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. Cette dernière n'est pas nécessaire pour la preuve, mais diminue beaucoup le temps de calcul.

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.