

[CYBER1][2024-2025] Partiel (Sujet A) CORRECTION
(2h00)
Algorithmique 1

NOM : _____

PRÉNOM : _____

Vous devez respecter les consignes suivantes, sous peine de 0 :

- Lisez le sujet en entier avec attention
- Répondez sur le sujet
- Ne détachez pas les agrafes du sujet
- Écrivez lisiblement vos réponses (si nécessaire en majuscules)
- Écrivez lisiblement votre nom et votre prénom sur la copie dans les champs prévus au dessus de cette consigne
- Ne trichez pas

1 Questions (sur 6 points)

- 1.1 (3 points)** En admettant que l'on dispose d'une pile vide et que les éléments « 1 2 3 4 5 6 » arrivent en entrée dans cet ordre exclusivement, et qu'à chaque fois qu'un élément est sorti, celui-ci est affiché, décrivez les scénarios permettant d'obtenir les sorties suivantes :

*exemple : pour « A B C » en entrée, on peut obtenir « B C A » en sortie en faisant :
 « push A », « push B », « pop », « push C », « pop », « pop »*

3, 2, 4, 1, 6, 5

- Push 1
- Push 2
- Push 3
- Pop
- Pop
- Push 4
- Pop
- Pop
- Push 5
- Push 6
- Pop
- Pop

2, 3, 1, 4, 6, 5

- Push 1
- Push 2
- Pop
- Push 3
- Pop
- Pop
- Push 4
- Pop
- Push 5
- Push 6
- Pop
- Pop

1, 4, 3, 2, 5, 6

- Push 1
- Pop
- Push 2
- Push 3
- Push 4
- Pop
- Pop
- Pop
- Push 5
- Pop
- Push 6
- Pop

1.2 (1 point) Donnez les caractéristiques de cette file dans chaque principe de fonctionnement :

Le but de l'exercice est d'illustrer les différences produites entre les 2 principales logiques d'implémentation des files

33	15	42	60	79	27
0	1	2	3	4	5

Tail = 4

En admettant que la queue de la file (Tail) est initialisée à la case « -1 » lorsqu'elle est vide, indiquez :

Taille maximale de la file : 6 Élément qui sera effacé lors du prochain *enqueue* : 27

Longueur utilisée de la file : 5 Prochain élément défilé : 33

En admettant que la queue de la file (Tail) est initialisée à la case « 0 » lorsqu'elle est vide, indiquez :

Taille maximale de la file : 6 Élément qui sera effacé lors du prochain *enqueue* : 79

Longueur utilisée de la file : 4 Prochain élément défilé : 33

1.3 (2 points) Donnez l'état final des tableaux sous-jacents aux structures de données vues en cours suite aux appels successifs suivants :

⇒ Push D, Push A, Push B,
 Pop, Push C, Pop,
 Push N, Push B, Pop,
 Pop, Push N, Push C,
 Push F, Pop, Push E. ⇒

⇒ Enqueue D, Enqueue U,
 Enqueue B, Dequeue,
 Enqueue D, Enqueue I,
 Enqueue N, Dequeue,
 Dequeue, Enqueue O. ⇒

D	A	N	C	E
0	1	2	3	4

D	I	N	O	(O)
0	1	2	3	4

2 Algorithmes (sur 14 points)

2.1 (3 points) Expliquez un des algorithmes de tri de votre choix, puis écrivez son implémentation sous forme de code.

Explications : (1 points)

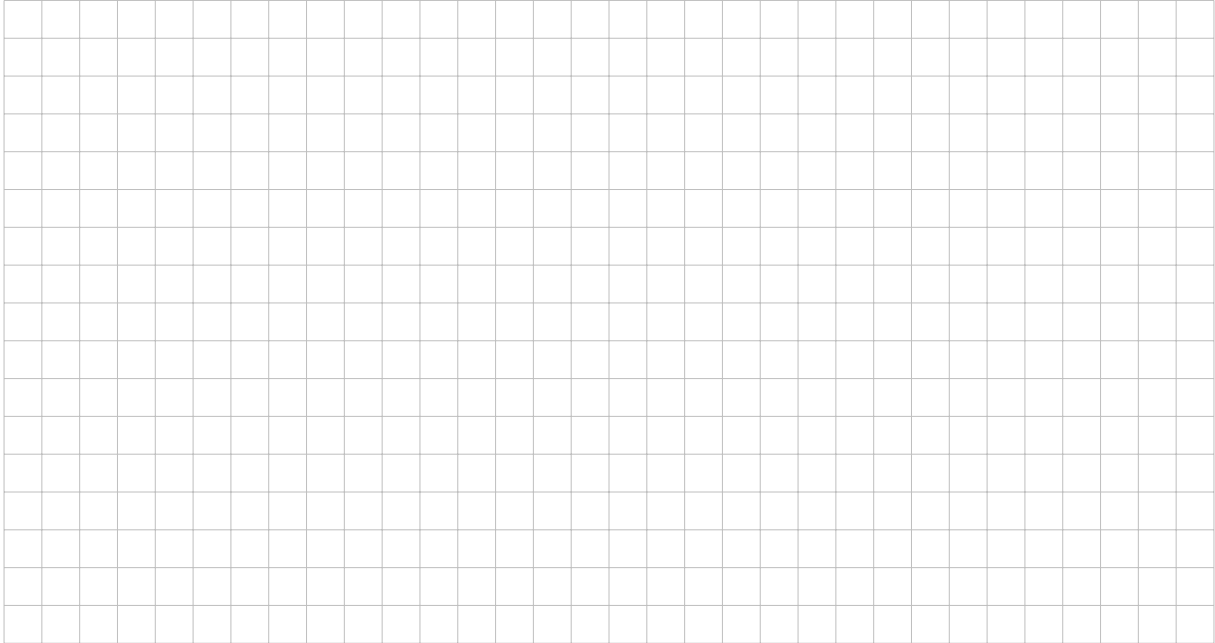
[illegible]

Implémentation : (2 points)

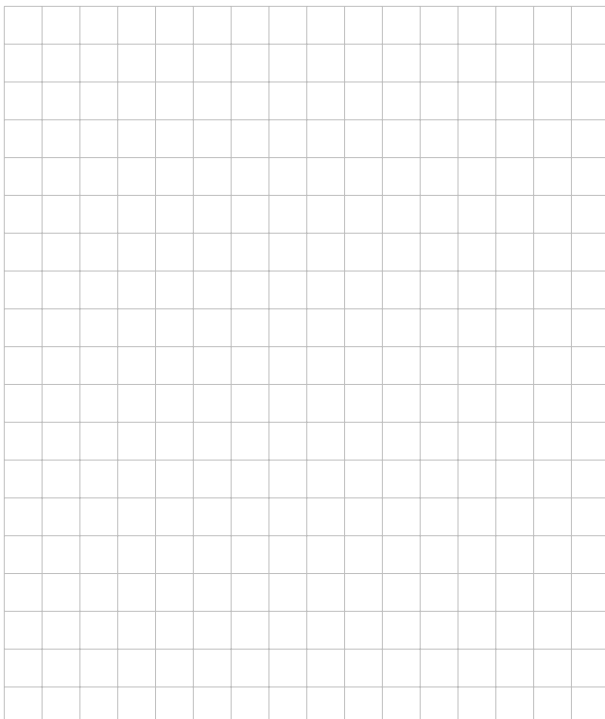
2.2 Pile (sur 7 points)

Le but de cette partie sera de réécrire quelques fonctions essentielles permettant d'utiliser une pile. Vous écrirez d'abord la structure en vous appuyant sur un modèle à base de tableaux, puis, vous implémenterez les fonctions avec cette structure.

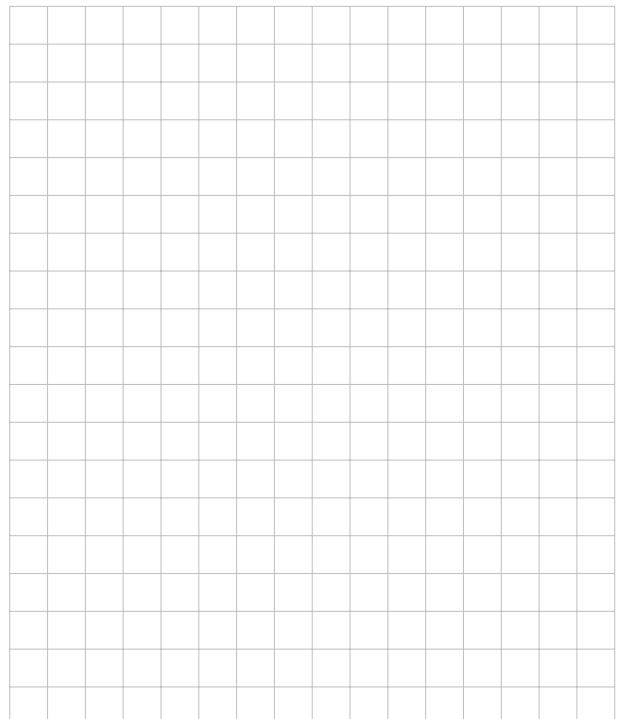
2.2.1 (1 point) Écrivez une structure de données « *stack_t* » pouvant servir de pile contenant des éléments étant des entiers positifs non-nuls. Cette pile doit obligatoirement s'appuyer sur un tableau.



2.2.2 (1 point) Écrivez une fonction « *length* » qui détermine la taille d'une pile.



2.2.3 (1 point) Écrivez une fonction « *is_empty* » qui détermine si une pile est vide ou non.



2.2.4 (2 points) Écrivez une fonction « *push* » qui empile un élément dans une pile.

2.2.5 (2 points) Écrivez une fonction « *pop* » qui dépile un élément d'une pile.


2.3 File avec une liste (sur 4 points)

Le but de cette partie sera d'implémenter quelques fonctions essentielles permettant d'utiliser une file. Nous disposons déjà d'une implémentation de la structure de liste, fournie par l'API ci-dessous. Vous utiliserez donc les fonctions de cette API pour réécrire les fonctions associées à la file.

API Liste :

- **list_t *CreateListT(int max_len)** : crée une liste vide de taille maximale *max_len*
- **bool InsertListT(list_t *l, int index, int elt)** : insère l'élément *elt* à la position *index* dans la liste *l* (en poussant vers la droite les éléments existants)
- **bool RemoveListT(list_t *l, int index)** : supprime l'élément en position *index* de la liste *l*
- **int LengthListT(list_t *l)** : renvoie la taille de la liste *l* (c'est-à-dire le nombre d'éléments présents)
- **int GetPositionListT(list_t *l, int elt)** : renvoie l'index du premier élément *elt* trouvé dans la liste *l* (si l'élément n'est pas trouvé, la fonction renvoie -1)
- **int GetEltListT(list_t *l, int index)** : renvoie l'élément présent en position *index* dans la liste *l* (si la position est incorrecte, la fonction renvoie -1)
- **bool IsEmptyListT(list_t *l)** : teste si la liste *l* est vide ou non
- **bool IsFullListT(list_t *l)** : teste si la liste *l* est pleine ou non
- **int ClearListT(list_t *l)** : vide la liste *l* et renvoie le nombre d'éléments supprimés
- **void DeleteListT(list_t *l)** : supprime la liste *l*

2.3.1 (2 points) Écrivez une fonction « *enqueue* » qui enfile un élément dans une liste utilisée comme une file.



2.3.2 (2 points) Écrivez une fonction « *dequeue* » qui défile un élément d’une liste utilisée comme une file.

N'oubliez pas d'écrire votre nom et votre prénom sur la 1^{ère} page.

CORRECTION SUJET A

ALGORITHMIQUE 1