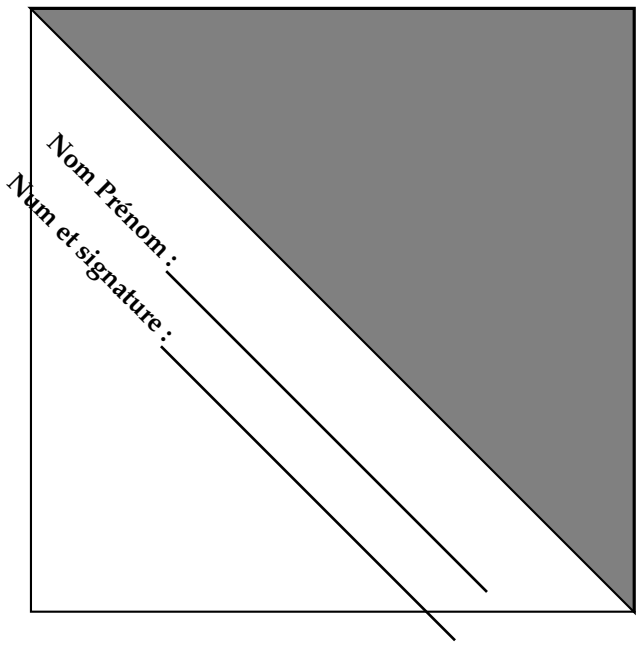
Université Claude Bernard, Lyon 1
Licence Sciences et Technologies - L3
Algorithmique, Programmation et Complexité - LIFAP6 –

Interrogation

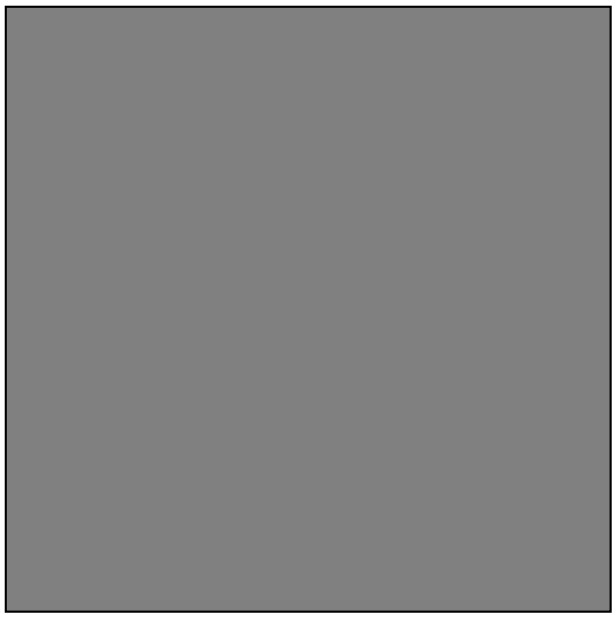


Durée : 1h30 minutes
Documents, calculatrices et téléphones portables non autorisés.

Note :

Attention :

- La qualité de l'**écriture** et de la **présentation** sera prise en compte (utilisez votre brouillon pour composer vos réponses).
- Il sera attaché plus d'importance à la **lisibilité** et à la **justesse** des algorithmes qu'à leur rigueur syntaxique.
- Ne pas oublier de **commenter** les algorithmes, en signalant les résultats, préconditions et postconditions s'il y a lieu. Ne pas oublier non plus de **déclarer** les variables auxiliaires utilisées. Soigner les en-têtes de sous-programmes : noms, types et modes de transmission des paramètres, entre autres.
- Il est conseillé de lire **tout** le sujet avant de commencer à rédiger.



Faire le tri

1. Faites la liste de l'ensemble des algorithmes de tri rappelés ou présentés dans l'UE LIFAPC, en donnant pour chacun sa complexité dans le meilleur des cas, le cas le pire et le cas moyen en fonction du nombre d'éléments. On précisera également pour chaque tri s'il s'agit d'un tri sur place dans le cas où on l'applique à un tableau.

Liste triées

On travaille ici sur un module $ListeTriee$ correspondant aux structures de données suivantes (le type $Element$ est importé et dispose d'opérations de comparaison qui seront utilisées pour maintenir la liste triée dans l'ordre croissant au fur et à mesure des insertions).

```
struct Cellule {
        Element info;
        Cellule* suivant;
};

struct ListeTriee {
       Cellule* adPremiereCellule;
};
```

On suppose également que la $ListeTriee$ est implantée à l'aide d'une $Cellule$ sentinelle dont le champs $info$ n'est pas pertinent mais dont le champs $adPremiereCellule$ contient l'adresse de la vraie première $Cellule$ (ou l'adresse nulle si la liste est vide). La procédure d'initialisation d'une $ListeTriee$ en liste vide est donc la suivante :

```
void initialiseListeTrieeVide(ListeTriee & l){
 l.adPremiereCellule = new Cellule;
 l.adPremiereCellule->suivant = nullptr;
}
```

2. Dessinez une $ListeTriee$ *lili* après initialisation par cette procédure.

On considère que le module offre également une procédure d'insertion d'un $Element$ dans une $ListeTriee$.

```
void insereElementDansListeTriee(ListeTriee & l, const Element & e);
```

3. Dessinez la $ListeTriee$ *lili* après exécution du programme suivant dans lequel le type $Element$ correspond au type int (avant l'appel à $testament(lili)$).

```
ListeTriee lili;
initialiseListeTrieeVide(lili);
insereElementDansListeTriee(lili, 3);
insereElementDansListeTriee(lili, 15);
insereElementDansListeTriee(lili, 1);
```

4. Donnez le code de la procédure de recherche d'un $Element$ dans une $ListeTriee$ (on ne demande pas d'insérer l'$Element$). Cette procédure renvoie l'adresse de la $Cellule$ contenant l'$Element$ si il est présent, le pointeur nul sinon. Quelle est sa complexité ?

```
Cellule * rechercheElementDansListeTriee(ListeTriee & l, const Element & e);
```

SkipList

5. Expliquez le principe suivi par les skip-lists pour faire tomber la complexité de recherche d'un $Element$ dans une $ListeTriee$.

6. Quelle évolution de la structure de donnée $ListeTriee$ correspond à la structure de donnée $SkipList$. Donnez cette structure de données en ajoutant toute explication nécessaire.

7. Dessinez une $SkipList\ lili$ initialisée comme étant une $SkipList$ vide.

8. Dessinez la $SkipList\ lili$, après insertion des valeurs 3 (tirages au sort associés : $Pile$, $Pile$, $Face$), 15 (tirages au sort associés : $Face$) et 1 (tirages au sort associés : $Pile$, $Face$).

9. Donnez le code de la procédure de recherche d'un élément dans une $SkipList$ (on ne demande pas d'insérer l'$Element$). Cette procédure renvoie l'adresse de la $Cellule$ contenant l'$Element$ si il est présent, le pointeur nul sinon.

```
Cellule * rechercheElementDansSkipList(SkipList & l, const Element & e);
```

Kième

On dispose d'un tableau T de n éléments dont on veut calculer le Kième élément dans l'ordre croissant, sans avoir à trier l'ensemble du tableau. Etant donné la valeur v d'un élément pivot pris au hasard dans le tableau T, on suppose que l'on dispose d'une procédure capable de remplir un tableau T' avec les $n1$ éléments de T plus petits que v (leur ordre est sans importance), puis les $n2$ éléments égaux à v, puis les $n3$ éléments plus grands que v.

```
procédure Partitionne(donnée
                                      debut, fin : [1..n],
                                      T : Tableau [debut..fin] de Element,
                                résultat
                                      T' : Tableau [debut..fin] de Element,
                                      n1, n2, n3 : 1..n)
```

10. Donnez le pseudo-code d'une fonction récursive $selectKieme$ qui renvoie la valeur du Kième élément d'un tableau de taille n (pour $K \leq n$). On utilisera pour cela la procédure $Partitionne$ et on effectuera l'appel récursif sur une des 3 parties du tableau T' résultant de la partition.