TD 2 : Evaluer des algorithmes, concevoir des algorithmes de tri

1 Rappels : quelques algorithmes de recherche (vus au TD1)

(A) un algorithme de recherche dans un tableau non trié

```
public static int recherche(int cherche, int[] t){
  int index=-1;
  for (int i=0; i< t.length;i++){
    if(t[i]== cherche) {
      index=i;
    }
  }
  return index;
}
```

(B) un algorithme de recherche dans un tableau trié

```
public static int rechercheTableauTrie(int cherche, int[] t){
  int index=-1;
  boolean trouve=false;
  int i=0;
  while(!trouve && i<t.length && t[i]<=cherche){
    if(t[i]== cherche) {
      index=i;
      trouve=true;
    }
    i++;
  }
  return index;
}
```

(C) un algorithme de recherche dichotomique dans un tableau trié

```
public static int rechercheDicho(int cherche, int[] t){
  int debut=0;
  int fin=t.length-1;
  boolean trouve=false;
  int milieu= (debut+fin)/2;
  while(!trouve && debut<=fin){
    milieu= (debut+fin)/2;
    if (t[milieu]==cherche)
      trouve=true;
    else if(t[milieu]>cherche)
      fin=milieu-1
    else debut=milieu+1;
  }
  if (trouve)
    return milieu;
  else return -1;
}
```

2 Estimer la complexité au pire cas d'un algorithme

1. ⋆ Donner une estimation de la complexité de l'algorithme de recherche (A) dans le pire des cas. **Justifier** le raisonnement à l'aide de la méthodologie présentée en cours.
2. ⋆⋆ Donner une estimation de la complexité de l'algorithme de recherche (B) dans le pire des cas. **Justifier** le raisonnement à l'aide de la méthodologie présentée en cours.
3. ⋆⋆⋆ Donner une estimation de la complexité de l'algorithme de recherche (C) dans le pire des cas.
4. ⋆⋆ Mêmes questions pour les algorithmes conçus au TD1. **Justifier** le raisonnement à l'aide de la méthodologie présentée en cours.

3 Concevoir des algorithmes de tri

Pour comprendre le fonctionnement des différents algorithmes, il est possible de les simuler en "informatique débranchée". Par exemple, on peut manipuler à la main un jeu de cartes à trier, et appliquer l'algorithme sur le paquet de cartes.

1. ⋆⋆ Proposer un algorithme de tri pour un tableau `int[] t`, par **sélection** du minimum (tri sélection).
   — dérouler l'algorithme sur un exemple
   — déterminer des valeurs pour une exécution dans le pire des cas
   — estimer la complexité de l'algorithme dans le pire des cas. **Justifier** le raisonnement à l'aide de la méthodologie présentée en cours.
2. ⋆⋆⋆ Proposer un algorithme de tri pour un tableau `int[] t`, par **construction incrémentale d'un tableau ordonné** (tri insertion)
   — dérouler l'algorithme sur un exemple
   — déterminer des valeurs pour une exécution dans le pire des cas
   — estimer la complexité de l'algorithme dans le pire des cas. **Justifier** le raisonnement à l'aide de la méthodologie présentée en cours.
3. ⋆⋆ Proposer un algorithme de tri pour un tableau `int[] t`, par **permutation de 2 voisins non ordonnés** (tri bulle).
   — dérouler l'algorithme sur un exemple
   — déterminer des valeurs pour une exécution dans le pire des cas
   — estimer la complexité de l'algorithme dans le pire des cas. **Justifier** le raisonnement à l'aide de la méthodologie présentée en cours.
4. ⋆⋆⋆⋆ Proposer un algorithme de tri pour un tableau `int[] t`, utilisant le principe **diviser pour régner** (tri rapide ou tri fusion).
   — dérouler l'algorithme sur un exemple
   — déterminer des valeurs pour une exécution dans le pire des cas
   — estimer la complexité de l'algorithme dans le pire des cas