Ana Carolina da Silva Carvalho de Sousa
Carlos Adir Ely Murussi Leite

1 Structures de données

Dans le fichier types.h.

1.1 Structure Matiere

Pour cela, un enregistrement a été réalisé avec deux caractéristiques définies: nom et poids, Le première du type chaîne de caractère, c'est-à-dire des string, et le deuxième du type réel, c'est-à-dire float.Il est important de mentionner qu'il s'agit de la première structure créée car elle sera ensuite utilisée dans la structure "Objet".

```
Enregistrement matiere
    chaine de caractère nom;
    réel poids;
Fin Enregistrement
```

1.2 Vector ensMatieres

Ensuite, une structure de données "ensMatieres" a été définie au format vectoriel. Ses éléments sont du type "matiere" défini précédemment et leur taille est inconnue. Ainsi, la structure "vector" du langage C ++ a été utilisée.

1.3 Structure Objet

Il a été défini par un enregistrement comportant six caractéristiques: nom, nombre, nombre_matières, matières, date et durée. Les types sont respectivement chaîne de caractère, entier, entier, ensMatieres (défini ci-dessus), entier et entier.

```
Enregistrement Objet
    chaîne de caractère nom;
    entier nombre;
    entier nombre_matieres;
    ensMatieres matieres;
    entier date;
    entier duree;
Fin Enregistrement
```

1.4 Vector ensObjets

Ensuite, une structure de données *ensObjets* a été définie au format vectoriel. Ses éléments sont du type *objet* défini précédemment et leur taille est inconnue. Ainsi, la structure *vector* du langage C++ a été utilisée.

1.5 Structure livraison

Ensuite, une structure de "livraison" a été définie afin que les commandes puissent être stockées. Cette structure a quatre caractéristiques: id, nom, poids et date. Ses types sont respectivement entier, chaîne de caractère, entier et entier.

```
Enregistrement livraison
    entier id;
    chaîne de caractère nom;
    entier poids;
    entier date;
Fin Enregistrement
```

1.6 Vector ensLivraison

Ensuite, une structure de données *ensLivraison* a été définie au format vectoriel. Ses éléments sont du type "livraison" défini précédemment et leur taille est inconnue. Ainsi, la structure *vector* du langage C++ a été utilisée.

2 Lecture et écriture

Dans les fichiers "utils.h" et "utils.cpp". Fonctions de tri contenues dans le fichier "tris.cpp" et "tris.h".

2.1 Fonction lireObjets_1

```
Description:
Cette fonction lit le texte "objetsInput_100.txt" dans le but de modifier un tableau ensObjets
donné en entrée. Pour cela, chaque ligne est lue et stockée dans un type Objet et cet objet est
inclus dans ce vecteur ensObjets. L'entrée est donnée sous forme de pointeur afin que la
modification puisse être faite dans la mémoire de l'ordinateur.

Spécification:
Fonction : 0 <- lireObjets\_1 (eo)
Parametres: lien ensObjets eo;
Resultats : 0;
```

Algorithm 1 lireObjets_1

Entrée: lien ensObjet *eo*
Sortie: ∅

```
1:  Variables locales:
2:  objet o
3:  matiere m
4:  caractere c {Pour resoudre le probleme d'espace au finale de la liste.}
5:  entier i
6:
7:  Algorithme debut LIREOBJETS_1( )
8:      fichier.ouvrir(""objetsInput_100.txt"")
9:      si errorOpeningFile alors
10:         Ecrire("Ne peut pas lire lefichier")
11:     sinon
12:         tant que fileNeEstPasAuFin faire
13:             fichier.lire(o.nom)
14:             fichier.lire(o.nombre)
15:             fichier.lire(o.nombre_matieres)
16:             pour i ← 0 a o.nombre_matieres faire
17:                 fichier.lire(m.nom)
18:                 fichier.lire(m.poids)
19:                 o.matieres.ajouter(m)
20:             fin pour
21:             fichier.lire(o.date)
22:             fichier.lire(o.duree)
23:             fichier.lire(c)
24:             eo.addAuFin(o)
25:         fin tant que
26:         fichier.fermer()
27:     fin si
28: fin Algorithme debut
```

2.2 Fonction ecrireObjets_1

```
Description:
Cette fonction est destinée à recevoir un vecteur "ensObjets" et à écrire dans un fichier texte
"objetsOutput_100_1.txt" tous les objets contenus dans ce vecteur. Cependant, comme les objets
devaient être écrits de manière ordonnée, cette première fonction Écrire a été lancée avec une
fonction de tri vectoriel. Cette commande a été effectuée via une fonction "tri_vecteur", contenue
dans le fichier "tri.cpp" et "tri.h".

Spécification:
Fonction : 0 <- ecrireObjets_1 (eo)
```

```
Parametres: lien ensObjets eo;
Resultats : 0;
```

Algorithm 2 ecrireObjets_1

Entrée: ensObjet *eo*
Sortie: ∅

```
1: Algorithme debut ECRIREOBJETS_1( )
2:     tri_vecteur(eo)
3:     fichier.ouvrir(objetsOutput_100.txt)
4:     si fichier.error() alors
5:         Ecrire("Ne peut pas ouvrir lefichier")
6:     sinon
7:         pour i ← 0 a eo.size() faire
8:             fichier.ecrire(eo[i].nom)
9:             fichier.ecrire(eo[i].nombre)
10:            fichier.ecrire(eo[i].nombre_matieres)
11:            pour j ← 0 a eo[i].nombre_matieres faire
12:                fichier.ecrire(eo[i].matieres[j].nom)
13:                fichier.ecrire(eo[i].matieres[j].poids)
14:            fin pour
15:            fichier.ecrire(eo[i].date)
16:            fichier.ecrire(eo[i].duree)
17:        fin pour
18:        fichier.fermer()
19:    fin si
20: fin Algorithme debut
```

2.3 Fonction tri_vecteur

```
Description:
Cette fonction utilise l'algorithme Bubblesort pour trier le vecteur d'objet passé
en entrée de la fonction.

Spécification:
Fonction : 0 <- tri\_vecteur (eo)
Parametres: lien ensObjets eo;
Resultats : 0;
```

Algorithm 3 tri_vecteur

Entrée: lien ensObjet *eo*
Sortie: ∅

```
1: Vaiables locales:
2: entier i
3: boolean aucunEchange
4:
5: Algorithme debut TRI_VECTEUR( )
6:     tant que auncunEchange = faux faire
7:         aucunEchange ← vrai
8:         pour j ← 0 a n − 2 faire
9:             si eo[i].date > eo[i+1].date alors
10:                trocar_objet (eo(i), eo(i+1))
11:                auncunEchange ← faux
12:            fin si
13:        fin pour
14:    fin tant que
15: fin Algorithme debut
```

3 Tris selon la date lors de la lecture

Dans les fichiers "utils.h" et "utils.cpp". Fonctions de tri contenues dans le fichier "tris.cpp" et "tris.h".

3.1 Fonction lireObjets_2

```
Description:
Cette fonction lit le texte "objetsInput_100.txt" dans le but de modifier un tableau ensObjets
donné en entrée. Pour cela, chaque ligne est lue et stockée dans un type Objet et cet objet est
inclus dans ce vecteur ensObjets. La principale différence avec cette fonction est qu'elle
utilise la fonction "ajouter_objet" lors de l'ajout de l'objet lu au tableau d'objets. Cela
entraîne l'inclusion d'objets dans le fichier de manière ordonnée. Cette commande a été faite de
manière linéaire.

Spécification:
Fonction : O <- lireObjets\_2 (eo)
Parametres: lien ensObjets eo;
Resultats : O;
```

Algorithm 4 lireObjets_2

Entrée: lien ensObjet *eo*
Sortie: ∅

```
1:  Variables locales:
2:  objet o
3:  matiere m
4:  caractere c {Pour resoudre le probleme d'espace au finale de la liste.}
5:  entier i
6:
7:  Algorithme debut LIREOBJETS_2( )
8:      fichier.ouvrir(""objetsInput_100.txt"")
9:      si errorOpeningFile alors
10:         Ecrire("Ne peut pas lire lefichier")
11:     sinon
12:         tant que fileNeEstPasAuFin faire
13:             fichier.lire(o.nom)
14:             fichier.lire(o.nombre)
15:             fichier.lire(o.nombre_matieres)
16:             pour i ← 0 a o.nombre_matieres faire
17:                 fichier.lire(m.nom)
18:                 fichier.lire(m.poids)
19:                 o.matieres.ajouter(m)
20:             fin pour
21:             fichier.lire(o.date)
22:             fichier.lire(o.duree)
23:             fichier.lire(c)
24:             eo.ajouter_objet(o)
25:         fin tant que
26:         fichier.fermer()
27:     fin si
28: fin Algorithme debut
```

3.2 Fonction ajouter_objet

```
Description:
Cette fonction prend un type Objet et ajoute au tableau d'objets,également donné en entrée, de
manière ordonnée. Cette commande est donnée par le critère de la date de livraison, c'est-à-dire
que les objets doivent être inclus pour que la date de livraison augmente. Cette recherche a été
effectuée de manière linéaire, c'est-à-dire que tous les éléments de la liste sont traversés pour
insérer l'élément.
```

```
Spécification:
Fonction : 0 <- ajouter\_objet (eo, o)
Parametres: lien ensObjets eo;
            Objet o;
Resultats : 0;
```

Algorithm 5 ajouterObjet

Entrée:
1: ensObjet *eo*
2: objet *o*
Sortie:
3: ensObjet *eo*
4: **Algorithme debut** AJOUTER_OBJET( )
5: **si** *eo.size*() = 0 **alors**
6: eo.addAuFin(o)
7: **sinon**
8: **pour** *i* ← 0 **a** *eo.size*() **faire**
9: **si** eo[i].date ¿ o.date **alors**
10: eo.insert(i, o)
11: **briser** {Pour arreter le *pour*}
12: **fin si**
13: **fin pour**
14: **si** *i* = *eo.size*() **alors**
15: eo.addAuFin(o)
16: **fin si**
17: **fin si**
18: fichier.ouvrir(objetsOutput_100.txt)
19: **si** fichier.error() **alors**
20: *Ecrire*("*Ne peut pas ouvrir le fichier*")
21: **sinon**
22: **pour** *i* ← 0 **a** *eo.size*() **faire**
23: fichier.ecrire(eo[i].nom)
24: fichier.ecrire(eo[i].nombre)
25: fichier.ecrire(eo[i].nombre_matieres)
26: **pour** *j* ← 0 **a** *eo*[*i*].*nombre_matieres* **faire**
27: fichier.ecrire(eo[i].matieres[j].nom)
28: fichier.ecrire(eo[i].matieres[j].poids)
29: **fin pour**
30: fichier.ecrire(eo[i].date)
31: fichier.ecrire(eo[i].duree)
32: **fin pour**
33: fichier.fermer()
34: **fin si**
35: **fin Algorithme debut**

3.3 Fonction lireObjets_3

```
Description:
Cette fonction lit le texte "objetsInput_100.txt" dans le but de modifier un tableau ensObjets
donné en entrée. Pour cela, chaque ligne est lue et stockée dans un type Objet et cet objet est
inclus dans ce vecteur ensObjets. La principale différence avec cette fonction est qu'elle
utilise la fonction "ajouter_objet2" lors de l'ajout de l'objet lu au tableau d'objets. Cela
entraîne l'inclusion d'objets dans le fichier de manière ordonnée. Cette commande a été
faite avec recherche dichotomique.
```

```
Spécification:
Fonction : 0 <- lireObjets\_3 (eo)
Parametres: lien ensObjets eo;
Resultats : 0;
```

Algorithm 6 lireObjets_3

Entrée: lien ensObjet *eo*
Sortie: $\emptyset$

```
1: Variables locales:
2: objet o
3: matiere m
4: caractere c {Pour resoudre le probleme d'espace au finale de la liste.}
5: entier i
6:
7: Algorithme debut LIREOBJETS_3( )
8:     fichier.ouvrir(""objetsInput_100.txt"")
9:     si errorOpeningFile alors
10:        Ecrire("Ne peut pas lire le fichier")
11:    sinon
12:        tant que fileNeEstPasAuFin faire
13:            fichier.lire(o.nom)
14:            fichier.lire(o.nombre)
15:            fichier.lire(o.nombre_matieres)
16:            pour i ← 0 a o.nombre_matieres faire
17:                fichier.lire(m.nom)
18:                fichier.lire(m.poids)
19:                o.matieres.ajouter(m)
20:            fin pour
21:            fichier.lire(o.date)
22:            fichier.lire(o.duree)
23:            fichier.lire(c)
24:            eo.ajouter_objet2(o)
25:        fin tant que
26:        fichier.fermer()
27:    fin si
28: fin Algorithme debut
```

Fonction ajouter_objet2

```
Description:
Cette fonction prend un type Objet et ajoute au tableau d'objets,également donné en entrée, de
manière ordonnée. Cette commande est donnée par le critère de la date de livraison, c'est-à-dire
que les objets doivent être inclus pour que la date de livraison augmente.

Spécification:
Fonction : 0 <- ajouter\_objet (eo, o)
Parametres: lien ensObjets eo;
            Objet o;
Resultats : 0;
```

Fonction ecrireObjets_2

```
Description:
La différence entre cette seconde fonction Écrire et la précédente réside uniquement dans le fait
qu'il n'ya pas besoin de classer les vecteurs, puisqu'elle a déjà été lue de manière ordonnée.
Cette recherche a été effectuée de manière dichotomique, c'est-à-dire qu'à chaque itération,
la moitié des éléments ont été ignorés.

Spécification:
Fonction : 0 <- ecrireObjets_2 (eo)
Parametres: lien ensObjets eo;
Resultats : 0;
```

Algorithm 7 ecrireObjets_1

Entrée: ensObjet *eo*
Sortie: ∅

```
1:  Algorithme debut ECRIREOBJETS_2( )
2:      fichier.ouvrir(objetsOutput_100_2.txt)
3:      si fichier.error() alors
4:          Ecrire("Ne peut pas ouvrir lefichier")
5:      sinon
6:          pour i ← 0 a eo.size() faire
7:              fichier.ecrire(eo[i].nom)
8:              fichier.ecrire(eo[i].nombre)
9:              fichier.ecrire(eo[i].nombre_matieres)
10:             pour j ← 0 a eo[i].nombre_matieres faire
11:                 fichier.ecrire(eo[i].matieres[j].nom)
12:                 fichier.ecrire(eo[i].matieres[j].poids)
13:             fin pour
14:             fichier.ecrire(eo[i].date)
15:             fichier.ecrire(eo[i].duree)
16:         fin pour
17:         fichier.fermer()
18:     fin si
19: fin Algorithme debut
```

3.6- Fonction auxiliaire trocar_objet

```
Description:
Cette fonction d'assistance a été définie uniquement pour modifier la position des objets dans
le tableau d'objets du type ensObjets. Il était requis dans les fonctions "ajouter_objet" et
"ajouter_objet2".

Spécification:
Fonction : 0 <- trocar_objet (obj1, obj2)
```

```
Parametres: objet obj1;
            objet obj2;
Resultats : 0;
```

4 Calcul du retard

4.1 calculRetard

Description

Pour faire le calcul du retard, nous avons adopter la stratégie de faire un loop, et calculer la pénalité pour chaque objet vers une fonction pour chaque objet

Pseudocode

Algorithm 8 calculRetard

Entrée:

1: ensMatieres stock ensObjets vectObj {le vecteur avec tout les objets obtenus}

Sortie:

2: entier *penalites*
3: **Algorithme debut** CALCULRETARD( )
4: $penalites \leftarrow 0$
5: **pour** $i \leftarrow 0$ **a** $vectObj.size()$ **faire**
6: $penalites \leftarrow penalites + calculRetardUnObjet(em, vectObj, vectObj[i])$
7: **fin pour**
8: **renvoyer** penalites
9: **fin Algorithme debut**

4.2 calculRetardUnObjet

Description

Pour faire le calcul du retard de un objet, la idée est que nous irons tester si il y a la quantité de matière nécessaire pour le objet, dans le stock. Si il n'y a pas, alors nous irons mettre ce objet pour le jour suivant, et incrémenter la pénalité du objet.

Pseudocode

Algorithm 9 calculRetardUnObjet

Entrée:

ensObjets vectObj {le vecteur avec tout les objets obtenus}
objet *ob*

Sortie:

entier *penalite*
Variables Locales
ensMatieres stock
entier date
Algorithme debut CALCULRETARD( )
    penalites ← 0
    Initialisation_matieres(stock) {Mettre 2000 dans chaque unite dans stock}
    miseajour_stock(stock, em, vectObj, ob, date)
    pour *i* ← 0 **a** *ob.nombre_matieres* **faire**
        pour *j* ← 0 **a** *stock.size*() **faire**
            si ob.matieres[i].nom == stock[j].nom **alors**
                tant que *stock[j].poids* < *ob.matieres[i].poids* **faire**
                    {La quantite dans le stock est plus petit}
                    date ← *date* + 1
                    penalite ← *penalite* + 1000 ∗ *ob.nombre*
                    Initialisation_matieres(stock)
                    miseajour_stock(stock, em, vectObj, ob, date)
                fin tant que
            fin si
        fin pour
    fin pour
    renvoyer penalite
fin Algorithme debut

5 Matieres premieres

Il est intéressant d'expliquer comment le problème de la variation des stocks au fil du temps sera traité pour contrôler les pénalités de retard. Afin de pouvoir calculer s'il y a du matériel nécessaire pour la production d'un certain lot, il est nécessaire de vérifier si du matériel est disponible le jour où il commencera la production.

Pour cela, un vecteur *ensMatieres* sera initialisé et représentera la quantité de stock à un moment donné. Ensuite, une fonction est créée pour mettre à jour les quantités en stock disponibles pour la production d'un certain objet. C'est-à-dire que la quantité d'articles ne sera mise à jour qu'en fonction des besoins de cet objet.

Afin de mettre à jour ce stock pour le moment où l'objet doit être produit, il est nécessaire de retirer de l'inventaire toute la quantité de matériau à utiliser pour la production des objets prioritaires (avec une date de livraison inférieure à objet) et ajoutez les livraisons effectuées jusqu'à présent. Pour cela, deux fonctions ont été faites.

Fonctions écrites dans les fichiers *utils.cpp* et *utils.h*.

5.1 Fonction lireMatiere

Description

Cette fonction est destinée à la lecture des commandes d'articles stockées dans un fichier texte "matieres_100.txt". Chaque ligne est considérée comme une demande et elle est stockée dans un type "livraison".

Pseudo-code

Algorithm 10 lireMatiere

Entrée: ∅
Sortie: ensLivraison *em*

```
1: Variable Locales:
2: livraison m
3: Algorithme debut LIREMATIERE( )
4:     ouvrirFichier("matieres_100.txt")
5:     si errorOpeningFile alors
6:         Ecrire("Ne peut pas lire lefichier")
7:     sinon
8:         tant que fileNeEstPasAuFin faire
9:             lire(m.id)
10:            lire(m.nom)
11:            lire(m.poids)
12:            lire(m.date)
13:            em.ajouter(m)
14:        fin tant que
15:        fermerFichier()
16:    fin si
17: fin Algorithme debut
```

5.2 Initialisation_matieres

Description

Cette fonction reçoit l'adresse du vecteur ensMatieres qui sera le vecteur de disponibilité du stock à un moment donné. Ce vecteur sera créé en fonction du vecteur généré par la fonction lireMatierere qui contient toutes les commandes article. En d'autres termes, un vecteur sera modifié en fonction des matériaux existants et la quantité de chaque matériau sera initialisée avec 2 tonnes (stocké dans une constante appelée QUANT_INITIAL).

Pseudocode

Algorithm 11 initialisation_matieres

Entrée: ∅
Sortie: ensMatieres *stock*

```
1: Variables locales
2: ensLivraison em
3: matiere newMat
4: Algorithme debut INITIALISATION_MATIERES( )
5:     em ← lireMatiere()
6:     pour i ← 0 a em.size() faire
7:         pour j ← 0 a stock.size() faire
8:             si em[i].nom = stock[j].nom alors {Ici nous avons trouve la matiere dans le stock}
9:                 briser {Nous pouvons arreter la recherche dans le stock}
10:            fin si
11:        fin pour
12:        si j = stock.size() alors {Si le j est au fin, alors il n'a pas trouve}
13:            newMat.nom ← em[i].nom
14:            newMat.poids ← QUANT_INITIAL
15:            stock.ajouter(newMat) {Et nous irons ajouter}
16:        fin si
17:    fin pour
18:    renvoyer stock
19: fin Algorithme debut
```

5.3 Fonction Add livraisons to stock

Description

Comme indiqué dans l'introduction de cette section, pour que l'inventaire soit mis à jour au moment où l'objet doit commencer la production, toutes les livraisons déjà effectuées doivent être ajoutées au stock.

Pseudocode

Algorithm 12 add_livraisons_to_stock

Entrée: ensMatieres *stock* ensLivraison *em* entier *date*
Sortie: ensMatieres *stock*

```
1: Algorithme debut ADD_LIVRAISONS_TO_STOCK( )
2:     pour i ← 0 a em.size() faire
3:         si em[i].date ≤ date alors {Nous prenons la date de la livraison, et si ça est avant le date}
4:             stock ← put_in_stock(stock, em[i]) {Nous pouvons mettre}
5:         fin si
6:     fin pour
7:     renvoyer stock
8: fin Algorithme debut
```

5.4 Fonction put in stock

Description

La fonction mettre seulement une livraison dans le stock. Cette fonction d'assistance a été définie pour faciliter la visualisation de la fonction ci-dessus uniquement.

Pseudocode

Algorithm 13 put_in_stock

Entrée: ensMatieres *stock* livraison *liv*
Sortie: ensMatieres *stock*

```
1: Algorithme debut PUT_IN_STOCK( )
2:     pour i ← 0 a stock.size() faire
3:         si stock[i].nom = liv.nom alors {Si la matiere est la meme}
4:             stock[i].poids ← stock[i].poids + liv.poids
5:         fin si
6:     fin pour
7:     renvoyer stock
8: fin Algorithme debut
```

5.5 RetirerObjetDuStock

Description

Comme indiqué dans l'introduction à la session, pour que le stock disponible pour la production d'un lot soit mis à jour, il est nécessaire, en plus d'ajouter le stock reçu selon les livraisons, de supprimer le stock qui sera utilisé dans la production d'objets prioritaires, c'est-à-dire les objets dont la date de livraison est inférieure à celle de l'objet traité.

De cette manière, la fonction reçoit le vecteur objet, un objet et un vecteur de stock de matériau. Pour tous les objets antérieurs à l'objet entrant, les quantités nécessaires à la production de ces autres articles sont sorties du stock.

Pseudocodigo

Algorithm 14 RetirerObjetDuStock

Entrée:

ensMatieres *stock*
ensObjets *vectObj*
objet o

Sortie:

ensMatieres *stock*
Algorithme debut RetirerObjetDuStock( )
    pour $i \leftarrow 0$ **a** *vectObj.size*() **faire**
        si $vectObj[i].date \leq o.date$ **alors**
            pour $j \leftarrow 0$ **a** $vectObj[i].nombre\_matieres$ **faire**
                pour $k \leftarrow 0$ **a** *stock.size*() **faire**
                    si $vectObj[i].matieres[j].nom = stock[k].nom$ **alors**
                        $stock[k].poids \leftarrow stock[k].poids - vectObj[i].matieres[j].poids$
                    fin si
                fin pour
            fin pour
        fin si
    fin pour
    renvoyer *stock*
fin Algorithme debut

5.6 Miseajour

Description

Cette fonction regroupe les deux pièces nécessaires à la mise à jour du stock jusqu'au jour où un certain objet (fourni en entrée) doit être produit. De cette façon, il utilise les deux fonctions précédentes "add_livraisons_to_stock" et "retirer_objets_from_stock".

Pseudocode

Algorithm 15 miseajour_stock

Entrée:

ensMatieres *stock*
ensLivraison *em*
ensObjets *vectObj*
objet *o*
entier *date*

Sortie:

ensMatieres *new_stock*
Algorithme debut MISEAJOUR_STOCK( )
    $new\_stock \leftarrow add\_livraisons\_to\_stock(stock, em, date)$
    $new\_stock \leftarrow retirer\_objets\_from\_stock(new\_stock, vectObj, o)$
    renvoyer *new_stock*
fin Algorithme debut

6 Matieres premieres

Tous les résultats de tests et conclusions d'algorithmes ont été rassemblés dans cette section pour faciliter la visualisation.

6.1 Tests avec Lecture et écriture

Nous avons constaté une augmentation significative du temps d'exécution des algorithmes devant traverser des structures de données pour trier par date de livraison. Toutefois, pour la question 2, qui propose le tri dans la partie écriture, le temps total de lecture et d'écriture de 1,495 seconde. L'algorithme Bubble a été utilisé pour ce tri, qui n'est pas le plus optimisé. Pour la question 3, le temps d'exécution total était 1,026 et 1,039 pour la recherche linéaire et dichotomique, respectivement. De cette façon, nous pouvons constater que les coûts d'exécution sont moins élevés pour les algorithmes de tri avant l'ajout au vecteur.

Table 1: Tableau qui montre le temps en miliseconds

Functions	Description	Temps (ms)
lireObjets_1	pas de tri	0,197
lireObjets_2	recherche lineaire	0,631
lireObjets_3	recherche dichotomique)	0,646
ecrireObjets_1	Bubble sorte	1,298
ecrireObjets_2	pas de tri	0,393

6.2 Tests avec Lecture et écriture

Pour représenter la variation des pénalités, les valeurs totales des pénalités ont été calculées pour différentes valeurs de stock initiales. Ceux-ci sont représentés dans le tableau ci-dessous: