Java & Algorithme – Corrigé du test final

1 PETITE MISE EN ROUTE (/6) – 20 MINUTES

1.1 Que font ces algorithmes ?

1. L'algorithme retourne le plus petit élément du tableau r.

2. L'algorithme retourne le nombre d'éléments du tableau r qui sont supérieurs à a.

3. L'algorithme cherche le c le plus petit possible tel que $c^a > r$, et retourne c^a-1.

4. L'algorithme retourne un tableau où une case sur deux est une case du tableau a, et une case sur deux est une case du tableau b. Si a et b ne font pas la même taille, l'algorithme retourne null.

1.2 Pourquoi ces algorithmes vont (ou peuvent) bugger, et comment l'éviter ?

1.

```
public static int[] Somme2a2(int[] a)
{
   int[] c = new int[a.length - 1];  L'algorithme va calculer dans c la somme des cases consécutives de a : dans la première case, on aura la première case de a plus la deuxième, dans la seconde case, on aura la seconde case de a plus la troisième. Le tableau c fait donc une case de moins que a.
   for(int d=0; d<a.length-1; d=d+1)  Pour éviter de sortir du tableau a, étant donné qu'on va jusqu'à a[d+1]
   {
      c[d] = a[d] + a[d+1];
```

```
   }
   return c;
}
```

On peut aussi proposer que c soit un tableau de long, afin d'être sûr de ne pas dépasser la capacité des entiers quand on fait la somme des cases de a (dans ce cas, point bonus).

2. Rechercher l'élément maximal du tableau a.

```
public static int RechercheMax(int[] a)
{
   int c=a[0]; Il faut initialiser c avec un élément du tableau a

   int b=0;
   while(b < a.length)
   {
      if(a[b] >= c)
      {
         c=a[b];
      }
      b=b+1; Quoiqu'il arrive, il faut avancer le compteur sinon, on peut rester bloqué dans la boucle.
   }
   return c;
}
```

3. Réaliser le produit de tous les éléments du tableau a.

```
public static int Produit(int[] a)
{
   long c=1; Pour éviter (mais ce n'est pas garanti) de dépasser la capacité des entiers lors des multiplications, on fera le calcul dans un long.

   for(int b=0; b<a.length; b=b+1)
   {
      c=c*a[b];
   }
   return c;
}
```

1.3 Comment corriger ces algorithmes pour qu'ils compilent et aient du sens ?

Je ne vous demande pas de réécrire tous les algorithmes, mais seulement de corriger les parties qui ne vont pas.

1. Fusion, dans le tableau c, des tableaux a et b.

```
public static int[] FusionTableaux(int[] a, int[] b)
{
   int[] c = new int[a.length + b.length];

   for(int cpt=0; cpt<a.length; cpt=cpt+1)
   {
      c[cpt]=a[cpt];
   }

   for(int cpt=0; cpt<b.length; cpt=cpt+1)
   {
      c[cpt+a.length]=b[cpt];
   }

   return c;
}
```

2. Calcul de la moyenne des éléments du tableau a.

```
public static double Moyenne(int[] a)
{
   int c = 0;

   for(cpt=0; cpt<a.length; cpt=cpt+1)
   {
      c=a[cpt]+c;
   }

   double moyenne = (double)c/a.length;
   return moyenne ;
}
```

2 FAIRE UNE SUPER EQUIPE DE SUPER-HEROS (/7) – 35 MINUTES

1. Voici le tableau de toutes les combinaisons possibles.

Bioman force rouge	**Batman**	**IronMan**	**KungFu Panda**	Poids	Puissance	Décimal
0	0	0	0	0	0	0
0	0	0	1	135	3	1
0	0	1	0	110	12	2
0	0	1	1	245 **X**	15	3
0	1	0	0	71	7	4
0	1	0	1	206	10	5
0	1	1	0	181	19	6
0	1	1	1	316 **X**	22	7
1	0	0	0	85	5	8
1	0	0	1	220 **X**	8	9
1	0	1	0	195	17	10
1	0	1	1	330 **X**	20	11
1	1	0	0	156	12	12
1	1	0	1	291 **X**	15	13
1	1	1	0	266 **X**	24	14
1	1	1	1	401 **X**	27	15

2. En tout, on a testé 16 combinaisons.
Dans le cas de n super-héros, il faudrait tester 2^n combinaisons.
Pour 10 super-héros, il faudra tester 2^{10}=1024 combinaisons.

3. A chaque ligne du tableau, on peut interpréter l'équipe formée (la suite de 1 et de 0) comme un nombre binaire, dont le nombre décimal correspondant apparait sur la colonne de droite.
On voit donc que, au lieu d'essayer de générer différentes équipes, il suffit de parcourir tous les nombres de 0 à 15, et convertir le nombre en binaire afin d'obtenir une composition d'équipe à tester.

4. Il suffit de suivre l'exemple donné en annexe.

```
public static int[] ConversionBinaire(int x)
{
    int[] result = new int[10];
    int a = x;
    int cpt = result.length - 1; //cpt permettra de pointer où écrire le 1 ou
                                 //le 0 dans result
                                 //au début, on se place tout à droite du tableau

    while( a != 0)
    {
        if(a%2 == 0)
        {
            result[cpt]=0;
        }
        else
        {
            result[cpt]=1;
        }
        cpt=cpt-1; //on se déplace vers la gauche dans le tableau result
        a=a/2;
    }

    return result;
}
```

5. On suit les indications données dans l'énoncé : on parcourt tous les nombres de 0 à 1023, on convertit chaque nombre en un nombre binaire sous forme de tableau grâce à ConversionBinaire. Ce tableau sera interprété comme une équipe, dont on calcule le poids et la puissance. En parcourant tous les nombre de 0 à 1023, on teste les 1024 possibilités d'équipe, on s'assure donc de tester toutes les combinaisons d'équipe.

```
int[] meilleure_equipe = new int[10];
int meilleure_puissance = 0;

for(int equipe=0; equipe<1024; equipe=equipe+1)
{
   int[] composition_equipe = ConversionBinaire(equipe);

   //composition_equipe est un tableau de 10 cases de 1 et de 0
   //représentant l'équipe
   //on calcule le poids de cette équipe et sa puissance
   int poids=0;
   int puissance=0;
   for(int cpt=0; cpt<composition_equipe.length; cpt=cpt+1)
   {
       if(composition_equipe[cpt]==1) //Si le héros n° cpt est dans l'équipe
       {
           poids = poids + Poids[cpt]; //On ajoute son poids
           puissance = puissance + Puissance[cpt];
       }
   }

   if(poids <= 300 && puissance > meilleure_puissance)
   {
       meilleure_puissance = puissance;
       for(int cpt=0; cpt<composition_equipe.length; cpt=cpt+1)
```

```
        {
            meilleure_equipe[cpt] = composition_equipe[cpt];
        }
    }
}

//On affiche la meilleure équipe
for(int cpt=0; cpt<meilleure_equipe.length; cpt=cpt+1)
{
    System.out.println(meilleure_equipe[cpt]);
}
```

6. Si l'on souhaite juste envoyer le plus de héros possibles, on place, dans l'ascenseur, les héros en commençant par les plus légers. On remplit ainsi l'ascenseur jusqu'à ne plus pouvoir mettre de héros dans dépasser le poids, et c'est bon.

3 BIENVENUE A LA STAR ACADEMY (/7) – 35 MINUTES

1. On parcourt la liste, et si jamais on tombe sur un vote possédant la même adresse que notre vote, on le note. Une fois le parcours terminé, si on n'a noté aucune fois l'adresse ip recherchée, alors on ajoute notre vote dans la liste. On ajoutera le vote en seconde position de liste, c'est le plus simple à réaliser.

```
public static void AjouterVote(Vote tete_de_liste, Vote vote_a_ajouter)
{
    Vote parcours = tete_de_liste;
    boolean trouve_adresse=false;

    while(parcours!=null)
    {
        if(parcours.adresse_votant == vote_a_ajouter.adresse_votant) //Le votant
a deja vote !
        {
            trouve_adresse = true; //on le mémorise
        }

        parcours = parcours.suivant;
    }

    if(trouve_adresse == false) //on n'a jamais vu l'adresse du votant dans la
liste
    {
        //on ajoute le votant dans la liste, en seconde position
        vote_a_ajouter.suivant=tete_de_liste.suivant;
        tete_de_liste.suivant=vote_a_ajouter;
    }
}
```

2. Il suffit de faire comme dans le cours.

```
public static int TailleDeListe(Vote tete_de_liste)
{
    int taille=0;
    Vote parcours = tete_de_liste;

    while(parcours!=null)
    {
        taille=taille+1;
        parcours = parcours.suivant;
    }

    return taille;
}
```

3. On parcourt la liste, et on range chacun de ses éléments dans un tableau. Le tableau aura la même taille que la liste, dont on peut connaître la taille grâce à la fonction précédente.

```
public static Vote[] ListeVersTableau(Vote tete_de_liste)
{
    int t = TailleDeListe(tete_de_liste);
    Vote[] resultat = new Vote[t];

    Vote parcours = tete_de_liste;
    int compteur=0;

    while(parcours!=null) //On parcourt la liste
    {
        resultat[compteur] = parcours; //on place l'élément en cours dans le
tableau
        compteur=compteur+1; //on avance dans le tableau
        parcours = parcours.suivant; //et aussi dans la liste
    }

    return resultat;
}
```

4. On trie à l'aide de l'un des algorithmes de tri vu en cours. J'ai choisi le tri à bulle, que j'adapte au fait que je trie un tableau de Vote (et non un tableau de int).

```
public static void TriTableau(Vote[] tableauVote)
{
    int cpt, limite, i;
    Vote temp;

    limite=tableauVote.length-1;

    for(cpt=0; cpt<tableauVote.length-1; cpt=cpt+1)
    {
        for(i=0; i<limite; i=i+1)
        {
            if(tableauVote[i].adresse_votant > tableauVote[i+1].adresse_votant)
            {
                temp=tableauVote[i];
```

```
                tableauVote[i]=tableauVote[i+1];
                tableauVote[i+1]=temp;
            }
        }
        limite=limite-1;
    }
}
```

5. On parcourt le tableau de Vote et on lie chaque élément à la case suivante. On renvoie ensuite le premier élément du tableau, ça donne bien une liste chaînée.

```
public static Vote TableauVersListe(Vote[] tableauVote)
{
    for(int cpt=0; cpt<tableauVote.length-1; cpt=cpt+1)
    {
        tableauVote[cpt].suivant = tableauVote[cpt+1];
    }

    tableauVote[tableauVote.length-1].suivant=null;

    Vote tete_de_liste = tableauVote[0];
    return tete_de_liste;
}
```

6. Il suffit, comme dit dans l'énoncé, de combiner les trois fonctions que l'on vient de voir. Il faut juste penser au fait que la fonction devra renvoyer en sortie la nouvelle tête de liste.

```
public static Vote TriListe(Vote tete_de_liste)
{
    Vote[] tabListe = ListeVersTableau(tete_de_liste);
    TriTableau(tabListe);
    Vote nouvelle_tete = TableauVersListe(tabListe);
    return nouvelle_tete;
}
```

7.

```
public static void APerdu(Vote tete_de_liste)
{
    int[] resultats_vote = new int[5]; //Une case pour chaque candidat

    Vote parcours = tete_de_liste;
    while(parcours!=null) //On parcourt la liste
    {
        //On récupère le numero de candidat pour qui est le vote
        int num_candidat = parcours.vote_candidat;
        //et on met à jour le tableau de résultats de vote en ajoutant 1 dans la
bonne case
        resultats_vote[num_candidat] = resultats_vote[num_candidat] + 1;
        parcours = parcours.suivant; //et on avance dans la liste
    }

    //on parcourt le tableau resultats_vote pour trouver qui a obtenu le moins
```

```
de vote
    int position_min = 0;
    for(int cpt=1; cpt<resultats_vote.length; cpt=cpt+1)
    {
        if(resultats_vote[position_min] > resultats_vote[cpt])
        {
            position_min = cpt;
        }
    }

    System.out.println(position_min);
}
```

8. Pour saboter le jeu, on dirait que le suivant du dernier élément de la liste de Vote serait égal au premier élément : la liste n'a pas de fin, puisqu'elle boucle sur elle-même. La fonction de calcul des votes (APerdu) ne se terminera jamais ! mouhahaha (rire diabolique).