Structures de données
TD6 les tris

1 Exercice 1 Tri par insertion dichotomique

1.1 Fonction rang

On fait un appel récursif tant que $deb \neq fin$ ou que $deb \neq fin - 1$, c'est-à-dire tant que le tableau est plus grand que 1. On cherche donc à diviser par 2 le plus de fois possible tel que $\frac{n}{2^p} \leq 1 \Rightarrow p \geq \log 2(n)$. La fonction est donc en $O(\log 2(n))$.

1.2 Fonction insertion

```
Tableau* triInsertDicho(Tableau* t) // t est non trie
{
        Tableau* tri = (Tableau*)malloc(sizeof(Tableau));
        tri->tab = (int*)malloc(sizeof(int)*t->taille);
        tri->taille = t->taille;
        tri->free = 0;

        for(int i=0;i<t->taille;i++) {
                int elt = t->tab[i];
                if(tri->free == 0) {
                        // si premier element pas besoin de chercher le rang
                        tri->tab[0] = elt;
                        tri->free++;
                } else {
                        // on doit chercher le rang où placer elt
                        int indice = rang(tri, elt, 0, tri->free-1);
                        for(int j=tri->free; j > indice; j--) {
                                tri->tab[j] = tri->tab[j-1];
                        }
                        tri->free++;
                        tri->tab[indice] = elt;
                }
        }
        return tri;
}
```

La recherche (fonction rang) est en $O(\log 2(n))$ et l'insertion en $O(n)$ car dans le pire cas, il faut décaler tous les éléments vers la droite. On a donc $O(\log 2(n) + n) = O(n^2)$.

La recherche dichotomique ne peut êre appliquée à des listes.

2 Exercice 2 Tri postal

```
void triPostal(Tableau* t, int min, int max)
{
        int taille = max-min+1;
        int T = malloc(...);
        for(int i=0;i<taille;i++) {
                T[i] = 0;
        }
        for(int i=0; i<t->free; i++) {
                T[ t->tab[i] - min ]++;
        }
        int k =0;
        for(int i=0;i< t->free; i++) {
                for(int j=0;j<T[i];j++) {
                        t->tab[k] = i+min;
                        k++;
                }
        }
}
```

complexité de la fonction : $\Theta(free + (max - min))$

On ne peut pas avoir des valeurs quelconques. Il faut qu'elles soient assimilées à des entiers avec une fonction de hachage (pour calculer l'indice) et il faut que (max-min) soit petit pour tenir en mémoire.

3 Exercice 3 Trier pour être efficace

3.1 Trouver deux entiers dont la somme vaut B

On peut faire cela naïvement avec une double boucle parcourant chacune le tableau et testant si $T[i] + T[j] == B$. Cependant on a une complexité en $O(n^2)$.

Pour être plus efficace, on peut faire un algorithme composé de trois étapes :

- Trier le tableau $\rightarrow$ complexité en O(nlog $n) Parcourirletableauàl'aided'uneboucle \rightarrow$ complexité en $O(n)$

— chercher si le tableau contient la valeur $B - T[i]$ $\rightarrow$ complexité d'une recherche dichotomique dans un tableau trié : $O(\log n)$

Ce deuxième algorithme est donc en $O(n \log n)$

3.2 Appartements

— Comparer chaque appartement avec a1 $\rightarrow O(n)$
— Trouver l'appartement avec le plus petit loyer $\rightarrow O(n)$
— Trier selon un critère et comparer selon l'autre critère $\rightarrow O(n \log n)$

3.3 Tâches

La meilleure solution est 8. Il faut trier les tâches par durée croissante ($O(n \log n)$) et les ordonnancer comme cela.