IN 101
7 janvier 2011

Contrôle

ENSTA ParisTech

NOM : **Prénom :**

- tous les documents (poly, slides, TDs, livres, brouillon du voisin...) sont **interdits**.
- les réponses doivent toutes tenir dans les cases prévues dans l'énoncé. Si cela ne suffit pas, utilisez la dernière page blanche (pas de feuilles supplémentaires).
- Dans l'exercice 2, le code des fonctions doit être donné en langage C. Dans le reste de l'examen, le code des algorithmes/fonctions pourra être donné soit en C soit en pseudo-code : la syntaxe n'a pas besoin d'être exacte, mais le code doit être présenté clairement (accolades, indentation, écriture lisible...). En pseudo-code, aucune syntaxe n'est imposée, mais tout devra être suffisamment détaillé pour ne pas laisser de place aux ambiguïtés (écrire "parcourir les sommets du graphe G" n'est pas assez précis, mais "parcourir les voisins du sommet S" est correct).
- le barème actuel de ce contrôle aboutit à une note sur 93 points. La note finale sur 20 sera dérivée de la note sur 93 grâce à une fonction croissante qui n'est pas encore définie (cela ne sert à rien de la demander).
- n'oubliez pas de remplir votre nom et votre prénom juste au dessus de ce cadre.

Exercice 1 : QCM (20 points)

Chaque bonne réponse rapporte 1 point. Chaque mauvaise réponse enlève 1 point. Si vous n'êtes pas certains de votre réponse, ne répondez pas au hasard, la note totale peut être négative !

1.a] Lequel des ces types n'est pas un type natif du langage C ?

☐ *int* ☐ *char* ☐ *node* ☐ *float*

1.b] Qu'affiche le programme :

```
1 int i=5;
2 for (i=0; i<7; i++) {
3   printf("%d,", i);
4 }
5 printf["%d.\n",i);
```

☐ 5,6,7,8. ☐ 0,1,2,3,4,5,6,7. ☐ 0,1,2,3,4,5,6,6. ☐ 5,6,8.

1.c] On veut utiliser `scanf` pour lire un entier sur l'entrée standard et le stocker dans la variable `n`. Quelle est la bonne syntaxe ?

☐ `scanf("%d") = &n;` ☐ `n = scanf("n=%d");`
☐ `scanf(n, "%d");` ☐ `scanf("%d", &n);`

1.d] On utilise un algorithme de tri à bulles pour trier un tableau de n chaînes de caractères de longueur 10 (plutôt que des entiers). Quelle est alors sa complexité ?

☐ $\Theta(n^{10})$ ☐ $\Theta(n \log_{10} n)$ ☐ $\Theta(10n^2)$ ☐ $\Theta(10^n)$

1.e] Une seule de ces fonction utilise de la récursivité *terminale* et pourra être spécialement optimisée par le compilateur, laquelle ?

☐
```
int f(int n) {
  if (n==0) {
    return 0;
  }
  printf("%d\n", f(n-1));
  return n;
}
```

☐
```
int f(int n) {
  if (n==0) {
    return 0;
  }
  printf("%d\n", f(n-1));
  return f(n) - 1;
}
```

☐
```
int f(int n) {
  if (n==0) {
    return 0;
  }
  printf("%d\n", n-1);
  return f(n) - 1;
}
```

☐
```
int f(int n) {
  if (n==0) {
    return 0;
  }
  printf("%d\n", n);
  return f(n-1);
}
```

1.f] La programmation dynamique sert en général à diminuer la complexité moyenne d'un algorithme récursif, mais augmente :

☐ son nombre d'appels récursifs ☐ sa complexité dans le pire cas
☐ sa complexité spatiale ☐ aucun des trois

1.g] Laquelle de ces complexités n'est pas polynomiale ?

☐ $\Theta(2^{(\log n)^2})$ ☐ $\Theta(2^{\log(\log(n))})$ ☐ $\Theta(2^{\log(n)})$ ☐ $\Theta(2^{\log(n^2)})$

1.h] On cherche à trier un tableau de n entiers dont seulement p éléments ne sont pas à leur place (si on retire ces p éléments, les $n-p$ restants sont bien dans l'ordre). Quelle est la complexité moyenne d'un tri par insertion sur un tel tableau ?

☐ $\Theta(n^2)$ ☐ $\Theta(p^2)$ ☐ $\Theta(np)$ ☐ $\Theta(p \log p)$

1.i] On cherche encore à trier un tableau de n entiers dont p éléments ne sont pas à leur place. Quelle est la complexité *dans le pire cas* d'un algorithme de tri rapide sur un tel tableau ?

☐ $\Theta(n^2)$ ☐ $\Theta(p \log n)$ ☐ $\Theta(n \log p)$ ☐ $\Theta(p \log p)$

1.j] Qu'affiche le code suivant quand on l'exécute ?

```
int i,j;
int** M = (int**) malloc(10*sizeof(int*));
M[0] = (int*) malloc(100*sizeof(int));
for (i=0; i<10; i++) {
  M[i] = &(M[0][10*i]);
  for (j=0; j<10; j++) {
    M[i][j] = i*j;
  }
}
printf("%d\n", M[2][34]);
```

☐ 68 ☐ 20
☐ 8 ☐ On ne peut pas savoir.

1.k] Un arbre binaire contenant 12 nœuds internes possède au minimum :

☐ 1 feuille ☐ 12 feuilles ☐ 13 feuilles ☐ 24 feuilles

1.l] Combien de feuilles comporte l'arbre représentant l'expression arithmétique $\frac{\log x+3}{x-1} + e^x$?

☐ 5 ☐ 6 ☐ 8 ☐ 11

1.m] On utilise une table de hachage pour faire du tri par paquets. Quelle propriété doit avoir la fonction de hachage f utilisée pour que ce tri soit efficace ?

- ☐ $\forall x$, $f(x)$ et $f(x+1)$ doivent être différents
- ☐ f doit être monotone
- ☐ $\forall x$, $f(x)$ doit être plus petit que x
- ☐ f doit être continue

1.n] On affiche les nœuds d'un arbre à l'aide de l'un des parcours suivants. Lequel ne permet pas de connaître de façon sûre la valeur de la racine ?

☐ en largeur ☐ préfixe ☐ infixe ☐ postfixe

1.o] On utilise une implémentation standard de tas, telle que vue en cours et en TD. On part d'un tas représenté par le tableau | 42 | 18 | 27 | 15 | 17 | 23 | | et l'on extrait sa racine 42. Quelle sera la nouvelle racine du tas ?

☐ 18 ☐ 27 ☐ 17 ☐ 23

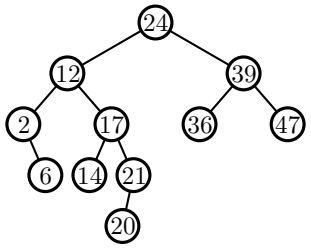


FIGURE 1 – Un arbre binaire de recherche.

1.p] L'arbre binaire de recherche de la FIGURE 1 n'est pas bien équilibré pour respecter les propriétés d'arbre AVL. Lequel des nœuds suivants possède une constante d'équilibrage qui ne convient pas ?

☐ 2 ☐ 17 ☐ 12 ☐ 24

1.q] On sait qu'une rotation ou double rotation suffit à rééquilibrer un arbre AVL après une insertion. C'est aussi le cas pour l'arbre de la FIGURE 1. Après rééquilibrage, quelle sera la nouvelle racine de cet arbre ?

☐ 24 ☐ 12 ☐ 17 ☐ 21

1.r] Laquelle de ces opérations est la plus coûteuse dans un graphe orienté à n sommets ?

- ☐ Un tri topologique
- ☐ Calculer les plus courts chemins d'un sommet à tous les autres
- ☐ Vérifier qu'il n'y a pas de cycles
- ☐ Calculer sa fermeture transitive

1.s] On recherche un motif de longueur m dans un texte de longueur n écrit avec un alphabet de p caractères. Quelle est la complexité totale, dans le pire cas, de la construction de l'automate approprié puis de la recherche du motif ?

☐ $\Theta(mn)$ ☐ $\Theta(mp+n)$ ☐ $\Theta(mnp)$ ☐ $\Theta(m+p+n)$

1.t] Un algorithme récursif de type « diviser pour régner » coupe un problème de taille n en deux sous-problèmes de taille $\frac{n}{2}$ avec un coût de division/recombinaison de $\Theta(n^2)$. Quelle est sa complexité totale ?

☐ $\Theta(n^3)$ ☐ $\Theta(n^2 \log n)$ ☐ $\Theta(n^2)$ ☐ $\Theta(n(\log n)^2)$

Exercice 2 : Programmation en C (20 points)

> Dans cet exercice il est demandé de répondre aux questions avec du vrai code C. La syntaxe doit être correcte, vous devez déclarer toutes vos variables et penser à **libérer** la mémoire dont vous n'avez plus besoin... Si vous voulez être certains d'obtenir tous les points, votre code doit pouvoir être compilé et exécuté sans erreurs.

(2 pts) **2.a]** Écrivez une fonction qui teste si un entier n est premier ou non en essayant de le diviser par tous les nombres plus petits que $\sqrt{n}$. Il est préférable de ne faire que des calculs entiers.

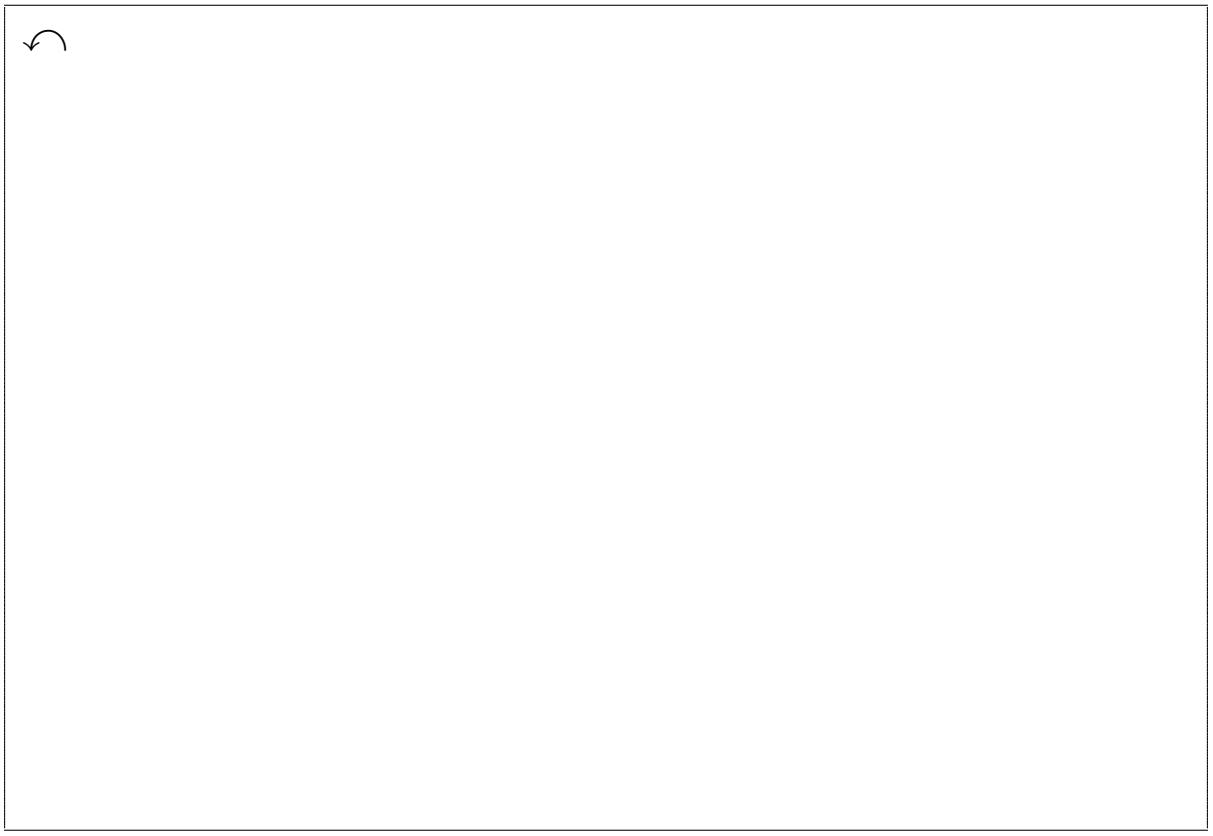
(4 pts) **2.b]** Écrivez une fonction `anagram` qui prend en argument deux chaînes de caractères (qui respectent bien la structure d'une chaîne de caractère du C) et retourne 1 ou 0 selon que ces deux chaînes de caractères sont des anagrammes l'une de l'autre (elles contiennent le même nombre de fois chaque caractère, mais dans un ordre qui peut être différent). Votre fonction doit avoir une complexité de $\Theta(n)$ pour des chaînes de caractères de longueur n.
⚠ Les arguments peuvent aussi contenir des caractères autres que les lettres de l'alphabet.

(3 pts) **2.c]** On définit une structure de nœud d'arbre binaire de la façon suivante :

```
1 typedef struct node_st {
2   int val;
3   struct node_st* left;
4   struct node_st* right;
5 } node;
```

Écrivez une fonction récursive `void print_and_free(node* A)` qui affiche l'arbre de racine `A` selon un parcours *infixe* et qui libère au passage la mémoire occupée par tous les nœuds de `A`.

(6 pts) **2.d]** On veut implémenter une *file* à l'aide d'une liste chaînée. Écrivez la (ou les) structure(s) de données que vous utiliseriez, ainsi que les fonctions `push` et `pop` qui permettent respectivement d'ajouter un élément à la file et d'en extraire un (en respectant l'ordre qui convient). Ces deux opérations doivent avoir une complexité en $\Theta(1)$. Faites bien attention à l'allocation/libération de la mémoire. Écrivez un `main` qui ajoute quelques éléments dans cette file puis les en extraits.



(5 pts) **2.e]** Écrivez une fonction *récursive* prenant en argument deux `int a` et `b` et qui affiche l'écriture de `a` en base `b`. Les valeurs possibles de `b` sont comprises entre 2 et 36 : les entiers de '0' à '9' suffisent pour les petites bases, et on les complète avec les lettres de 'a' à ''z' pour les bases plus grandes.
⚠ Cette fonction doit être récursive et afficher *un* caractère de l'écriture de `a` à chaque appel récursif. Faites bien attention à afficher `a` dans le bon ordre, avec les chiffres de poids fort en premier (à gauche), comme quand on écrit un nombre en base 10.

Exercice 3 : Recherche de cycles dans un graphe (15 points)

(1 pts) **3.a]** Soit M la matrice d'adjacence d'un graphe orienté G à n sommets. M^k est la puissance k-ième de cette matrice. Que représente $M^k_{(i,j)}$, le coefficient i, j de M^k, pour les sommets i et j du graphe G ?

(3 pts) **3.b]** Écrivez (en C ou en pseudo-code) un algorithme qui prend en entrée la matrice d'adjacence M d'un graphe, puis calcule les puissances de M (on suppose que l'on a accès à des fonctions permettant de faire des calculs matriciels : addition, produit...) et en déduit la longueur du plus court cycle du graphe correspondant à M.

(1 pts) **3.c]** On considère que le produit de deux matrices $n \times n$ coûte $\Theta(n^3)$. Quelle est la complexité dans le pire cas de l'algorithme précédent ?

(2 pts) **3.d]** L'algorithme de Aho-Hopcroft-Ullman permet lui aussi de rechercher les plus courts cycles d'un graphe. Expliquez ce que calcule cet algorithme et comment on peut l'utiliser pour connaître la longueur du plus court cycle dans un graphe. Quelle est sa complexité ?

(4 pts) **3.e]** On souhaiterait maintenant connaître le plus court cycle passant par un sommet donné du graphe. Écrivez un algorithme prenant en argument un graphe G (représenté sous forme de listes de successeurs) et un sommet s et qui affiche la longueur et l'ensemble des sommets du plus court cycle de G passant par le sommet s.

↷

(1 pts) **3.f]** Si le graphe possède $|S|$ sommets et $|A|$ arcs, quelle est la complexité dans le pire cas de l'algorithme précédent ?

(3 pts) **3.g]** On veut maintenant éliminer *tous* les cycles du graphe en utilisant un algorithme glouton qui supprime un arc à la fois. Expliquez comment vous choisiriez l'arc à supprimer afin d'essayer de minimiser le nombre total d'arcs à retirer. Quelle serait alors la complexité de votre algorithme d'élimination des cycles ?

Exercice 4 : Tri fusion avec des listes chaînées (10 points)

L'algorithme de tri fusion vu en cours utilise habituellement des tableaux, mais il peut aussi être implémenté avec des listes chaînées en gardant la même complexité (et sans recopier le contenu de la liste dans un tableau). C'est ce que nous allons faire dans cet exercice.

(2 pts) **4.a]** Expliquez brièvement le fonctionnement du tri fusion classique (avec des tableaux) et expliquez comment calculer sa complexité.

(2 pts) **4.b]** Écrivez une fonction (en C ou en pseudo-code) qui prend en argument une liste `L` et retourne deux listes (on supposera ici que `return` peut renvoyer plusieurs éléments, même si ce n'est pas le cas en C) : l'une correspondant à la première moitié de la liste `L` et l'autre à la deuxième moitié. Cette fonction ne doit jamais recopier les éléments de la liste `L` et la liste `L` initiale peut être modifiée. Cette fonction doit avoir une complexité linéaire en la taille de `L`.

(3 pts) **4.c]** Écrivez une fonction qui prend en argument deux listes triées et les fusionne en une seule liste triée (qu'elle renvoie à la fin). Encore une fois, cette fonction ne doit pas recopier les éléments des listes passées en argument, mais doit les modifier. Elle doit aussi avoir une complexité linéaire en la longueur totale des listes passées en argument.

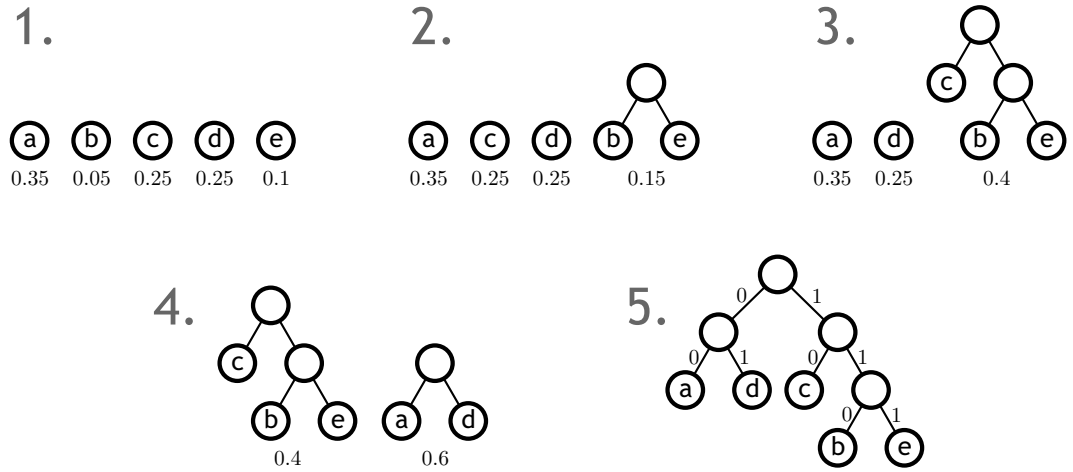
(3 pts) **4.d]** En utilisant les fonctions précédentes, écrivez un algorithme de tri fusion qui prend en argument une liste chaînée `L`, la trie, et retourne un pointeur sur le premier élément de la nouvelle liste triée.

Exercice 5 : Arbres de Huffman (15 points)

Les arbres de Huffman sont un outil de théorie de l'information permettant de faire de la compression sans pertes. Dans une chaîne de caractères en C, chaque caractère est codé par un octet : un motif de 8 bits. Cependant, quand on veut stocker un texte, certains caractères reviennent plus souvent que d'autres : plutôt que d'utiliser 8 bits pour représenter chaque caractère, mieux vaut avoir des séquences plus courtes pour les caractères les plus fréquents et des séquences plus longues pour les caractères qui n'apparaissent quasiment jamais. Les arbres de Huffman proposent une solution optimale pour ce type de codage : étant donné une fréquence d'apparition de chaque caractère, on construit un arbre qui va déterminer le codage de chacun des caractères de façon à ce qu'un texte qui respecte ces fréquences soit codé sur le moins de bits possible. L'arbre est construit de la façon suivante :

- Chaque caractère est transformé en un arbre composé d'une seule feuille représentant ce caractère, la fréquence associée à cet arbre est la fréquence du caractère.
- On stocke tous les arbres ainsi créés dans une structure adaptée.
- À chaque étape de l'algorithme, on extrait de la structure les deux arbres ayant la fréquence la plus petite, et on les fusionne en un nouvel arbre. L'opération de fusion consiste à créer un nouveau nœud dont les fils gauche et droit sont les deux arbres extraits. Le nouvel arbre ainsi créé est réinséré dans la structure avec pour fréquence la somme des fréquences des deux arbres qui le constituent.
- Quand il ne reste plus qu'un arbre dans la structure, l'algorithme s'arrête : c'est l'arbre de Huffman associé aux fréquences de départ.

Voici les étapes de la construction pour un alphabet de 5 caractères avec les fréquences suivantes : $(a, 0.35)$, $(b, 0.05)$, $(c, 0.25)$, $(d, 0.25)$ et $(e, 0.1)$.



Une fois l'arbre construit, le codage est déterminé en faisant un parcours de l'arbre : descendre dans le fils gauche correspond à un 0, descendre dans le fils droit à un 1. Ainsi, dans l'arbre que nous venons de construire le codage est le suivant : $a = 00$, $d = 01$, $c = 10$, $b = 110$ et $e = 111$. On construit donc une table de correspondance que l'on utilise pour coder notre texte. Par exemple, le texte *abdac* sera codé 00110010010.

Pour le décodage d'un texte, il suffit de lire la séquence binaire bit par bit : à chaque bit on descend à gauche ou à droite dans l'arbre et, dès que l'on atteint une feuille, on a un caractère du texte décodé et on repart de la racine de l'arbre.

(1 pts) **5.a]** On veut que le construction de l'arbre coûte $\Theta(n \log n)$ dans le pire cas pour un alphabet de n caractères. Quelle structure de donnée utiliseriez vous pour stocker les petits arbres pendant la construction ? Expliquez pourquoi la complexité de la construction est alors bien $\Theta(n \log n)$.

(4 pts) **5.b]** On suppose que la structure de donnée de la question précédente est implémentée et que les fonctions `insert` et `extract` permettent d'y insérer un arbre et d'en extraire l'arbre de fréquence la plus faible. Écrivez une fonction (en C ou en pseudo-code) qui prend en argument deux tableaux de longueur n (l'un contenant les caractères de l'alphabet et l'autre les fréquences associées à ces caractères) et qui retourne l'arbre de Huffman correspondant.

(4 pts) **5.c]** On veut maintenant pouvoir coder un texte composé de caractères de l'alphabet à l'aide de l'arbre construit. On suppose que l'on a accès à une fonction `char2int` qui prend en argument un caractère de l'alphabet et retourne un entier entre 0 et $n-1$. Écrivez une fonction qui prend en argument un arbre de Huffman (construit à la question précédente) et qui retourne un tableau de n cases dont la i-ème case contient le codage binaire du i-ème caractère (celui pour lequel `char2int` retourne i).

(2 pts) **5.d]** Écrivez une fonction qui prend en argument le tableau de codage de la question précédente et un tableau de n caractères (représentant le texte à codé) et qui retourne un tableau de bits correspondant au codage du texte (inutile de vous occuper de l'allocation mémoire pour ce tableau).

(4 pts) **5.e]** On veut maintenant décoder un texte binaire à l'aide de l'arbre de Huffman. Écrivez une fonction qui prend en argument un tableau de bits et un arbre de Huffman et retourne le tableau de caractère correspondant au texte décodé (inutile de vous occuper de l'allocation mémoire pour ce tableau).

Exercice 6 : Le naufrage du Titanic (6 points)

C'est le naufrage du Titanic, le nombre de canots de sauvetage est très limité et certaines personnes doivent passer avant d'autres : en particulier, il est impensable de sauver certaines personnes si d'autres ne le sont pas. De plus, le temps pour faire monter les gens sur les canots est très limité, et il faut donc optimiser le remplissage le plus vite possible.

On peut modéliser le problème du remplissage de la façon suivante : on se donne un ensemble de n personnes, et la personne i possède un poids w_i et une priorité p_i. Les canots peuvent contenir un poids total W. Si les personnes i et j ont des priorités telles que $p_i < p_j$, si l'on sauve i on doit aussi sauver j, et si l'on ne sauve pas j, on ne sauve pas i non plus. On cherche à trouver la solution optimale au problème (faire monter le maximum de personnes sans dépasser le poids W tout en respectant les contraintes de priorité) en temps $\Theta(n)$.

(3 pts) **6.a]** En vous inspirant de l'algorithme de tri rapide (un peu comme pour le calcul de la médiane d'un tableau) expliquez comment calculer le poids total des $\frac{n}{2}$ personnes de priorité maximale en temps linéaire (en moyenne).

(3 pts) **6.b]** Déduisez en un algorithme qui résout le problème et montrez qu'il a bien une complexité linéaire en n (en moyenne).

Exercice 7 : Un peu de dessin pour occuper le temps (7 points)

(3 pts) **7.a]** Dessinez un automate déterministe pour la recherche du motif *dbaddbc*. On veut un automate qui reste dans l'état final une fois le motif rencontré.

(4 pts) **7.b]** Dessinez un automate déterministe reconnaissant un polynôme en X. L'alphabet est constitué des symboles +, -, X, ^ et des chiffres de 0 à 9. Les polynômes reconnus doivent être, par exemple : -3, $1+X$, $X\hat{}2+3X\hat{}3-5X\hat{}4+2$ ou $-X\hat{}5+7X$. Plusieurs monômes de même degré peuvent apparaître dans le même polynôme.