Divide-and-Conquer Algorithms
Application to the Closest Pair Problem

2 Specification and Brute-Force Algorithm

Question 1

On spécifie simplement que les deux indices sont dans les bornes et sont distincts.

Question 2

Les deux points d'un résultat correct entre `low` et `high` doivent être espacés de la distance δ indiquée dans le résultat, leurs indices doivent vérifier `distinct_indices_in_range` dans l'intervalle (`low`, `high`), et toute paire de points d'indices vérifiant `distinct_indices_in_range` entre `low` et `high` doit être espacée d'au moins δ.

Question 3

`closest_pair_post` est simplement `closest_pair_post_for` entre 0 et la longueur du tableau.

Question 4

On impose $0 \leq$ `low` $<$ `high` $\leq$ `a.length` comme précondition évidente à `brute_force_search_sub_array`. On veut aussi spécifier que `low` et `high` sont espacés d'au moins 2 : j'ai choisi de le faire en imposant `low` $<$ `low`$+1 <$ `high` pour que l'initialisation de l'invariant `distinct_indices_in_range`soit plus claire.
On utilise comme invariants de boucle les trois membres de la conjonction qui définit `closest_pair_post_for`, et on les répète dans la boucle interne (avec les changements nécessaires à la quantification).
Pour `brute_force_search`, la seule précondition nécessaire est `a.length` $<$ 2 comme voulu.

3 Closest Pair in 1D

On introduit le lemme `middle_are_closer` qui est utile pour toute la section. Il justifie de ne considérer que les paires de points voisins dans un tableau trié par abscisse, et aussi de ne considérer qu'une paire de points venant de deux moitiés de tableau différentes dans l'algorithme diviser pour régner.

J'ai choisi de ne pas définir ce lemme en tant que fonction ghost pour ne pas avoir à l'appeler manuellement : il se trouve que les prouveurs arrivent assez bien à l'utiliser automatiquement en peu de temps pour prouver des invariants de forme `dist a[i] a[l]` $\geq$ `dist a[j] a[k]`, probablement parce qu'il existe peu d'autres façons d'arriver à un résultat de cette forme.

Question 5

Comme le tableau est trié par abscisse, on peut se contenter d'explorer les paires de points voisins. On utilise comme invariants de boucle les trois membres de la conjonction qui définit `closest_pair_post_for`, en donnant une indication `dist a[k] a[l]` $\geq$ `dist a[k] a[k+1]` $\geq$ `min` pour inciter les prouveurs à utiliser le lemme `middle_are_closer`.

Question 6

La première assertion introduit une disjonction de cas sur les positions d'une paire de points dans le tableau, l'idée étant de prouver que toute paire de points est espacée de plus que le résultat dans chaque cas séparément. Les trois assertions suivantes traitent chacune un cas. Selon la distance la plus petite parmi `d`, `r1.delta` et `r2.delta`, on conclut la disjonction de cas avec une dernière assertion. Ces assertions ne sont pas strictement nécessaires pour la preuve, mais diminuent beaucoup le temps de calcul et permettent de ne pas utiliser CVC3.

Question 7

On utilise une assertion pour indiquer au prouveur d'utiliser que les tableaux `a` et `b` sont égaux à permutation près pour montrer que toutes les paires de points sont éloignées d'au moins `r.delta`.

4 Closest Pair in 2D

Question 8

Les invariants dans la double boucle `for` sont encore les trois membres de la conjonction qui définit `closest_pair_post_for`, à condition qu'on ait trouvé une paire espacée de moins de δ. Si on atteint une différence d'ordonnée supérieure à `!m`, alors on peut affirmer que la distance entre `a[i]` et les points qui suivent est supérieure à `!m` et sortir de la boucle interne.
Dans le cas où on a trouvé une paire de points espacée de moins de δ, on indique au prouveur d'utiliser le fait que `a` et `b` sont égaux à permutation près pour montrer que toutes les paires de points sont éloignées d'au moins δ. Les deux autres assertions ne sont pas nécessaires mais diminuent le temps de preuve. J'ai du mal à expliquer pourquoi, surtout pour l'assertion

`distinct_indices_in_range f' s' 0 a.length` : elle est prouvée très rapidement, mais si on ne la met pas, il faut plusieurs secondes de plus pour montrer la postcondition.
Si on n'a pas trouvé de paire espacée de moins de δ, on conclut en utilisant que `b` est une permutation de `a`.

Question 9

left_border
On peut simplement utiliser `!l` comme variant et les postconditions comme invariants.

right_border
On parcourt encore la moitié de tableau de droite à gauche, ce qui permet d'utiliser les mêmes variants et invariants que dans `left_border`.

Question 10

On prend comme variant `high - low`. La variable `mu` calculée dans le corps de la fonction convient à la précondition de `search_strip` : la première assertion sert à indiquer cela au prouveur. On doit ensuite montrer que toute paire de points est espacée de plus que le résultat. On introduit une disjonction de cas sur les positions des points de chaque paire avec la deuxième assertion. Deux de ces cas sont traités par les deux assertions qui suivent. On traite ensuite le cas où la paire est comprise entre `left` et `right` selon que la fonction `search_strip` a réussi ou échoué à trouver une paire de points moins espacée que les résultats de chaque appel récursif. Dans le cas de succès, on ajoute aussi l'assertion `forall i j. distinct_indices_in_range i j 0 temp.length -> dist temp[i] temp[j] >=. res.delta` . C'est une postcondition de `search_strip` (une conséquence du prédicat `closest_pair_post_for`), mais la rappeler ici diminue beaucoup le temps de preuve.

Question 11

On applique simplement la fonction précédente. L'assertion indique au prouveur d'utiliser le fait que `a` et `b` sont égaux à permutation près.

5 Complexity Analysis of the Processing of the Strip

Question 12

On utilise les invariants de boucle suivants : les tableaux `left` et `right` sont strictement ordonnés, leur contenu décrit le tableau `side`, et contiennent des indices compris entre k et i ($k+6$ à la fin de la boucle). On doit aussi maintenir que les longueurs des tableaux sont comprises entre 0 et 7 et de somme $i-k$ (7 à la fin de la boucle). Ceci sert à montrer qu'un des tableaux est de longueur supérieure ou égale à 4 à la fin de la boucle.

Question 13

On applique simplement une translation dans le cas où `side` est vrai et une symétrie et une translation (qui se combinent exactement en une utilisation de l'axiome `dist_inv_by_symmetry`) dans le cas contraire.

Question 14

On introduit le tableau ghost `bside`, qui est une copie de `side` à laquelle on a appliqué la permutation p. On montre également que `bside` est à `b` ce que `side` est à `a`, ce qui est une des préconditions de `too_many_points_in_rectangle` plus tard. En plus de l'invariant `j <= i + 6`, on ajoute aussi à la boucle principale l'invariant `!m <=. delta` . Ceci est nécessaire pour montrer une des préconditions de `too_many_points_in_rectangle`, à savoir que la différence d'ordonnée entre `b[i]` et les points de `b` entre les indices i et $i+6$ est inférieure à δ (c'est clairement vrai pour `!m`). Les deux assertions dans la boucle interne permettent chacune de démontrer une précondition de `too_many_points_in_rectangle` en utilisant le fait que `b` est une permutation de `a`.