Localisation de Point

Samy Jaziri

Sujet de khôlle inspiré du papier de recherche "Planar Point Location Using Persistent Search Trees" de SARNAK et TARJAN (1986).

Préambule

*Tous les algorithmes de ce sujet devront être implémentés en **OCaml**.*

Pour vous mettre dans les conditions du concours, vous n'avez pas le droit d'accéder aux ressources en ligne. Une documentation hors-ligne peut être téléchargée à l'adresse suivante en début d'épreuve : `https://kholles.jaziri.eu/doc-ocaml.zip`

Vous indiquerez vos réponses sur la fiche réponse qui vous a été fournie en utilisant, en entrée de vos programmes, le numéro u_0 inscrit sur cette fiche. Vous remettrez cette fiche à l'examinateur en fin de séance. Une fiche réponse est mise à disposition à la fin du sujet en tant qu'exemple des réponses attendues pour un $\widetilde{u_0}$ particulier.
En ce qui concerne les questions orales de la khôlle, lorsque la description d'un algorithme est demandée, vous devez présenter son fonctionnement de façon schématique, courte et précise. Vous ne devez pas expliquer votre code ligne par ligne! Quand la complexité d'un algorithme est demandée en temps ou en mémoire en fonction d'un paramètre n, on demande l'ordre de grandeur en fonction du paramètre, donné en notation de Landau ($\mathcal{O}(n), \mathcal{O}(\log(n)), \dots$). Prenez des notes lorsque vous préparez une question orale pour retrouver plus rapidement les grandes lignes de votre explication lorsque l'examinateur passe vous voir.
Il est recommandé de **tester vos programmes sur des petits exemples** que vous aurez résolus préalablement à la main ou bien à l'aide de la fiche réponse type fournie en annexe.
Il vous est demandé d'aborder les questions dans l'ordre et de noter vos difficultés à répondre à une question avant de passer à la suivante. Vous pourrez alors les aborder avec l'examinateur.

1 Introduction

Le problème considéré dans ce sujet est celui de la localisation d'un point dans un plan partitionné en plusieurs zones. On cherche en particulier à créer une structure de donnée, nommée *partition* permettant de répondre à ce problème en temps logarithmique. Les trois dernières parties s'intéresseront à la création de la structure de donnée à partir d'un ensemble de sommets et d'arêtes partitionnant le plan. Chaque partie propose un algorithme différent, mais ayant le même principe de base. Ce principe est décrit dans la section *Localisation de point*.

Librairie de travail

Téléchargez l'archive `https://kholles.jaziri.eu/kholle_localisation_de_point.zip` et extrayez les fichiers dans votre répertoire de travail. Il s'agit d'une librairie de travail pour ce sujet. Pour l'utiliser :

- Si vous utiliser l'interpréteur OCaml, en vous assurant qu'il est bien lancé dans le répertoire de travail, lancez la commande `#load "kholle_localisation_de_point.cmo" ;;` puis commencez votre code par `open Kholle_localisation_de_point ;;` et interprétez cette ligne.
- Si vous compilez votre code OCaml, commencez simplement votre code par

  `open Kholle_localisation_de_point ;;`.

Si vous rencontrez un problème quelconque, appelez l'examinateur. Le code source est fourni si, au pire des cas, vous devez interpréter directement, ou recompiler le code.

Génération aléatoire d'ensembles

Pour générer aléatoirement un partitionnement du plan, nous utiliserons la triangulation de Delaunay d'un ensemble de points générés aléatoirement. Dans cette partie nous donnons une méthode pour construire ces ensembles aléatoires.
La librairie `kholle_localisation_de_point` fourni les types suivants :

```
type point = int * int (* abscisse et ordonnée *)
type 'a ensemble = 'a array
```

Considérons $(u_k)_{k\geq 0}$ la suite d'entiers définie par :

$$u_k = \begin{cases} u_0 \text{ (sur votre fiche réponse)} & \text{si } k = 0 \\ 37698 \times u_{k-1} \mod 524287 & \text{si } k > 0 \end{cases}$$

Question 1

Que valent :
a) u_{100} **b)** u_{200} **c)** u_{300}

Pour tout $n \in \mathbb{N}$ on définit l'ensemble

$$E_n = \{(2\,u_{2\,i}, 2\,u_{2\,i+1}), i \in [\![0, n-1]\!]\}$$

Question 2

Implémentez une fonction `e` prenant en argument un entier `n` et renvoyant un élément de type `point ensemble` représentant l'ensemble E_n défini ci-dessous.

2 Recherche logarithmique

On aura besoin dans ce sujet, étant donné un élément e et un ensemble ordonné E, de rechercher le plus grand élément de E qui est plus petit ou égal à e. Toutefois, si e est comparable à tous les éléments de E, ils ne sont pas forcément de même type. Par exemple, un point p et un segment s seront comparable : p est au-dessus, sur ou en-dessous de s.
Dans ce sujet, l'ensemble en question peut-être représenté par un tableau, auquel cas on utilisera une recherche dichotomique, ou par un *arbre binaire de recherche* (ABR) auquel cas on utilisera une recherche dans cet arbre.

Recherche Dichotomique dans un Tableau

La fonction de recherche dichotomique dont on a besoin aura la signature suivante :

`recherche_dichotomique` : (`'a` → `'b` → `ordre`) → `'a` → `'b` → `'b ensemble` → `'b`

Si `E` est un `'b ensemble` trié et si `comp : 'a -> 'b -> ordre` est une fonction de comparaison vérifiant pour tout élément a de type `'a`, tout élément b de type `'b` et tout élément b' de type `'b` plus petit que b :

- **(a)** `comp a b = PLUS_GRAND` implique `comp a b' = PLUS_GRAND`
- **(b)** `comp a b' = PLUS_PETIT` implique `comp a b = PLUS_PETIT`
- **(c)** `comp a b = EGAUX` implique `comp a b' = PLUS_GRAND`
- **(d)** `comp a b' = EGAUX` implique `comp a b = PLUS_PETIT`

alors pour tout élément `a` de type `'a` et tout élément `inf` de type `'b` tel que tous les éléments de `E` soit plus grand que `inf` et `comp a inf = PLUS_GRAND`, le résultat de `recherche_dichotomique comp a inf E` est le plus grand élément de `E` $\cup\{$`inf`$\}$ plus petit que `a` pour la relation de comparaison `comp`.

Question 3

Implémentez la fonction `recherche_dichotomique` décrite ci-dessus.

Préparer une réponse à donner à l'oral

Écrivez la preuve de correction de votre algorithme et faîtes l'analyse de complexité en fonction du nombre d'élément de `E`.

On définit l'ordre lexicographique sur $\mathbb{N}^2$:

$$\forall (x_1,y_1)\in\mathbb{N}^2, \forall (x_2,y_2)\in\mathbb{N}^2, (x_1,y_1)<(x_2,y_2) \iff x_1<x_2 \text{ ou } (x_1=x_2 \text{ et } y_1<y_2)$$

La librairie `kholle_localisation_de_point` fourni le type et les fonctions suivantes :

```
type ordre = PLUS_PETIT | PLUS_GRAND | EGAUX
val ordre_lex : point -> point -> ordre
val tri_ensemble : point ensemble -> point ensemble
```

implémentant l'ordre lexicographique défini ci-dessus et triant un ensemble selon cet ordre.

Question 4

Pour chaque paire d'ensemble `E` préalablement trié dans l'ordre lexicographique et d'entiers `x` donné ci-dessous, donnez les coordonnées du plus grand point de `E` dont l'abscisse est plus petite ou égale à `x` :
a) E_{212}, 121212 **b)** E_{512}, 212121 **c)** E_{1000}, 1112222

Recherche dans un Arbre Binaire de Recherche

La librairie `kholle_localisation_de_point` fourni les types suivants pour représenter les *arbres binaires de recherche* (ABR) :

```
type 'a abr = Vide | Noeud of 'a * 'a abr * 'a abr
```

La fonction de recherche dans un ABR dont on a besoin aura besoin a la signature suivante :

$$\texttt{recherche\_abr} : (\texttt{'a} \to \texttt{'b} \to \texttt{ordre}) \to \texttt{'a} \to \texttt{'b} \to \texttt{'b abr} \to \texttt{'b}$$

Si `E` est un `'b abr` trié et si `comp : 'a -> 'b -> ordre` est une fonction de comparaison vérifiant pour tout élément a de type `'a`, tout élément b de type `'b` et tout élément b' de type `'b` plus petit que b les propriétés **(a)**, **(b)**, **(c)** et **(d)** décrite précédemment, alors pour tout élément `a` de type `'a` et tout élément `inf` de type `'b` tel que tous les éléments de `E` soit plus grand que `inf` et `comp a inf = PLUS_GRAND`, le résultat de `recherche_abr comp a inf E` est le plus grand élément de `E` plus petit que `a` pour la relation de comparaison `comp`.

Question 5

Implémentez la fonction `recherche_abr` décrite ci-dessus.

Préparer une réponse à donner à l'oral

Écrivez la preuve de correction de votre algorithme et faîte l'analyse de complexité en fonction de la hauteur de l'ABR.

Recherche dans un Arbre Bicolore

Les *arbres bicolore, ou arbres rouge-noir* (ARN) sont des ABR où chaque nœud possède en plus de son étiquette une couleur, `ROUGE` ou `NOIR`, et doit vérifier les propriétés suivante :

- Les feuilles `Vide` sont colorées en noires.
- Chaque chemin de la racine à une feuille `Vide` doit rencontrer le même nombre de nœud `NOIR`. Ce nombre sera appelé *hauteur noire*.

- Chaque nœud coloré en `ROUGE` possède un parent coloré en `NOIR`. La racine est donc nécessairement colorée en `NOIR`.

La librairie `kholle_localisation_de_point` fourni les types suivants pour représenter les ARN.

```
type couleur = ROUGE | NOIR ;;
type 'a arn = ( 'a * couleur ) abr ;;
```

La recherche dans un ARN est une recherche dans un ABR.

Question 6

Implémentez une fonction `recherche_arn` dont la signature est

$$\texttt{recherche\_arn} : (\texttt{'a} \to \texttt{'b} \to \texttt{ordre}) \to \texttt{'a} \to \texttt{'b} \to \texttt{'b arn} \to \texttt{'b}$$

Et qui réalise la recherche dans l'ARN vu comme un ABR.

Préparer une réponse à donner à l'oral

Donnez la complexité de la recherche dans un ARN en fonction du nombre de nœuds de l'arbre.

3 Localisation de point

Algorithme de Localisation d'un point

Considérons un partitionnement en différentes zones du plan par un ensemble $\mathcal{S}$ de sommets et $\mathcal{A}$ d'arêtes. Chaque zone est identifiée par un entier. Étant donné un point de l'espace, le but est de renvoyer l'identifiant de la zone du plan dans laquelle il se trouve.

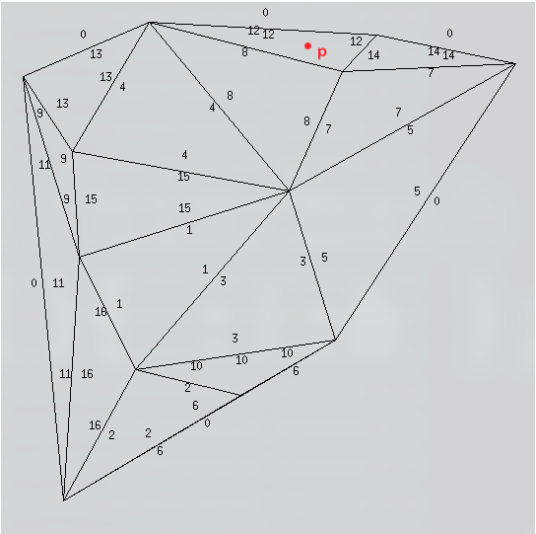

Figure 1: Un partitionnement et un point p dans la zone 12

Pour ce faire on va découper le plan en *dalles*. Une dalle est la données de deux abscisses $x_1 < x_2$ telle que tous les sommets (x, y) des zones partitionnant le plan vérifient $x_1 \leq x < x_2$. x_1 est la *borne inférieure* de la dalle et peut éventuellement valoir $-\infty$. x_2 est la *borne supérieure* de la dalle et peut éventuellement valoir $+\infty$. Intuitivement une dalle consiste en deux lignes horizontales à l'intérieur desquelles il n'y a aucun sommet, les sommets étant situés sur les lignes horizontales. Pour découper le plan en un nombre minimal de dalles, on fait passer une ligne horizontale par chaque sommet des zones partitionnant le plan.

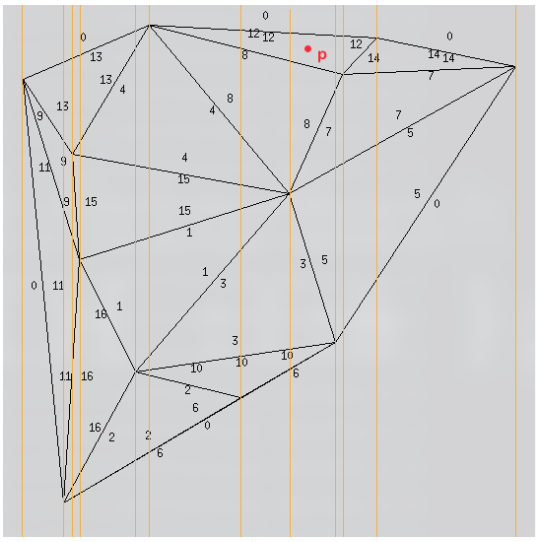

Figure 2: Dalles du partitionnement précédent

Une fois le plan découpé en dalle on peut remarquer que chaque arête *ou bien* traverse entièrement la dalle – la coupant en deux parties, une au-dessus, une en-dessous de l'arête – *ou bien* est intégralement en dehors de la dalle. Chaque dalle est ainsi partitionnée en bandes horizontales correspondant à exactement une zone du partitionnement du plan.

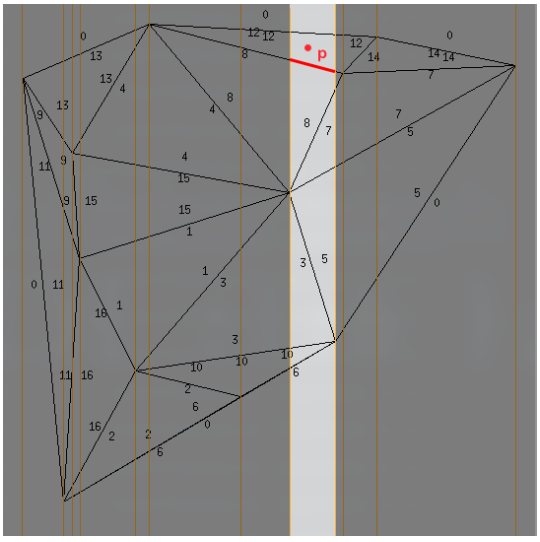

Figure 3: Localisation de p

Pour localiser un point, on cherche la dalle dans laquelle il se situe, puis on cherche l'arête la plus haute située en dessous du point dans la dalle (éventuellement l'arête sur laquelle se situe le point). La zone directement au dessus de cette arête est la zone dans laquelle se situe le point.
Pour pouvoir aboutir à une localisation logarithmique une fois la structure de donnée construite, les dalles seront sauvegardées dans un tableau trié. Les dalles construites par l'algorithme sont disjointes et juxtaposée, on peut les trier selon leur borne inférieure (de gauche à droite). Ce tableau de dalles sera nommé une *partition*.
Chaque dalle contiendra un *arbre binaire de recherche (ABR)* sauvegardant les segments qui traverse la dalle. La relation d'ordre entre les segments ne sera pas détaillée dans ce sujet, mais correspond intuitivement à ordonner les segments du plus bas au plus haut. On choisit d'utiliser des ABR particuliers, des *arbres bicolores*, ou *arbre rouge-noir (ARN)* pour obtenir une recherche logarithmique.

Génération aléatoire de zones

Nous ne nous étendrons pas sur la triangulation de Delaunay évoquée en introduction. On admettra qu'étant donné un ensemble de point du plan, le résultat de la triangulation est un partitionnement du plan en zones disjointes dont les sommets sont les éléments de l'ensemble. Si E est un ensemble de point du plan, on note $\mathcal{T}(E)$ le partitionnement du plan par la triangulation de Delaunay de E.
La librairie `kholle_localisation_de_point` fourni les types `segments`, `aretes` et `sommet` décrites ci-dessous.
Un segment est la donnée de deux points. Ils seront désignés respectivement comme *l'extrémité de départ et d'arrivée* du segment.

```
type segment = point * point (* extrémité de départ et d'arrivée *)
```

Une arête est la donnée d'un segment et de deux entiers. Le segment représente l'arête à proprement parler. L'entier `zone_droite` est l'identifiant de la zone situé directement à droite lorsque qu'on se déplace sur le segment de son extrémité de départ à son extrémité d'arrivée. L'entier `zone_gauche` est l'identifiant de la zone situé directement à gauche.

```
type arete = { segment : segment ;
               mutable zone_droite : int ;
               mutable zone_gauche : int }
```

Un sommet est la donnée d'un point et de deux listes d'arêtes. Le point représente le sommet à proprement parler. La liste des arêtes entrantes, est la liste de toutes les arêtes donc le sommet et l'extrémité d'arrivée. La liste des arêtes sortantes, est la liste de toutes les arêtes dont le sommet et l'extrémité de départ.

```
type sommet = { point : point ;
                mutable sortantes : arete list ;
                mutable entrantes : arete list }
```

La librairie `kholle_localisation_de_point` fourni la fonction :

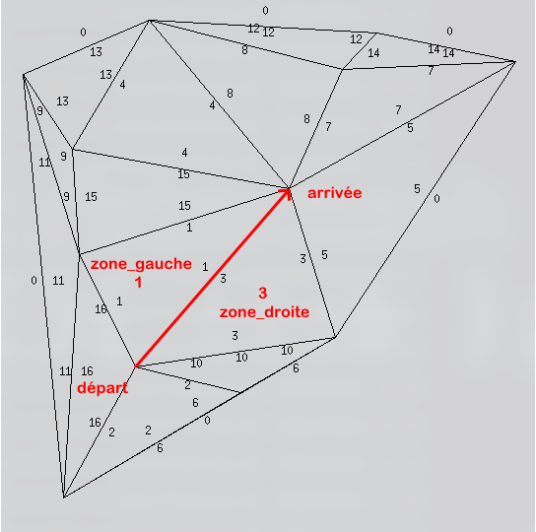

Figure 4: Arête

```
val delaunay_triangulation : point ensemble -> sommet ensemble
```

qui renvoie la liste des sommets (au sens du type `sommet` défini ci-dessus) d'un partitionnement du plan dont les sommets sont des points de l'ensemble passé en paramètre. Les sommets sont triés par ordre lexicographique sur leurs coordonnées.

Question 7

Donnez le nombre d'arêtes du partitionnement engendré par la fonction `delaunay_triangulation` appelée sur les ensembles suivant :
a) E_{212} **b)** E_{512} **c)** E_{1000}

Localisation dans une partition.

Question 8

Définissez un type de donnée pour représenter une dalle et un type de donnée pour représenter une partition, comme décrit en introduction.

La librairie `kholle_localisation_de_point` fourni la fonction :

```
val ordre_point_arete : point -> arete -> ordre
```

renvoyant la position relative d'un point appartenant à une dalle par rapport à une arête traversant la dalle. C'est-à-dire si un point est inférieur (en dessous), égal (sur) ou supérieur (au-dessus) à une arête.

Question 9

Implémentez une fonction `localisation_de_point : point → partition → int` qui prend en argument un point et une partition et renvoie la zone du plan dans laquelle se trouve le point.

4 Premier algorithme de création d'une partition

La première approche naïve pour créer une partition et de créer les dalles indépendamment les unes des autres en cherchant toutes les arêtes qui traverse la dite dalle et en les insérant une part une dans l'ARN de la dalle initialement `Vide`.

Question 10

Implémentez une fonction `traverse : arete → int → int → bool` qui prend en arguments une arête, la borne inférieure et la borne supérieure d'une dalle et qui renvoie vrai si l'arête traverse la dalle et faux sinon.

Question 11

Implémentez une fonction `creation_dalle : sommet ensemble → int → int → dalle` qui prend en arguments un partitionnement du plan, la borne inférieure, `inf`, et la borne supérieure, `sup`, d'une dalle et qui renvoie un élément de type `dalle` représentant la dalle du partitionnement ayant comme bornes `inf` et `sup`.

Question 12

Implémentez une fonction `creation_partition : sommet ensemble → partition` qui prend en argument un partitionnement du plan et renvoie sa représentation dans le type `partition` comme décrite précédemment par l'algorithme.

Question 13

Pour chaque paire de partitionnement et de point, donnez la zone du partitionnement dans laquelle se situe le point.
a) $\mathcal{T}(E_{212})$, $(212121, 1221)$ **b)** $\mathcal{T}(E_{512})$, $(2211, 14421)$ **c)** $\mathcal{T}(E_{1000})$, $(111111, 212121)$

Préparer une réponse à donner à l'oral

Quelle est la complexité temporelle de la fonction `creation_partition`, sa complexité spatiale ?

5 Algorithme de création d'une partition à l'aide d'ARN persistants

L'approche précédente est coûteuse en temps et en espace. On peut améliorer ces deux complexités en construisant les dalles dans l'ordre, de la plus à gauche à la plus à droite du plan, et en utilisant l'ARN construit pour une dalle pour construire l'ARN de la suivante.

Préparer une réponse à donner à l'oral

Expliquez comment construire l'ARN des arêtes traversant une dalle à partir de l'ARN des arêtes traversant la dalle précédente et des sommets appartenant à la dalle à construire (les sommets ayant comme abscisse la borne inférieure de la dalle).

La librairie `kholle_localisation_de_point` fourni les fonctions :

```
val insertion_arn : ( 'a -> 'a -> ordre ) -> 'a -> 'a arn -> 'a arn
val suppression_arn : ( 'a -> 'a -> ordre ) -> 'a -> 'a arn -> 'a arn
```

qui prennent en paramètre un `'a arn`, `A`, une relation de comparaison sur les éléments de type `'a` et un élément de type `'a`, `e` et qui insère ou supprime `e` dans `A`.
On notera que ces fonctions renvoient un nouvel ARN. Par ailleurs elles ont été implémentées de telle sorte que tous les sous-arbres de `A` ne soit pas copiés. Seulement un nombre de nœuds logarithmique en le nombre de nœuds de `A` sont nouvellement créés. Cela signifie que si `A` est un ARN de n nœuds et `A'` est l'ARN renvoyé par la fonction d'insertion ou de suppression, `A'` possède $n-1 \leq m \leq n+1$ nœuds, mais seulement $\mathcal{O}(\log(n))$ nœuds ne sont pas commun à `A`. L'espace occupé par `A et A'` est donc $n + \mathcal{O}(\log(n))$.

Question 14

Implémentez une fonction `creation_partition_persistent : sommet ensemble →` `partition` qui prend en argument un partitionnement du plan et renvoie sa représentation dans le type `partition` comme décrite précédemment par l'algorithme.

Préparer une réponse à donner à l'oral

Donnez la complexité temporelle et spatiale de votre algorithme.

6 Ouverture sur la création d'une partition de taille $\mathcal{O}(n)$

Sarnak et Tarjan proposent dans un papier de 1986 un algorithme permettant le calcul d'une partition de taille $\mathcal{O}(n)$ avec n le nombre d'arêtes qui permet la localisation en $\mathcal{O}(\log(n))$. Nous ne pourrons pas étudier cette solution entièrement ici.
L'une des idées de base consiste à considérer comme partition un ARN modifié de manière à ce que pour chaque nœud, le fils gauche et droit soient paramétrés par les dalles du plan identifiée par sa borne inférieure. Pour économiser de l'espace toutes les dalles n'apparaissent pas nécessairement dans le paramétrage. On considère alors que c'est la dalle de gauche la plus proche apparaissant dans l'arbre qui désigne dans quel sous-arbre voyager.
Voici le type d'une partition telle que proposée par Sarnak et Tarjan :

```
type 'a gros_arn =
    Vide
  | 'a * ( int * ( 'a gros_arn ) arn ) * ( int * ( 'a gros_arn ) arn ) ;;
type partition_ST = ( int * ( 'a gros_arn ) arn )
```

Question 15

Implémentez une fonction de recherche d'un élément de type `'a` dans un `'b gros_arn` utilisant une fonction de comparaison `comp 'a 'b` vérifiant les mêmes propriétés que les fonctions de comparaison utilisées dans la section *Recherches logarithmique*.
On pourra utiliser la fonction `ordre_int` de la librairie qui compare deux entier classiquement et renvoie un type `ordre`.

Préparer une réponse à donner à l'oral

Donnez la complexité temporelle de votre algorithme.

Fiche réponse : Localisation de Point

Nom, prénom : DOUZ Nadine $\widetilde{u_0}$: 12

Question 1:

a) 168156

b) 397392

c) 160164

Question 4:

a) (119570, 778808)

b) (2077726, 104116)

(111192, 565738)

Question 7:

c) 616

d) 1517

e) 2983

Question 13:

a) 75

b) 973

c) 1722