Examen du 6/03/2010 (2h)

TOP: Techniques and tOols for Programming

Première année

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

La notation tiendra compte de la présentation et de la clarté de la rédaction.

★ **Questions de cours.** (4pt)

▷ **Question 1:** (1pt) Quelle est la différence entre la complexité dans le pire cas et la borne supérieure de complexité ?

▷ **Question 2:** (1pt) Définissez les types de récursivité suivants : terminale, générative, mutuelle (ou croisée) et structurelle.

▷ **Question 3:** (1pt) Qu'est ce que le backtracking ?

▷ **Question 4:** (1pt) Définissez les tests (1) white box (2) black box (3) d'intégration (4) de régression.

★ **Exercice 1: Code récursif mystère** (5pt).

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(4,25)`.

▷ **Question 2:** (1pt) Calculez le résultat de la fonction puzzle pour les valeurs suivantes. (1,10) (2,10) (3,10) (4,10) (6,10) (8,10) (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 5:** ($\frac{1}{2}$pt) Est-il possible de dérécursiver directement cette fonction ? Pourquoi ?

▷ **Question 6:** (2pt) Dérécursivez cette fonction en appliquant les méthodes vues en cours (en une ou plusieurs étapes). Explicitez ce que vous faites et pourquoi.

★ **Exercice 2: Encore un code mystère (mais pas récursif)** (d'après Maylis DELEST – 4pt).

On considère le tableau T012=$\{1, 0, 2, 0, 2, 1\}$ et la fonction `swap(tab, a,b)`, qui inverse les valeurs des cases `tab[a]` et `tab[b]`.

```
void puzzle2(int tab[]) {
  int i=0,j=0,k=tab.length-1;
  while (i<=k) {
    if (tab[i] == 0) {
      swap(tab,i,j);
      j=j+1;
      i=i+1;
    } else if (tab[i] == 1) {
      swap(tab,i,k);
      k=k-1;
    } else {
      i=i+1;
    }
  }
}
```

▷ **Question 1:** (2pt) Donnez la valeur du tableau aux différentes étapes de l'appel `puzzle2(T012)`.

▷ **Question 2:** (1pt) Dans le cas général si T est un tableau d'entiers dont les valeurs sont les entiers 0, 1 ou 2 ($T \in \{0,1,2\}^{|T|}$), quel est le résultat de la fonction puzzle2() sur T ? Argumentez votre réponse.

▷ **Question 3:** (1pt) Quelle est la complexité de cette fonction ? Montrez la terminaison de cet algorithme. Justifiez vos réponses.

★ **Exercice 3: Preuve de programmes** (4pt).

Considérez le code de la fonction ci-contre calculant la factorielle de façon itérative. Calculez la plus faible précondition (Weakest Precondition, **WP**) nécessaire pour que la post-condition soit :

$$Q \triangleq res = n!$$

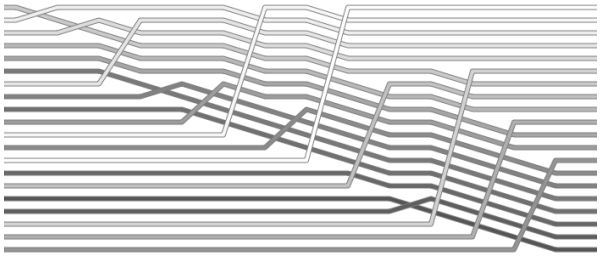
Les règles de calcul des préconditions sont rappelées en annexe.

```
int res;
void Factorial (int n) {
  int f = 1;
  int i = 1;
  while (i < n) {
    i = i + 1;
    f = f * i;
  }
  res = f;
}
```

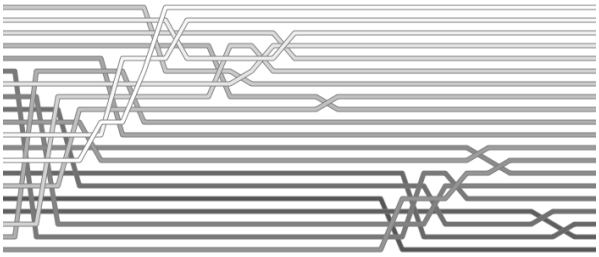
★ **Exercice 4: Identification d'algorithmes de tri** (d'après Aldo Cortesi – 3pt).

Les schémas suivants montrent le fonctionnement de divers algorithmes de tris. Chaque trait grisé indique une valeur, et l'axe des abscisses montre le temps qui passe tandis que l'axe des ordonnées montre la position de chaque valeur (=trait) dans les différentes cases du tableau. La case n°1 est en haut, et la case n°20 est en bas, et une couleur plus claire signifie une valeur plus petite. Quand deux traits se croisent, c'est que l'algorithme a inversé les deux valeurs à cet instant.

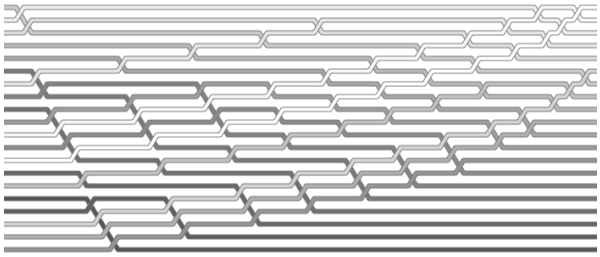
▷ **Question 1:** Identifiez le comportement des algorithmes suivants : tri à bulle, tri par insertion, tri par sélection, shell sort, quick sort. Argumentez vos réponses.



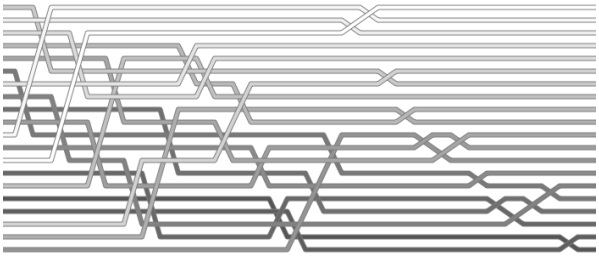
Algorithme A.



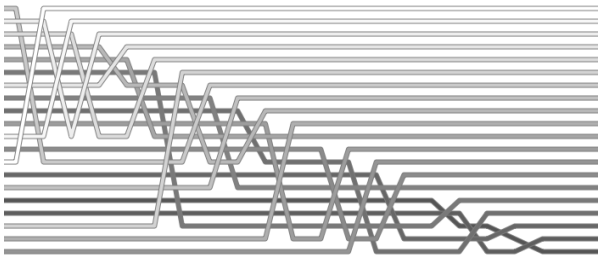
Algorithme B.



Algorithme C.



Algorithme D.



Algorithme E.

★**Annexe :** Règles de calcul des préconditions.

- $\mathbf{WP}(nop, Q) \equiv Q$
- $\mathbf{WP}(x := E, Q) \equiv Q[x := E]$
- $\mathbf{WP}(C; D, Q) \equiv \mathbf{WP}(C, \mathbf{WP}(\mathrm{D,Q}))$
- $\mathbf{WP}(\texttt{if}\ Cond\ \texttt{then}\ C\ \texttt{else}\ D, Q) \equiv (Cond = \texttt{true} \Rightarrow \mathbf{WP}(C, Q)) \wedge (Cond = \texttt{false} \Rightarrow \mathbf{WP}(D, Q))$
- $\mathbf{WP}(\texttt{while}\ E\ \texttt{do}\ C\ \texttt{done}\ \{\mathrm{inv\ I\ var\ V}\}, \mathrm{Q}) \equiv I$; Obligations de preuve :
  - $(E = \texttt{true} \wedge I \wedge V = z) \Rightarrow \mathbf{WP}(C, I \wedge V < z))$
  - $I \Rightarrow V \geq 0$
  - $(E = \texttt{false} \wedge I) \Rightarrow Q$