Tous les documents papiers sont autorisés. Les documents électroniques sont interdits, les ordinateurs, tablettes, téléphones, montres connectés, écouteurs, et autre appareils intelligents doivent être rangés dans les sacs.

Conseils: Cet examen est constitué de 4 exercices de difficulté croissante. Les questions de programmations sont des programmes courts (pas plus d'une vingtaine de ligne), et vous ne serez pas évalués sur l'exactitude de la syntaxe, tant que vos programmes ressemblent suffisamment à du C++. Pour les questions d'analyses de complexité, il vous est demandé de justifier du mieux possible. Vous pouvez indiquer des numéro de lignes, ou mettre des marqueurs dans vos programmes, pour vous aider à désigner un endroit précis. Dans les 4 exercices, vous êtes guidés sur ce qu'il faut faire, et il n'est pas attendu que vous introduisiez vous-même des fonctions ou des arguments auxiliaires. L'examen contient également une question bonus et un exercice bonus qui sont hors barême, **donc à ne regarder que si vous avez fini tout le reste**. Ces questions sont beaucoup moins guidés, il vous sera demandé de la prise d'initiative.

Remarque: On omettra toujours les lignes

```
#include<iostream>
using namespace std;;
```

Vous pourrez toujours supposer qu'elles sont là au début de tous les programmes.

Exercice 1 (Mise en jambes)**.** Cet exercice est un échauffement pour vous aider à vous mettre dans le bain, et vous rappeler des fonctions sur les tableaux.

1. On commence par écrire une fonction permettant d'afficher un tableau
   (a) Écrivez une fonction
   ```
   void afficherTableau (int n, int tab[])
   ```
   qui prend en argument un entier $n > 0$ et un tableau de longueur n et qui affiche les éléments du tableau.
   (b) Analysez la complexité de la fonction de la question précédente.
2. On souhaite ensuite calculer la somme d'un tableau
   (a) Écrivez une fonction
   ```
   int sommeTableau (int n, int tab[])
   ```
   qui prend en argument un entier $n > 0$ et un tableau de longueur n et qui renvoie la somme des éléments du tableau.

(b) Analysez la complexité de la fonction de la question précédente.

3. Finalement, on termine par un algorithme pour échanger deux éléments dans un tableau.

   (a) Écrivez une fonction

   ```
   void echange (int i, int j, int tab[])
   ```

   qui prend en argument deux entiers $i, j > 0$ et un tableau de longueur $> \max i, j$ et qui échange les éléments du tableau en position i et j.

   (b) Analysez la complexité de la fonction de la question précédente.

Exercice 2 (Sommes partielles d'un tableau)**.** L'objectif de cet exercice est d'écrire un programme permettant de calculer et d'afficher les sommes partielles associé à un tableau. Par exemple, étant donné le tableau suivant:

1	3	4	5	7

on voudra que le programme affiche:

```
1 4 8 13 20
```

Le nombre 1 est l'élément en position 0, le nombre 4 est la somme des éléments en position 0 et 1, le nombre 8 est la somme des éléments en position 0, 1 et 2 etc.

1. On donne le programme suivant pour afficher les sommes partielles:

```
1  void sommesPartielles(int n, int tab[]){
2    // Entrees: un entier n>0 et un tableau de taille n
3    // Sorties: affiche les sommes partielles du tableau
4    for(int i = 0; i < n; i++){
5      int somme = 0;
6      for (int j = 0; j <= i; j++){
7        somme = somme + tab[j];
8      }
9      cout << somme << " ";
10   }
11 }
```

   Analysez la complexité de ce programme. Vous pouvez vous aider des numéros de ligne pour justifier votre réponse.

2. On souhaite maintenant améliorer le programme précédent qui n'est pas optimal. En effet, pour calculer la troisième somme par exemple, on ajoute les 3 premiers nombres, au lieu de réutiliser le résultat de la somme des 2 premiers nombres calculé à l'étape précédente.

(a) Donnez un programme itératif avec une complexité plus petite pour réaliser la même tache.

(b) Justifiez la complexité du programme que vous avez proposé.

3. On voudrais maintenant donner une version récursive du programme plus efficace. Il faudra pour cela reformuler le problème et définir une fonction récursive auxilliaire qui manipule plus d'arguments, et qui renvoie un résultat.

   (a) Ecrivez un programme récursif

   ```
   int sommesPartiellesRec(int n, int tab[])
   ```

   prenant en argument un entier $n > 0$ et un tableau de taille $\geq n$, et qui affiche **et renvoie** la somme des n premières entrées du tableau.

   (b) Analysez la complexité de votre algorithme récursif en fonction de n, en résolvant une suite définie par récurrence.

   (c) Ecrivez un programme

   ```
   void sommesPartielles(int n, int tab[])
   ```

   qui prend en argument un entier $n > 0$ et un tableau de taille n, et qui affiche toutes les sommes partielles, en appelant la fonction définie dans la question 3a

   (d) Donnez la complexité de cet algorithme, en vous aidant du résultat de la question 3b

Exercice 3 (Tri par sélection miroir)**.** On rappelle le principe du tri par sélection, consistant dans un premier temps à calculer la position du minimum du tableau, et à échanger le premier élément avec le minimum, de manière à positionner le minimum en première position, puis à calculer la position du minimum parmi les éléments restants et à échanger ce dernier avec le deuxième élément, de manière à placer le deuxième plus petit élément en deuxième position, et ainsi de suite. L'objectif de cet exercice est d'écrire une version miroir de ce tri, c'est à dire que l'on va chercher la position du maximum du tableau et on va l'échanger avec le dernier élément du tableau, de manière à placer correctement le maximum en dernière position, puis on va chercher la position du maximum parmi les éléments restants et l'échanger avec l'avant-dernière position du tableau de manière à correctement placer le deuxième plus grand élément, et ainsi de suite.

1. On commence par la fonction de recherche de minimum:

   (a) Écrire une fonction

   ```
   int indiceMax(int n, int tab[])
   ```

prenant en argument un entier $n > 0$ et un tableau de taille $\geq n$, et qui renvoie l'indice de l'élément maximum parmi les n premiers éléments du tableau.

(b) Analysez la complexité de la fonction `indiceMax`.

2. On procède ensuite au tri par sélection en miroir, de manière itérative

(a) Écrire une fonction

```
void triSelectionMiroir(int n, int tab[])
```

qui prend en entrée un entier $n > 0$ et un tableau de longueur n, et qui effectue le tri par sélection en miroir sur le tableau. Vous pouvez réutiliser la fonction `echange` de l'Exercice 1.

(b) Analysez la complexité de cette fonction, en justifiant bien vôtre réponse.

3. On termine en remarquant qu'il est également possible d'implémenter cet algorithme de manière récursive. Pour cela en reformule le problème en disant que l'on cherche à trier les n premiers éléments du tableau, et on remarque pour le cas récursif qu'effectuer le tri par sélection en miroir revient à d'abord placer correctement le maximum parmi les n premiers éléments en position $n-1$ dans le tableau, puis effectuer un tri par sélection miroir de manière récursive sur les $n-1$ premiers éléments du tableau.

(a) Écrivez une fonction

```
void triSelectionMiroir (int n, int tab[])
```

qui implémente la version récursive du tri par sélection. Vous pouvez réutiliser la fonction `echange` de l'Exercice 1.

(b) Analysez la complexité de cette fonction, en résolvant une suite définie par récurrence.

Exercice 4. L'objectif de cet exercice est de calculer toutes les sommes possibles (avec répétition potentielle) réalisables à partir des éléments d'un tableau. Par exemple, en partant du tableau suivant:

1	3	4	5	7

on voudra que le programme affiche (pas nécessairement dans le même ordre):

```
0 1 3 4 4 5 7 8 5 6 8 9 9 10 12 13 7 8 10 11 11 12 14
15 12 13 15 16 16 17 19 20
```

Dans cet exemple le nombre 17 est obtenu comme la somme $1 + 4 + 5 + 7$, le nombre 4 est affiché deux fois, car il est obtenu à la fois comme la somme $1 + 3$ et comme la somme 4. Le nombre 20 est la somme de tous les éléments, et le nombre 0 est la somme obtenue en ne prenant aucun élément. On va programmer cette fonction de manière.

1. Écrivez une fonction récursive

```
void sommesRec (int n, int tab[], int sommeCourante)
```

qui prend en argument un entier $n > 0$, un tableau de taille $\geq n$, et un entier `sommeCourante`, et qui affiche toutes les sommes possibles obtenues à partir de sommeCourante en ajoutant des éléments parmi les n premiers du tableau. Pour cela la stratégie consiste à remarquer que pour obtenir toutes les sommes possibles, il suffit d'afficher toutes les sommes possibles dans lesquelles `tab[n-1]` n'apparaît pas ainsi que toutes les sommes possibles dans lesquelles `tab[n-1]` apparaît.

2. Écrivez une fonction

```
void sommes (int n, int tab[])
```

qui prend en argument un entier $n > 0$, un tableau de taille $\geq n$, et qui affiche toutes les sommes possibles obtenues en ajoutant des éléments du tableau en appelant la fonction de la question précédente.

3. Analysez la complexité de cet algorithme en posant et résolvant une suite définie par récurrence.

4. **(Question bonus)** Donnez un argument d'énumération mathématique indiquant pourquoi il n'est pas possible d'écrire un programme avec une meilleure complexité, en dénombrant les sommes possibles en fonction de la longueur du tableau.

Exercice 5 (**Bonus**)**.** L'objectif de cet exercice est de calculer toutes les valeurs possibles (avec répétition potentielle) réalisables à partir des éléments d'un tableau grâce aux opérations de d'additions et de soustraction. Par exemple, en partant du tableau suivant:

1	3	4	5	7

on voudra que le programme affiche (pas nécessairement dans le même ordre):

```
0 1 -1 3 4 2 -3 -2 -4 4 5 3 7 8 6 1 2 0 -4 -3 -5 -1 0 -2 -7 -6
-8 5 6 4 8 9 7 2 3 1 9 10 8 12 13 11 6 7 5 1 2 0 4 5 3 -2 -1
-3 -5 -4 -6 -2 -1 -3 -8 -7 -9 -1 0 -2 2 3 1 -4 -3 -5 -9 -8 -10
-6 -5 -7 -12 -11 -13 7 8 6 10 11 9 4 5 3 11 12 10 14 15 13 8 9
7 3 4 2 6 7 5 0 1 -1 12 13 11 15 16 14 9 10 8 16 17 15 19 20 18
13 14 12 8 9 7 11 12 10 5 6 4 2 3 1 5 6 4 -1 0 -2 6 7 5 9 10 8
3 4 2 -2 -1 -3 1 2 0 -5 -4 -6 -7 -6 -8 -4 -3 -5 -10 -9 -11 -3 -2
-4 0 1 -1 -6 -5 -7 -11 -10 -12 -8 -7 -9 -14 -13 -15 -2 -1 -3 1 2
0 -5 -4 -6 2 3 1 5 6 4 -1 0 -2 -6 -5 -7 -3 -2 -4 -9 -8 -10 -12
-11 -13 -9 -8 -10 -15 -14 -16 -8 -7 -9 -5 -4 -6 -11 -10 -12 -16
-15 -17 -13 -12 -14 -19 -18 -20
```

Écrivez une fonction permettant de réaliser cette opérations de manière récursive, et analysez sa complexité. Pensez à introduire des fonctions auxiliaires si besoin et à spécifier les entrées et sorties de vos fonctions.