complexité
Examen

La sélection du k^{ième} élément dans un tableau de *n* éléments est définie comme étant la recherche de l'élément *x* tel que *k-1* éléments du tableau sont inférieurs ou égaux à *x* et les autres éléments sont plus grand ou égaux à *x*. Si *k* vaut 1 il s'agit de la recherche du plus petit élément, si *k* vaut *n* c'est la recherche du plus grand.

Question 1. (5 points)

Une première idée d'algorithme consiste à trier le tableau et à renvoyer la valeur se trouvant en k^{ième} position. Programmer cet algorithme, et donner le coût de l'algorithme en fonction de *n*.

Question 2. (5 points)

Nous considérons la fonction *int partition (int t[], int d, int f)* qui partitionne la partie du tableau *t [d, f[* en deux parties telles que :

- La partie *[d, m[* des éléments inférieurs ou égaux à *pivot*.
- La partie *[m, f[* des éléments supérieur ou égaux à *pivot*.
- Chaque partie contient au moins un élément.

```
int partition( int t[], int d, int f) {
      int pivot = t[d];
      int i = d-1;
      int j = f;
      while(true){
         do{ j = j-1;} while(t[j] > pivot);
         do{ i = i+1;} while(t[i] > pivot);
         if( i<j) {
            int a = t[i];
            t[i]=t[j]; t[j]= a;
         } else  return j;
      }
  }
```

Quel est le nombre de comparaisons entre éléments de tableau effectuées par une partition d'une partie de tableau à *n* éléments.

Question 3. (7 points)

On peut alors programmer la recherche du $k^{\text{ième}}$ élément de la façon suivante :

```
int kieme( int t[] , int d, int f, int k) {
     if(d==f-1) return d; // il y a un seul élément c'est le bon !
     else{
         int m = partition(t, d, f);
         if(k<=m) return kieme (t, d,   m+1, k);
         else     return kieme (t, m+1, f,   k);
     }
  }
```

Dans quel(s) cas cet algorithme est-il le plus lent, et quel est le nombre de comparaisons effectuées dans ce cas.

Dans l'hypothèse où la partition calcule un indice *m* qui est toujours le milieu de la partie, et si le nombre d'éléments de la partie est *n*, calculer le nombre de comparaisons effectuées entre éléments du tableau.

Question 4. (5 points)

La qualité de la partition dépend du choix du pivot. Le choix idéal pour la partition serait le choix de l'élément médian (celui qui serait au milieu si le tableau était trié). Pour assurer que la partition choisisse un pivot qui ne soit pas « trop mauvais » on pourrait utiliser l'idée suivante : on groupe les éléments par 3 et on garde l'élément médian des trois. On recommence l'opération jusqu'à obtenir 1 seul élément. Programmer cet algorithme en supposant que *n* est une puissance de 3 et estimer le temps de calcul.