# TD 05 – Allocation de la mémoire (ROM)

**Exercice 1.** *une windowserie de plus…*

Voici un aperçu écran de Windows Vista.

| | | |
|---|---|---|
| Type : | Disque local | |
| Système de fichiers : | NTFS | |
| Espace utilisé : | 85 455 966 208 octets | 79,5 Go |
| Espace libre : | 65 989 451 776 octets | 61,4 Go |
| | 151 445 417 984 octets | 141 Go |

1. Voyez-vous un problème ?

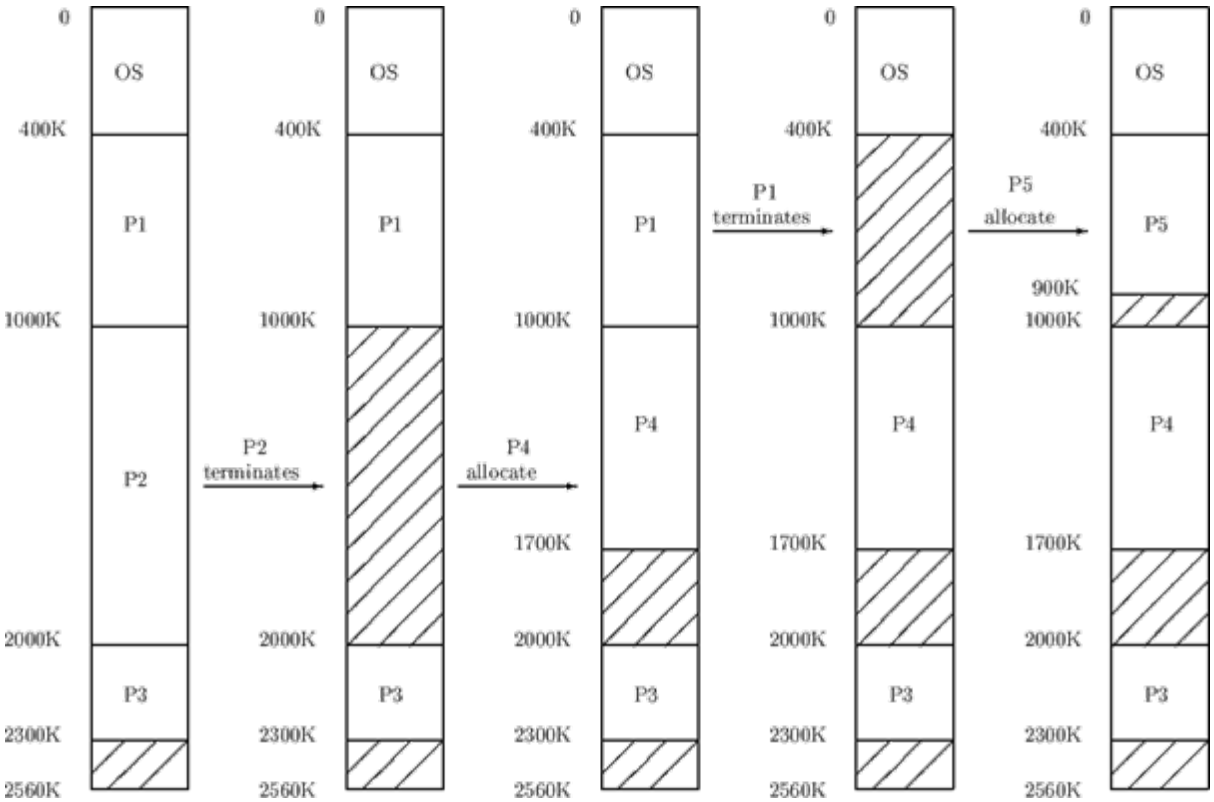
✎ Est-ce que l'un de vous a VISTA ou un autre Windows pour voir si il y a une confusion ?

**Exercice 2.** *fragmentation*

Voici une mesure de la fragmentation :

$$\text{fragmentation} = 1 - \frac{\text{plus grande zone de mémoire libre}}{\text{mémoire libre totale}}.$$

1. Mesurer la fragmentation de cette mémoire RAM à chacune des cinq étapes représentées.




2. Quelle(s) critique(s) peut-on faire à cette mesure ? Comment l'améliorer ?

**Exercice 3.** *résolution du chemin d'accès*

`tableau[i]` est la case `i` de `tableau`.
`tableau[i..j]` avec `i`$\leq$`j` désigne le sous `tableau` des cases `i` à `j`.
`tableau[i..]` désigne `tableau` à partir de la case `i`.
On peut tester si `tableau` est vide avec la fonction `est_vide(tableau)`.
Chaque fichier a un inode de type `inode`.
L'inode de la racine est `inode_racine`.
On peut tester les fichiers (entrées) de l'inode `fichier` avec la boucle

```
pour entree dans entrees_du_repertoire(fichier):
```

(à chaque itération, `entree` prend une nouvelle valeur de la liste des fichiers du répertoire `fichier`; si `fichier` n'est pas un répertoire alors aucun passage dans la boucle n'est effectué).
Une entrée comporte un champ `inode` désigné par `entree.inode`.
Une entrée comporte un champ `nom_du_fichier` désigné par `entree.nom_du_fichier`.

1. Écrire un algorithme récursif de résolution des chemins d'accès, dont le prototype est

   ```
   inode cherche_fichier(inode ou_je_suis, string[] chemin_restant).
   ```

   En cas de réussite, on retourne l'inode du fichier. En cas d'échec, on retourne `fichier introuvable`. Voici un exemple d'appel :

```
cherche_fichier(inode_racine,["home","kevin","sujet_du_prochain_examen.pdf"]).
```

**Exercice 4.** *table de blocs*

Pour chacun des ensembles de blocs *libres*, *occupés*, *défectueux*, on souhaite maintenir une table de bits $B$ telle que $B_i = 1$ si et seulement si le bloc $i$ appartient à l'ensemble.

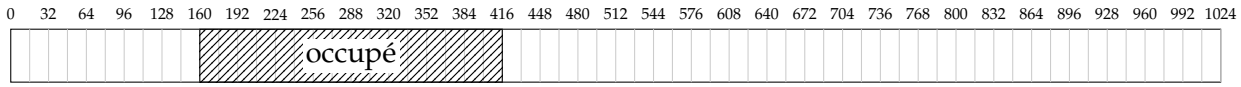
1. Calculer en kio la taille de chaque table pour un disque de 1 Gio et des blocs de 4 kio.
2. Calculer en ko la taille de chaque table pour un disque de 1 Go et des blocs de 4 ko.

**Exercice 5.** *SGF naïf*

Nous avons une mémoire de 1 Gio.

✎ Choisissons ensemble 4 demandes d'allocation de mémoire, dont les tailles sont des multiples de 32 et telles que la taille totale soit $22 \times 32 = 704$ Mio.

L'état de la mémoire est initialement le suivant (on négligera la taille de la table d'allocation) :




Vous pouvez utiliser la page 6 en fin de TD pour répondre à ces questions.

1. Appliquer l'algorithme First-fit pour la séquence de demandes :
   **(a)** demande 1 ;
   **(b)** demande 2 ;
   **(c)** libération 1 ;
   **(d)** demande 3 ;
   **(e)** libération 2 ;
   **(f)** demande 4.
2. Même question pour Best-fit et Worst-fit.
3. Quelle est la complexité de ces trois algorithmes en fonction du nombre $n$ de zones libres ?

**Exercice 6.** *buddy system $2^n$*

Nous avons une mémoire de 1 Gio, initialement complètement libre.
Vous pouvez utiliser la page 5 en fin de TD pour répondre à cette question.

1. Appliquer l'algorithme du *système du compagnon* pour la séquence de demandes :
   - **(a)** $A$ demande 68 Mio ;
   - **(b)** $B$ demande 132 Mio ;
   - **(c)** $C$ demande 70 Mio ;
   - **(d)** $D$ demande 134 Mio ;
   - **(e)** $B$ libère sa mémoire ;
   - **(f)** $D$ libère sa mémoire ;
   - **(g)** $E$ demande 400 Mio.

**Exercice 7.** *buddy system Fibonacci*

Vous pouvez utiliser la page 5 en fin de TD pour répondre à cette question.

1. Même exercice que précédemment, mais les tailles des zones suivent maintenant la suite de Fibonacci ($F_n = F_{n-1} + F_{n-2}$). On a initialement un bloc de taille $F_{16} = 987$. (0,)1,1,2,3,5,8,13,21,34,55,89,144,… (le découpage de $F_n$ donne $F_{n-1}$ et $F_{n-2}$)

**Exercice 8.** *FAT32*

Nous avons un disque dur avec un partition en FAT32 de 512 Gio et des blocs de 2 kio.

1. Quelle est la taille des adresses (toujours une puissance de 2) ?
2. Combien de blocs occupe la FAT ? Quelle est la place disponible pour l'utilisateur ?

Soit un fichier qui occupe 12 000 blocs : bloc 0 à bloc 11 999.

3. Combien de références pour accéder au bloc 1 ? 10 ? 100 ? 1 000 ? 10 000 ?

En FAT32, les adresses des blocs sont codées sur 32 bits, et la taille des blocs est limitée à 128 secteurs de 512 octets chacun. De plus, la taille d'un fichier est stockée sur 32 bits.

4. Quelle est la taille maximale d'une partition en FAT32 ?
5. Quelle est la taille maximale d'un fichier dans un SGF en FAT32 ?

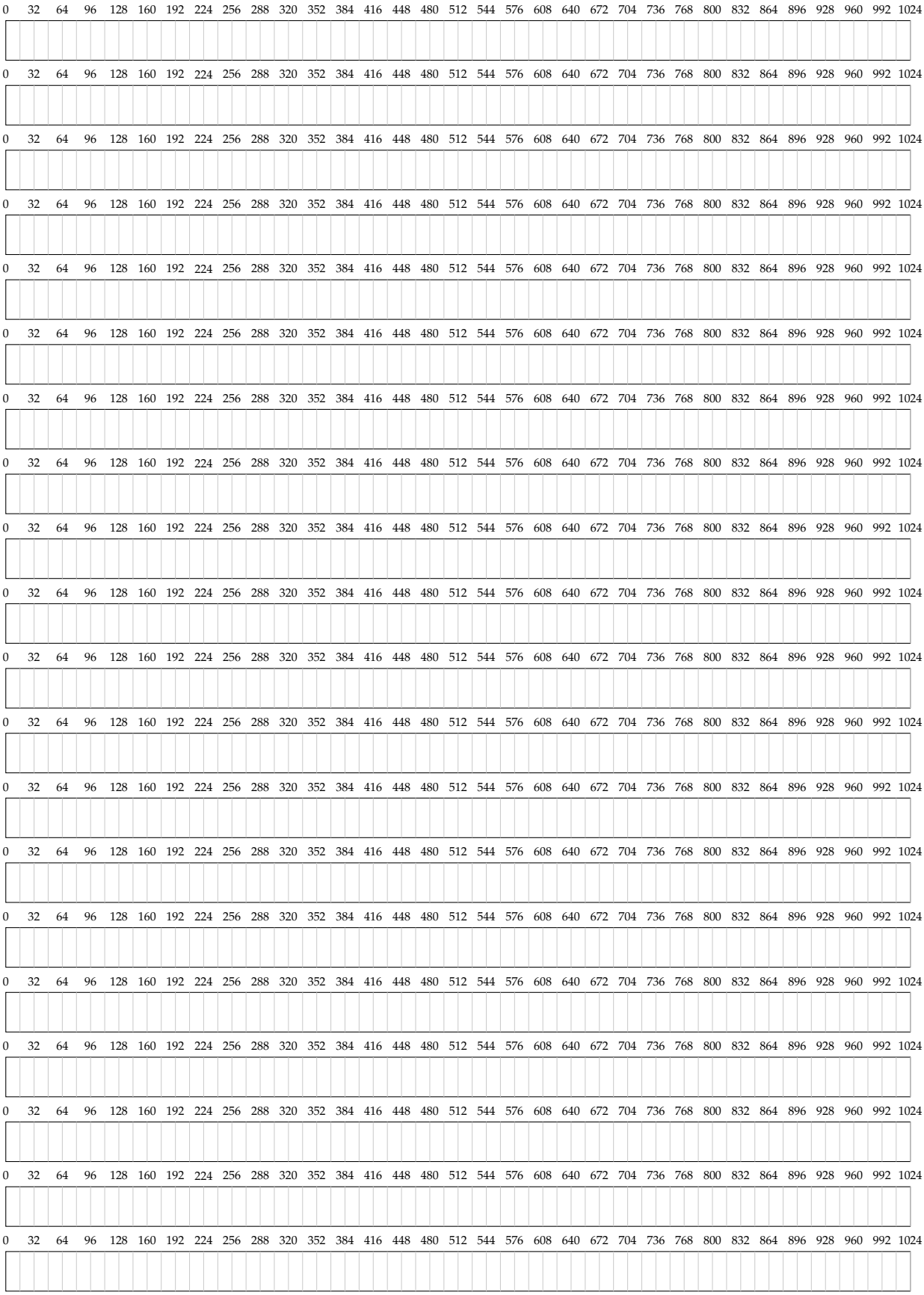
**Exercice 9.** *ext2*

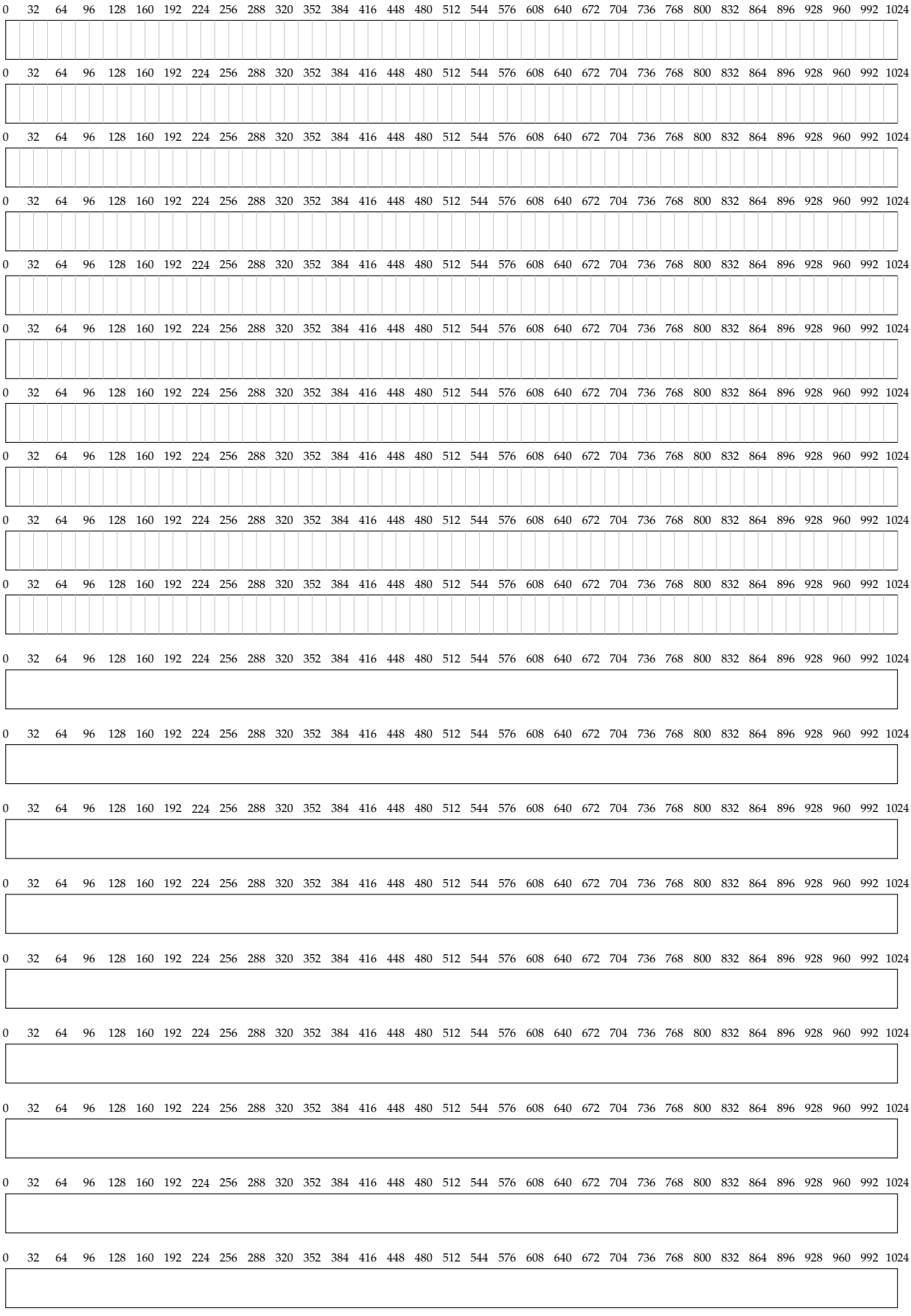
Les blocs ont une taille de 1024 octets, et leurs adresses sont codées sur 32 bits.

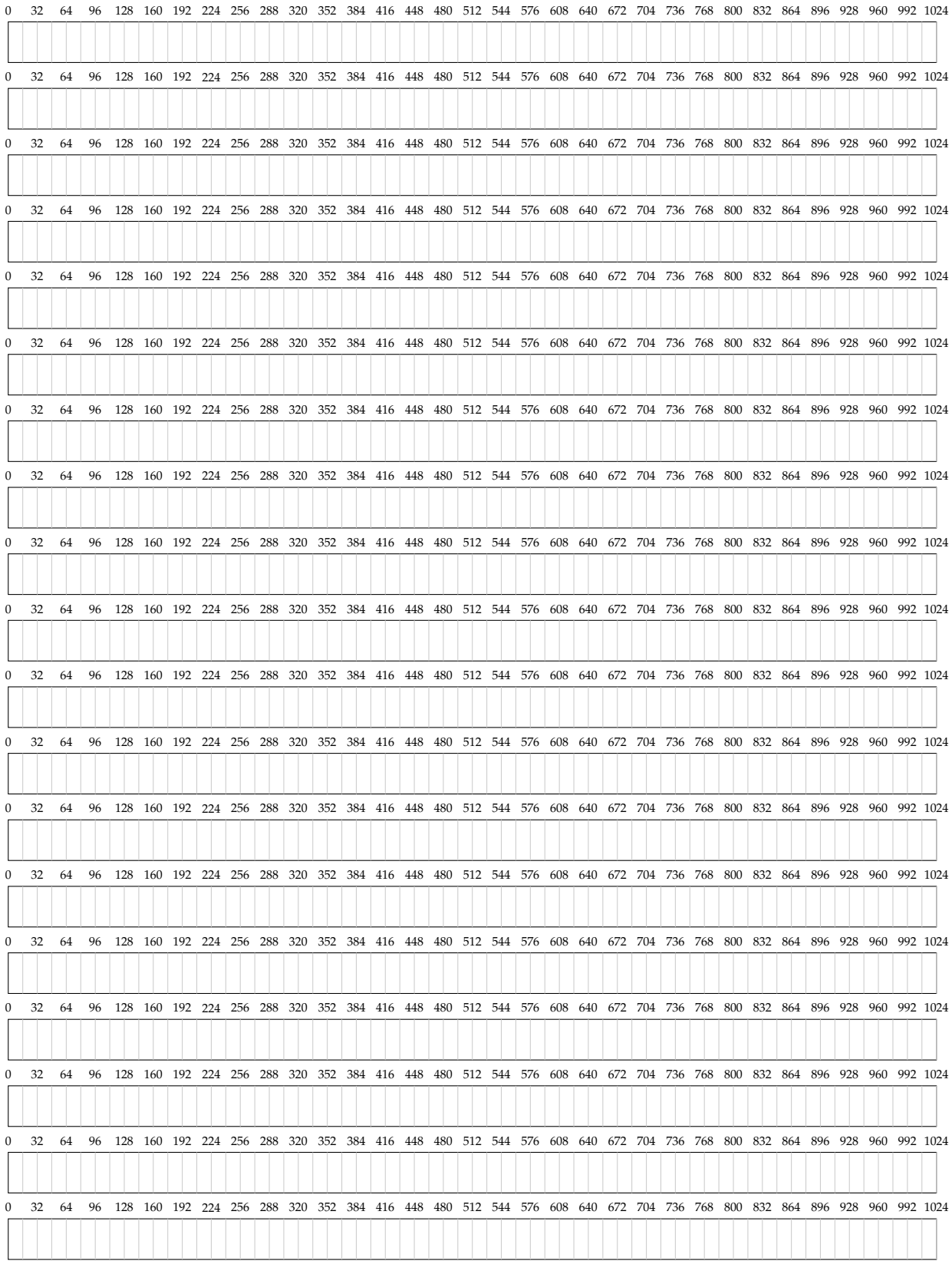
1. Combien un bloc d'indirection peut-il contenir d'adresses ?
2. Quelle est la taille maximale d'un fichier avec la triple indirection ?

Soit un fichier de 1 Gio. Les 64 champs de l'inode occupent 1 bloc.

3. Combien de blocs sont nécessaires au total (dont les tables) pour représenter ce fichier ?
4. Combien de références pour accéder au bloc 1 ? 10 ? 100 ? 1 000 ? 10 000 ? 100 000 ? 1 000 000 ?
5. Combien de blocs sont nécessaires au total pour représenter un fichier de 1 octets ?


0 32 64 96 128 160 192 224 256 288 320 352 384 416 448 480 512 544 576 608 640 672 704 736 768 800 832 864 896 928 960 992 1024


0 32 64 96 128 160 192 224 256 288 320 352 384 416 448 480 512 544 576 608 640 672 704 736 768 800 832 864 896 928 960 992 1024


0 32 64 96 128 160 192 224 256 288 320 352 384 416 448 480 512 544 576 608 640 672 704 736 768 800 832 864 896 928 960 992 1024