X-Espci MP-PC 2006
Disque dur à 2 têtes
Correction Maple.

Question 1

Dans la procédure coutDe , r et t sont des listes , utilisées comme des tableaux à une entrée, ce qui allège les initialisations.

```
> restart:
> n:=3:rt:=[5,2,4]: #rt comme requêtes test
> coutDe:=proc(r,d) local pos,i,Ct;
> pos:=array(1..2,[0,0]); # position des 2 têtes
> Ct:=0;  #initialisation de la variable de coût
> for i from 1 to n do
>   if d[i]=1 then Ct:=Ct+abs(r[i]-pos[1]);pos[1]:=r[i]
>     else Ct:=Ct+abs(r[i]-pos[2]);pos[2]:=r[i]      fi; od;
> Ct
> end:
> d:=[1,1,2]:coutDe(rt,d):
> d:=[1,2,1]:coutDe(rt,d):
```

Question 2

Evident car intervertir l'ordre des têtes ne change une séquence de déplacement que par la transposition systématique (1,2).

Question 3

Pour un bloc de n requêtes on a évidemment 2^n séquences de déplacement possibles, et 2^(n-1) si l'on prend en compte la convention de la partie B

Question 4

(1,2) convient pour [10,3] et (1,1) convient pour [[3,10]. Remarquer que "donner une séquence à coût minimal " ne signifie pas prouver qu'ellel'est.

Question 5

Il n'y a que deux séquences possibles commençant par 1 et de longueur 2; il suffit de les comparer directement.

```
> n:=2:
> coutOpt2:=proc(r1,r2);
> if coutDe([r1,r2],[1,1]) <= coutDe([r1,r2],[1,2])
>      then [1,1] else [1,2] fi;
> end:
> coutOpt2(10,3):coutOpt2(3,10):
```

Remarquer que, ici, nous avons passé les requêtes en séquence et non en tableau; par contre, pour utiliser la procédure coutDe , nous reformons la liste des requêtes en tableau.

Question 6

Dans ceette question, on demande simplement d'appliquer la règle sur l'exemple, avec la convention que la première tête déplacée est la n°1.
Nous appliquons la règle Q5 à la séquence extraite [20,9] puis à la séquence extraite [9,1], en tenant compte des positions actualisées des deux têtes après

la première application de Q5. Ci-dessous, (p1,p2) représente l'ensemble des positions des deux têtes après chaque déplacement.
Le premier déplacement fournit (0,0) --> (20,0) ; or 20-9=11 alors que 9-0=9 d'ou (Q5) (20,0) --> (20,9) ; l'application répétée de (Q5) donne alors : (20,9) --> (20,1) . Le coût total est de 20+9+8=37 et la suite des déplacements (1,2,2).

Question 7
D'après la remarque faite en (Q3) il y a dans ce cas 4 stratégies possibles :
§ le déplacement (1,1,1) qui donne : (0,0) --> (20,0) --> (9,0) --> (1,0) de coût 39
§ le déplacement (1,1,2) qui donne : (0,0) --> (20,0) --> (9,0) --> (9,1) de coût 32
§ le déplacement (1,2,1) qui donne : (0,0) --> (20,0) --> (20,9) --> (1,9) de coût 48
§ le déplacement (1,2,2) qui donne : (0,0) --> (20,0) --> (20,9) --> (20,1) de coût 37
L'approche de la question (Q6) correspond au 4° déplacement; on constate que la solution optimale est donnée par le 2°.
Remarquer que la meilleure stratégie pour 3 requêtes n'utilise pas la meilleure stratégie pour 2 requêtes (ce qui était implicitement l'idée de (Q6)). En particulier, ceci interdit une approche "récursive élémentaire" pour trouver une solution optimale.

Question 8
Vu la remarque ci-dessus, nous allons nous contenter d'un balayage à 2 étapes.

```
> n:=3:
> coutOpt3:=proc(r1,r2,r3) local Ct,Opt,i,j,d,d1,r;
>    # d est un tableau représentant le déplacement "courant" et on
>    # stocke dans d1 le meilleur déplacement obtenu jusque là.
> Opt:=n*(r1+r2+r3); # initialisation à une valeur de toute façon trop
# grande
> r:=[r1,r2,r3]; # transformation des requêtes en un tableau (en fait une
# liste)
> for i from 1 to 2 do
>    for j from 1 to 2 do
>        d:=[1,i,j]; Ct:=coutDe(r,d);
>        if Ct<Opt then Opt:=Ct;d1:=d fi;
>    od;  od;
> d1,Opt
> end:
> coutOpt3(20,9,1):
```

Dans la partie finale, nous ne rremplacerons pas les tableaux carrés par des listes.
Question 9

i ou j doit être égal à n puisqu'il a fallu satisfaire la n° requête. Le coût optimal est donc la plus petite valeur rencontrée dans l'ensemble formé par la ligne (n+1) et la colonne (n+1).

Question 10
§ cout0(0,0)=0 par la convention de configuration initiale.
§ coutk(i,k) est atteint avec la requête rk satisfaite par la 2° tête, donc à partir d'une position (ri,rj) : ceci donne un coup de
|rk- rj|+cout(k- 1)(i,j) et l'on va prendre le minimum de ces valeurs sur j (j<=(k- 1)).
§ coutk(k,j)=coutk(j,k) car le tableau est nécessairement symétrique puisqu'on peut partir indifféremment de la première ou de la deuxième tête en effectuant systématiquement ensuite la transposition (1,2) des deux têtes.
§ coutk(i,j)=Infini si i<>k et j<>k car la dernière requête est rk et donc l'une au moins des 2 têtes doit être positionnée en rk. Ceci entraine que, dans la mise à jour, il faudra remettre à Infini une partie de la ligne et de la colonne indicées (k- 1).
Remarque : coutk(k,k)=Infini!

Question 11
Vu (Q10) , seule la (k- 1)° ligne de coutk- 1 est utilisée pour construire la k° . Il est également important de prendre en compte la remarque en italique qui termine le texte, laquell ne sert pas que pour la dernière question (elle me semble d'ailleurs assez mal placée).

```
> Infini=- 1:
> mettreAJour:=proc(cout,r,k) local i,j,aux,min;
>   #on construit la k°colonne
> for i from 0 to k- 2 do
>    cout[i,k]:=abs(r[k]- r[k- 1])+cout[i,k- 1] od;
> min:=r[k]+cout[k- 1,0];
> for j from 1 to k- 2 do
>   aux:=abs(r[k]- r[j])+cout[k- 1,j];
>   if aux<min then min:=aux fi; od;
> cout[k- 1,k]:=min;
>   #  puis on remet à Infini certains termes en ligne
>   #  et colonne (k- 1) et on symétrise
> for i from 0 to k- 1 do cout[i,k- 1]:=Infini;cout[k- 1,i]:=Infini od;
> for j from 0 to k- 1 do cout[k,j]:=cout[j,k] od;
> eval(cout)
> end:
```

Question 12
On construit le tableau coutn par itération de la procédure précédente, puis on renvoit le minimum des termes non Infini.

```
> coutOpt:=proc(r) local cout,i,j,k,min;
>   # initialisation du tableau cout
> cout:=array(0..n,0..n);
> for i from 0 to n do
>    for j from 0 to n do cout[i,j]:=Infini od;od;
> cout[0,0]:=0;
```

```
>   # boucle de mise à jour
> for k from 1 to n do cout:=mettreAJour(cout,r,k) od;
>   # recherche du minimum
> min:=cout[n,0];
> for i from 0 to n do
>    if (cout[n,i]<>Infini) and (cout[n,i]<min) then min:=cout[n,i] fi  od;  min
> end:
> n:=3:r:=array(0..3,[0,20,9,1]):coutOpt(r);
```

32

Temps d'exécution : boucle de k=1 à n sur mettreAJour qui emploie des doubles boucles : situation de boucle triple en O(n^3).

Question 13

Nous avons remarqué que seule la (k-1)° ligne était utilisée pour construire la k° ligne, et que le cout optimal était entièrement déterminé par le contenu de n° ligne du tableau coutn. Ceci permet de travailler sur une ligne unique que l'on va progressivement "décaler vers le bas" suivant le processus précédent (en utilisant toutefois en variable locale un "double" pour ne pas perdre les données antérieures indispensables).

Nous allons réécrire de façon plus simple la procédure "interne" mettreAJour puisqu'il n'est plus utile de remettre à Infini certains éléments, ni de symétriser, et surtout parce que la variable essentielle n'est plus un tableau carré mais une ligne.

```
> mettreAJourrapide:=proc(cout,r,k) local nvcout,i,j,aux,min;
> nvcout:=array(0..n);
> for i from 0 to k-2 do
>    nvcout[i]:=abs(r[k]-r[k-1])+cout[i]  od;
> min:=r[k]+cout[0];
> for j from 1 to k-2 do
>    aux:=abs(r[k]-r[j])+cout[j];
>    if aux<min then min:=aux  fi;  od;
> nvcout[k-1]:=min;
> eval(nvcout)
> end:
> coutOptrapide:=proc(r)  local  cout,i,k,min;
> cout:=array(0..n);cout[0]:=0;
> for k from 1 to n do cout:=mettreAJourrapide(cout,r,k)  od;
> min:=cout[0];
> for i from 0 to n-1 do
>    if cout[i]<min then min:=cout[i]  fi;  od;
> min
> end:
> n:=3:r:=array(0..3,[0,20,9,1]):coutOptrapide(r);
```

32

Temps d'exécution : boucle de k=1 à n sur mettreAJourrapide qui emploie une boucle simple : situation de boucle double avec temps en O(n^2).