  return 0;
}
```

À ce stade, vous devez hésiter entre deux ou trois fonctions $\Phi(n)$ pour la classe du résultat renvoyé par votre fonction mystérieuse.
Pour chacune des fonctions $\Phi(n)$ à laquelle vous songez, de regarder si le *float* $rap = \frac{f_x(n)}{\Phi(n)}$ (en appelant f_x la fonction mystérieuse que vous étudiez) se stabilise à une valeur (non nulle) quand n croit dans sa plage significative.

Cette valeur stable vous donnera la constante multiplicative (qui est toujours soit un entier soit l'inverse d'un entier).
Par exemple, l'exemple suivant

```
int main(){
  float rap;
  for (int n=100000; n < 1100000; n+=1000){
    rap = ((float) (  fx(n)) /  (sqrt(n)); // deuxieme essai : fx(n) / (log(n))
    std::cout<< rap <<"\n";
  }
  return 0;
}
```

propose deux essais possibles pour deviner la complexité de fx (notez que la plage significative trouvée lors du premier test est utilisée lors du deuxième).

Attention au cas où des nombres négatifs apparaissent : cela signifie en général que soit $f_x(n)$, soit $\Phi(n)$ ont dépassé la valeur *int* maximale autorisée.
Pour les fonctions qui renvoie des réels, ne vous étonnez pas de petites erreurs d'arrondi.

Travail à rendre pour la question 2

Votre `main` devra successivement, pour chacune des fonctions mystérieuses et sa solution, afficher leur rapport pour un intervalle significatif de valeurs (comme dans l'exemple ci dessus).

Question 2 au cas où vous vous ennuyez

1. écrivez une fonction *int Incrementer(int n, bool T[])*, qui
   — étant donné un tableau T de dimension n qui représente un entier n_T écrit en base 2 avec n bits, [1] tel que $T[0]$ vaut 0
   — modifie ce tableau T pour représenter le nombre $n_T + 1$
   — et renvoie le nombre d'**affectations** effectuées dans le tableau T (on ne s'occupe pas du nombre d'affectations des variables auxiliaires).
   Écrire cette fonction de façon à minimiser ce nombre d'affectations (on ne s'occupe pas non plus du nombre de tests).

2. Écrire ensuite une fonction *int Compte(int n)* qui compte le nombre moyen d'affectations que l'on effectue quand on incrémente **tous** les tableaux qui représentent les entiers entre 0 et $2^n - 1$.
   Tester pour les valeurs de n de 1 à 28 et conclure.

1. par exemple le tableau | 0 | 1 | 0 | représente le nombre 2