

**Consignes générales :** À faire en équipe de 1 personne minimum et 2 personnes maximum. Remettre un seul pdf et les fichiers de code python et C++. Indiquez vos noms et matricules sur tous les fichiers (pdf et code).

**General instructions:** *To be done in teams of 1 person minimum and 2 people maximum. Submit only one pdf and the python and C++ code files. Indicate your names and registration numbers on all files (pdf and code).*

Votre code doit obligatoirement passer les tests fournis avec le devoir, sinon une note de 0 sera attribué aux exercices dont les tests de base ne sont pas réussis. D’autres tests seront rajoutés lors de la correction. Seulement les librairies standards de Python et C++ sont permises, sauf si mentionné autrement. Il va de soi que le code doit être clair, bien indenté et bien commenté.

*Your code must pass the tests provided with the assignment, otherwise a grade of 0 will be given to exercises whose basic tests are not passed. Other tests will be added during the grading. Only standard Python and C++ libraries are allowed, unless otherwise stated. It goes without saying that the code must be clear, well indented and well commented.*

**La date de remise est le vendredi 7 mars à 22h30.** La structure de la remise dans Studium doit être la suivante :

**The due date is Friday, March 7th at 10:30 PM.** *The structure of the submission in Studium should be as follows:*

StudiUM  
└── devoir1\_NOM1\_NOM2.pdf  
└── Q3.py  
└── MaxToysCalculator.cpp  
└── MaxToysCalculator.h  
└── CareerCalculator.cpp  
└── CareerCalculator.h  
└── Q6.py

Il ne faut **pas** remettre de zip. Également, il ne faut pas modifier les noms de fichiers ou de fonctions (sous risque de perte de points).

*You should **not** submit a zip file. Also, you should not change the names of files or functions (at the risk of losing points).*

# 1 Notation asymptotique *Asymptotic notation* (25 points)

**Question 1:** (10 points) En utilisant les définitions de  $O$ ,  $\Omega$  et  $\Theta$  vu en classe (sans utiliser les limites), montrer ou infirmez les énoncés suivants :

(10 points) Using the definitions of  $O$ ,  $\Omega$  and  $\Theta$  seen in class (without using limits), prove or disprove the following statements:

1.  $\lfloor \frac{n}{10} \rfloor \in \mathcal{O}(\sqrt{n})$
2.  $n\sqrt{n} \log(n!) \in \mathcal{O}(n^3 \log(n))$

**Question 2:** (10 points) En utilisant la règle de la limite, déterminez l'ordre relatif ( $O$ ,  $\Omega$  ou  $\Theta$ ) des fonctions suivantes :

(10 points) Using the limit rule, determine the relative order ( $O$ ,  $\Omega$  or  $\Theta$ ) of the following functions:

1.  $f(n) = \frac{n}{\sqrt[3]{n}}$ ,  $g(n) = \ln \sqrt{n}$
2.  $f(n) = 2^{bn}$ ,  $g(n) = 3^n$  où where  $b \in \mathbb{N}^{\geq 2}$

**Question 3:** (5 points) Montrez que la fonction suivante est É.N.D. :

(5 points) Show that the following function is E.N.D.:

$$f(n) = 2n^2 - n \sin(n)$$

## 2 Récurrences *Recurrences* (25 points)

**Question 1:** (10 points) Montrez les démarches complètes qui permettent de trouver le polynôme caractéristique et les racines (avec leurs multiplicités) de la récurrence ci-dessous. Après avoir trouvé les racines, écrivez la forme générale de la récurrence avec des constantes génériques  $c_i$ .

(10 points) Show the complete steps for finding the characteristic polynomial and the roots (with their multiplicities) of the recurrence below. After finding the roots, write the general form of the recurrence with generic constants  $c_i$ .

$$t_n = 4t_{n-1} - 4t_{n-2} + 4(n+1)4^n$$

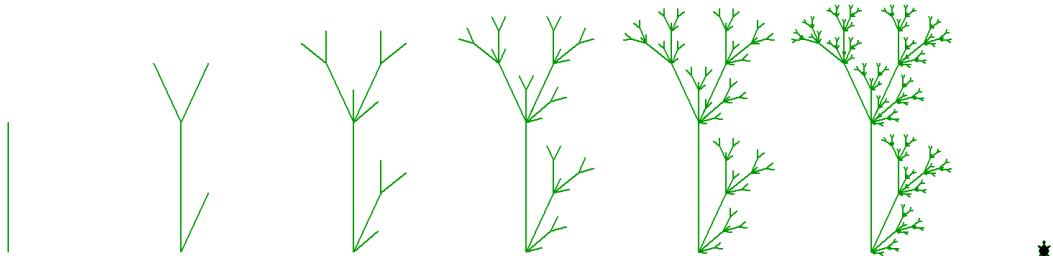
Indice : commencer par multiplier la récurrence par une constante et changer  $n$  par  $n-1$   
Hint: Start by multiplying the recurrence by a constant and changing  $n$  to  $n-1$

Attention : Vous ne devez pas répondre en utilisant simplement la formule en page 126 du livre de référence Bratley-Brassard, mais servez-vous en pour valider votre propre réponse.

Please note: You should not answer by simply using the formula on page 126 of the Bratley-Brassard reference book, but use it to validate your own answer.

**Question 2:** (10 points) Soit le code suivant qui dessine une fractale végétale avec python turtle.

(10 points) Consider the following code that draws a plant fractal with python turtle.



Quelle est la complexité de ce code? Combien de tracés "t.forward(length)" sont fait pour un certain *level*? Modéliser la récurrence et résoudre exactement. On veut avoir la forme exacte et la complexité. Notez que le code se trouve aussi en fichier avec le devoir et un compteur a été rajouté pour valider votre propre réponse à la fin.

What is the complexity of this code? How many "t.forward(length)" plots are made for a certain level? Model the recurrence and solve exactly. We want to have the exact form and complexity. Note that the code is also in a file with the assignment and a counter has been added to validate your own answer at the end.

```
import turtle
```

```

t = turtle.Turtle()
t.shape("turtle")
t.speed(0)
count = 0
alpha = 2
angle = 25

t.penup()
t.lt(90)
t.backward(200)

def tree(length, level):
    global count
    if level > 0 :
        t.pendown()
        t.forward(length)
        count += 1
        t.penup()

        t.rt(angle)
        tree(length/alpha, level - 1)
        t.lt(angle*2)
        tree(length/alpha, level - 1)
        t.rt(angle)
        t.backward(length)
        t.rt(angle)
        tree(length/alpha, level - 1)
        t.lt(angle)

tree(200,4)
print(count)
turtle.exitonclick()

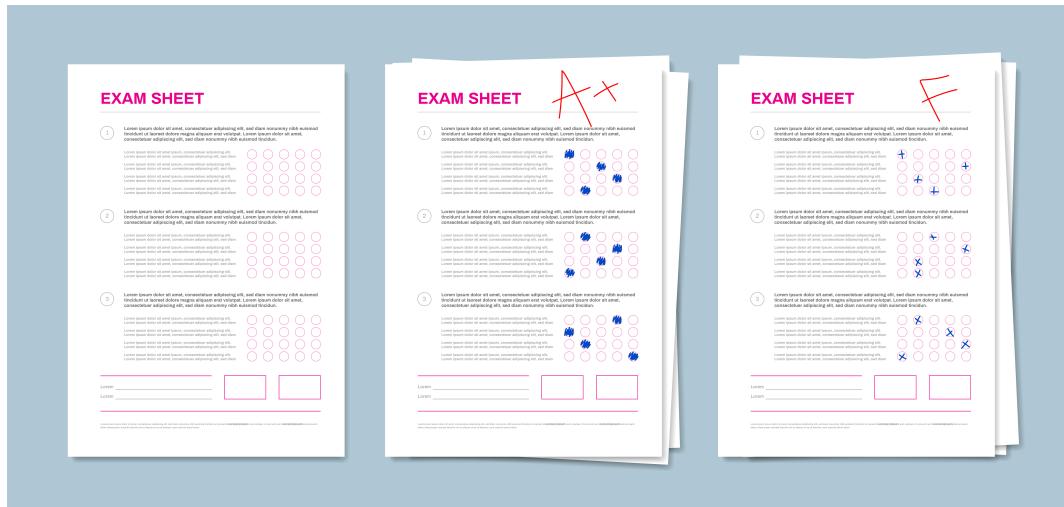
```

**Question 3:** (5 points) Utiliser le théorème maître pour trouver l'ordre exacte des récurrences suivantes :

(5 points) Use the master theorem to find the exact order of the following recurrences:

1.  $t(n) = t\left(\frac{n}{2}\right) + 4n$
2.  $t(n) = 2t\left(\frac{n}{2}\right) + 2n$
3.  $t(n) = 3t\left(\frac{n}{3}\right) + 3n^3$
4.  $t(n) = 4t\left(\frac{n}{3}\right) + 2n$
5.  $t(n) = 4t\left(\frac{n}{2}\right) + 2n^2$

### 3 Doubles pointeurs - Code Facile *Double pointers - Code Easy* (10 points)



#### Question:

Vous avez une liste triée en ordre croissant de  $n$  notes d'examen. Trouvez le nombre de paires de notes distinctes de notes qui somment à la médiane de cette liste. Par exemple, si nous avons  $\{1, 1, 1, 3, 3, 3\}$ , la médiane est 2 et la seule paire distincte de notes est  $(1, 1)$ . Résolvez le problème en Python 3.

*You have an ascending sorted list of  $n$  exam scores. Find the number of distinct pairs of scores that sum to the median of this list. For example, if we have  $\{1, 1, 1, 3, 3, 3\}$ , the median is 2 and the only distinct pair of scores is  $(1, 1)$ . Solve the problem in Python 3.*

Complexité recherchée :  $\mathcal{O}(n)$  où  $n$  est la taille d'entrée. Votre algorithme devrait être efficace pour avoir tous les points.

*Desired complexity:  $\mathcal{O}(n)$  where  $n$  is the input size. Your algorithm should be efficient to get all points.*

#### Code

Une partie du code est déjà écrite. Dans le fichier `Q3.py`, il faut compléter la fonction `solve()` (qui prend une liste d'éléments triées et retourne le nombre de paires qui satisfont le problème). Vous pouvez écrire des fonctions auxiliaires au besoins. Un fichier `Q3_test.py` et des fichiers d'entrées vous sont fournis pour vous aider à tester votre code.

*Some of the code is already written. In the `Q3.py` file, you need to complete the `solve()` function (which takes a sorted list of elements and returns the number of pairs that satisfy the problem).*

*You can write auxiliary functions if needed. A `Q3_test.py` file and input files are provided to help you test your code.*

Exemple d'appel (dans le dossier où se trouvent `Q3.py` et `input.txt`):  
*Example call (in the folder where `Q3.py` and `input.txt` are located):*

```
python3 Q3.py input.txt
```

**Remise :** Compléter le fichier `Q3.py` fournis, et ne remettre **uniquement** que ce fichier.  
Ne **pas** remettre le fichier `Q3_test.py`, ou n'importe quel autre fichier de test.

**Submission:** Complete the file `Q3.py` provided, and submit **only** this file. Do not submit the file `Q3_test.py`, or any other test file.

#### 4 Fenêtre glissante - Code Facile *Sliding Window - Code Easy* (10 points)



##### Question:

Un marchand vend des jouets pour enfants. Les  $n$  jouets sont présentés sur une longue rangée derrière une vitrine. Chaque jouet  $i$  est étiqueté d'un prix  $J[i] \in \mathbb{N}, \forall 0 \leq i < n$ . Un enfant a reçu une somme d'argent  $S$  et souhaite acheter le plus grand nombre de jouets avec cette somme. Cependant, les parents l'ont mis au défi d'acheter seulement un ensemble de jouets qui sont voisins dans la rangée. L'enfant doit donc trouver le plus grand segment de jouets qu'il peut acheter. Quelle est la taille de ce segment? Résolvez le problème en C++.

*A merchant sells children's toys. The  $n$  toys are displayed in a long row behind a glass case. Each toy  $i$  is labeled with a price  $J[i] \in \mathbb{N}, \forall 0 \leq i < n$ . A child has been given a sum of money  $S$  and wants to buy as many toys as possible with this sum. However, the parents have challenged him to buy only a set of toys that are neighbors in the row. The child must therefore find the largest segment of toys that he can buy. How big is this segment? Solve the problem in C++.*

Complexité recherchée :  $\mathcal{O}(n)$  où  $n$  est le nombre de jouets. Votre algorithme devrait être efficace pour avoir tous les points.

*Desired complexity:  $\mathcal{O}(n)$  where  $n$  is the number of toys. Your algorithm should be efficient to have all the points.*

##### Code

Exemple d'appel :

*Example call:*

Compilation :

```
g++ -o max_toys.exe max_toys.cpp MaxToysCalculator.cpp
```

Execution :

```
.\max_toys.exe
```

**Remise :** Compléter les fichiers `MaxToysCalculator.cpp` et `MaxToysCalculator.h` fournis, et ne remettre **uniquement** que ces fichiers. Ne **pas** remettre le fichier `max_toys.cpp`.

**Submission:** Complete the provided `MaxToysCalculator.cpp` and `MaxToysCalculator.h` files, and only submit **these** files. Do not submit the `max_toys.cpp` file.

## 5 Vorace - Code Intermédiaire *Greedy - Code Medium* (15 points)



### Question:

Dans un monde figuré et simplifié, une personne est sur un grand ruban d'une carrière planifiée constitué de  $n$  cases. Chaque case du ruban  $R$  représente des étapes clés de la carrière et permet à la personne de se transporter jusqu'à  $R[i]$  cases vers l'avant. Par exemple, arriver à la fin de l'école primaire permet de se rendre à la fin de l'école secondaire, qui permet à son tour des nouveaux avancements. La personne commence sur la première case du ruban et souhaite se rendre à la dernière case du ruban qui représente l'objectif ultime de son plan de carrière. Veuillez déterminer si cet objectif est réalisable selon les étapes planifiées de la carrière si la personne prends les meilleures choix possible. Résolvez le problème en C++.

*In a simplified and figurative world, a person is on a large ribbon of a planned career consisting of  $n$  squares. Each square of the ribbon  $R$  represents key stages of the career and allows the person to move up to  $R[i]$  squares forward. For example, reaching the end of primary school allows one to go to the end of secondary school, which in turn allows further advancements. The person starts on the first square of the ribbon and wants to go to the last square of the ribbon which represents the ultimate goal of his/her career plan. Please determine if this goal is achievable according to the planned stages of the career if the person makes the best possible choices. Solve the problem in C++.*

Complexité recherchée :  $\mathcal{O}(n)$  où  $n$  est le nombre de cases. Votre algorithme devrait être efficace pour avoir tous les points.

*Desired complexity:  $\mathcal{O}(n)$  where  $n$  is the number of boxes. Your algorithm should be efficient to have all the points.*

### Code

Exemple d'appel :

*Example call:*

Compilation :

```
g++ -o career.exe career.cpp CareerCalculator.cpp
```

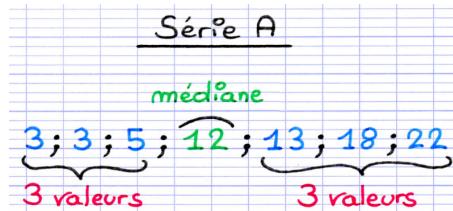
Execution :

```
.\\career.exe
```

**Remise :** Compléter les fichiers `CareerCalculator.cpp` et `CareerCalculator.h` fournis, et ne remettre **uniquement** que ces fichiers. Ne **pas** remettre le fichier `career.cpp`.

**Submission:** Complete the provided `CareerCalculator.cpp` and `CareerCalculator.h` files, and submit **only** these files. Do not submit the `career.cpp` file.

## 6 Diviser-pour-régner - Code Difficile *Divide and Conquer* - *Code Hard* (15 points)



### Question:

Vous obtenez 2 listes de nombres déjà triées. Vous devez trouver la médiane de tous ces nombres, et ce, le plus rapidement possible. Résolvez le problème en Python 3.

*You get 2 lists of numbers already sorted. You have to find the median of all these numbers, and do it as quickly as possible. Solve the problem in Python 3.*

Complexité recherchée :  $\mathcal{O}(\log(n+m))$  où  $n$  et  $m$  sont respectivement les longueurs des 2 listes. Votre algorithme devrait être efficace pour avoir tous les points. Commencez par une solution en  $\mathcal{O}((n+m)\log(n+m))$  où l'on tri simplement le tout, suivi d'une solution en  $\mathcal{O}(n+m)$  qui tri en utilisant le fait que les 2 listes sont déjà triées, pour tester vos solutions suivantes. Des points partiels pourront être donnés pour ces solutions si aucune solution n'est remise avec la complexité recherchée.

*Desired complexity:  $\mathcal{O}(\log(n+m))$  where  $n$  and  $m$  are the lengths of the 2 lists respectively. Your algorithm should be efficient to get all the points. Start with a solution in  $\mathcal{O}((n+m)\log(n+m))$  where we simply sort everything, followed by a solution in  $\mathcal{O}(n+m)$  which sorts using the fact that the 2 lists are already sorted, to test your next solutions. Partial points may be given for these solutions if no solution is returned with the desired complexity.*

### Code

Une partie du code est déjà écrite. Dans le fichier Q6.py, il faut compléter la fonction `solve()` (qui prend deux listes d'éléments triées et retourne la médiane des deux listes). Vous pouvez écrire des fonctions auxiliaires au besoins.

*Some of the code is already written. In the Q6.py file, you need to complete the function `solve()` (which takes two sorted lists of elements and returns the median of the two lists). You can write auxiliary functions if needed.*

Exemple d'appel (dans le dossier où se trouvent Q6.py et `input.txt`):

*Example call (in the folder where Q6.py and input.txt are located):*

```
python Q6.py input.txt
```

Un fichier `Q6_test.py` et des fichiers d'entrées vous sont fournis pour vous aider à tester

votre code.

*A Q6\_test.py file and input files are provided to help you test your code.*

**Remise :** Compléter le fichier Q6.py fournis, et ne remettre **uniquement** que ce fichier.

Ne **pas** remettre le fichier Q6\_test.py, ou n'importe quel autre fichier de test.

**Submission:** Complete the file Q6.py provided, and only submit **this** file. Do not submit the file Q6\_test.py, or any other test file.

#### **Astuces Hints**

- Commencer par un nombre impair d'éléments au total. *Start with an odd number of items in total.*
- Faire une recherche dichotomique qui utilise le fait que les listes sont triées. *Perform a dichotomous search that uses the fact that the lists are sorted.*
- Une vraie médiane va séparer les éléments de la liste en combien d'éléments à gauche et à droite? *A true median will separate the elements of the list into how many elements to the left and right?*
- Si on prend un candidat pour la médiane dans une première liste, où devrait-il se situer dans la seconde liste? *If we take a candidate for the median in a first list, where should he be located in the second list?*