Examen du 24 Mars 2012 (2h)
TOP : Techniques and tOols for Programming
Première année

La notation tiendra compte de la validité des réponses, mais aussi de la présentation et de la clarté de la rédaction.

Documents interdits, à l'exception d'une feuille A4 à rendre avec votre copie.

Questions de cours (2pts)

- Question 1. Définissez en français (sans équation) les notations O, Ω et Θ utilisées pour dénoter la complexité algorithmique en insistant sur leurs relations les unes avec les autres ($\frac{1}{2}$ pt).
- Question 2. Quel est le rapport entre les notations que vous venez de définir et les temps de calcul dans le meilleur des cas, le pire des cas et le cas moyen ? ($\frac{1}{2}$ pt)
- Question 3. À quelles classes de complexité (en notation Θ) appartiennent les algorithmes 1 et 2 suivants ? ($\frac{1}{2}$ pt)

algorithme 1

```
pour i = 1 à n faire
  pour j = 1 à n faire
    x += 3
```

algorithme 2

```
pour i = 1 à n faire
  pour j = 1 à n faire
    x += 3
pour i = 1 à n faire
  y = x + 5
```

- Question 4. Qu'est ce que le backtracking ($\frac{1}{2}$ pt) ?

Identification d'algorithmes de tris (4pts)

La colonne la plus à gauche constitue les données d'entrée du problème. La colonne la plus à droite représente les données de sortie, c'est-à-dire les données triées alphabétiquement. Chacune des autres colonnes représente une étape intermédiaire de l'un des algorithmes de tri listés ci-dessous.

① Tri à bulle, ② Tri fusion, ③ Tri par insertion, ④ Tri par selection, ⑤ QuickSort (en prenant le premier élément du sous-tableau comme pivot), ⑥ ShellSort.

(in)	A	B	C	D	E	F	(out)
gens	aveu	dard	gens	dard	gens	aveu	aveu
lama	base	file	lama	file	lama	base	base
zoom	miel	gens	inox	char	pain	char	char
pain	char	inox	pain	base	zoom	dard	dard
inox	file	lama	dard	aveu	dard	file	file
tape	rage	pain	file	gens	file	gens	gens
dard	dard	tape	miel	lama	inox	pain	inox
file	inox	zoom	char	zoom	tape	inox	kilo
miel	tape	miel	kilo	pain	char	miel	lama
char	gens	char	rage	inox	kilo	zoom	miel
kilo	kilo	kilo	sort	tape	miel	kilo	pain
rage	vous	rage	base	miel	rage	rage	rage
sort	pain	sort	tape	kilo	aveu	sort	sort
base	lama	base	aveu	rage	base	lama	tape
vous	zoom	vous	vous	sort	sort	vous	vous
aveu	sort	aveu	zoom	vous	vous	tape	zoom

- **Question.** Pour chacun des algorithmes, indiquez la colonne représentant une étape intermédiaire. Décrivez également quelle serait l'opération suivante (il n'est pas nécessaire de calculer l'état du tableau après cette opération, mais simplement de décrire en français l'opération réalisée).

Récursivité (5 pts)

Considérez le code mystère suivant.

```
public int puzzle(int i, int j) {
  if (i == 1)
    return j;
  if (i % 2 == 1)
    return j+puzzle(i/2,j*2);
  else
    return   puzzle(i/2,j*2);
}
```

- **Question 1($\frac{1}{2}$pt)** Explicitez les appels récursifs effectués pour `puzzle(25,4)`.
- **Question 2(1pt)** Calculez le résultat de la fonction puzzle pour les valeurs suivantes. (10,1) (10,2) (10,3) (10,4) (10,6) (10,8) (10,10) Que semble calculer `puzzle()` ?
- Question 3($\frac{1}{2}$pt) Montrez la terminaison de cet algorithme.
- Question 4($\frac{1}{2}$pt) Quelle est la complexité algorithmique de `puzzle` (en nombre d'appels récursifs) ?
- Question 6($\frac{1}{2}$pt) Est-il possible de dérécursiver directement cette fonction ? Pourquoi ?
- Question 7(2pts) Dérécursivez cette fonction en appliquant les méthodes vues en cours (en une ou plusieurs étapes). Explicitez ce que vous faites et pourquoi.

Les tests (5pts)

Les fonctions ci-dessous appartiennent à une classe permettant de stocker des chaînes de caractères sous forme d'une pile.

- Question 1. Pour chaque méthode, indiquez les tests qu'il faudrait écrire pour vérifier leur bon fonctionnement. Écrivez simplement les tests à réaliser en français (vous n'avez pas à les écrire explicitement en utilisant JUnit ou une autre technique) (3pts).
- Question 2. Écrivez en JUnit les tests de la méthode `isEmpty()` (1pt).
- Question 3. Avez vous utilisé une approche whitebox, greybox, bluebox ou blackbox testing ? Justifiez votre réponse ? (1pt)

```
public class MyStack {
  /** Tests if this stack is empty.
   * @return True if and only if this stack contains no items; false otherwise
   */
  public boolean isEmpty()

  /**
   * Pushes an item onto the top of this stack.
   * @param item { the item to be pushed onto this stack
```

```
     * @return The item that was pushed onto the stack
     */
    public String push(String item)

    /**
     * Removes the object at the top of this stack and returns that object as the value of this
     * function.
     * @return The object at the top of this stack
     * @throws EmptyStackException if this stack is empty.
     */
    public String pop()

    /**
     * Looks at the object at the top of this stack without removing it from the stack.
     * @return the object at the top of this stack
     * @throws EmptyStackException if this stack is empty
     */
    public String peek()
}
```

Preuve de programme (4 pts) (D'après Baynat, Exercices et problèmes d'algorithmique)

Soit l'algorithme itératif ci-contre. Nous allons calculer formellement la précondition nécessaire pour que la post-condition de cette algorithme soit $Q \equiv res = 2^{2^n}$ en utilisant les formules de calcul de la plus faible précondition rappelées en annexe.

```
public int puzzleIter(int n) {
  int res=2;
  int i=0;
  while (i<n) {
    res = res * res;
    i=i+1;
  }
  return res;
}
```

- Question 1(1pt). Explicitez l'invariant et le variant de la boucle. N'oubliez pas que cet invariant et variant vont servir ensuite pour les obligations de preuve.
- Question 2 (3pts). Calculez la plus faible précondition nécessaire pour que cet algorithme calcule 2^{2^n}. Explicitez ce que vous faites et pourquoi.

Annexe : Règles de calcul des préconditions.

1- $\mathbf{WP}(nop, Q) \equiv Q$

2- $\mathbf{WP}(x := E, Q) \equiv Q[x := E]$

3- $\mathbf{WP}(C; D, Q) \equiv \mathbf{WP}(C, \mathbf{WP}(\mathrm{D},\mathrm{Q}))$

4- $\mathbf{WP}(\texttt{if } Cond \texttt{ then } C \texttt{ else } D, Q) \equiv (Cond = \texttt{true} \Rightarrow \mathbf{WP}(C, Q)) \wedge (Cond = \texttt{false} \Rightarrow \mathbf{WP}(D, Q))$

5- $\mathbf{WP}(\texttt{while } E \texttt{ do } C \texttt{ done } \{\mathrm{inv\ I\ var\ V}\}, \mathrm{Q}) \equiv I$; Obligations de preuve :

- $(E = \texttt{true} \wedge I \wedge V = z) \Rightarrow \mathbf{WP}(C, I \wedge V < z))$
- $I \Rightarrow V \geq 0$
- $(E = \texttt{false} \wedge I) \Rightarrow Q$