Will Franzen
CSDS 293 – Programming 6 Pseudocode

1) Algorithm Overview:

Narrative Section –

• The purpose of the createGalleryPairs algorithm is to consolidate pairings of Dali and Picasso paintings into a dictionary of sorted order with the keys and values as the artists respectively. The specified constraints for these pairs require the heights of the Picasso to be less than the Dali and for the paintings to be ordered in non-decreasing price. Additionally, the final display should be of the size that allows for the maximum number of pairings that meets the required constraints.

• Assuming the given input is a single collection of paintings each containing information about its artist, height, and price, it is necessary to separate these paintings into two separate arrays by their respective artists and ordering them by price. This can be achieved by using a simple getter method follow by a compare method. The artist is stored as a String and the height and price as numeric variables.

• A few additional methods are used for additional abstraction and information hiding that will be touched on in the pseudocode section. Methods I will not include as pseudocode:

- validate methods for my Paintings class
- toString methods for my Paintings class or ArtGallery class
- compare method for my Paintings class
    - (return negative if smaller, 0 if equal, positive if larger)

Explanatory Diagrams –

Initial Collection: (*Artist [Dali/Picasso], Height, Price)*

P, 4, 20	P, 3, 30	P, 1, 100	D, 3, 20	D, 2, 10	D, 6, 100	P, 2, 40	D, 4, 25	P, 5, 10	D, 5, 30

Dali and Picasso Collections (pre-ordering):

D:

D, 3, 20	D, 2, 10	D, 6, 100	D, 4, 25	D, 5, 30

P:

P, 4, 20	P, 3, 30	P, 1, 100	P, 2, 40	P, 5, 10

Final Dali and Picasso Collections (post-ordering):

D:

D, 2, 10	D, 3, 20	D, 4, 25	D, 5, 30	D, 6, 100

P:

P, 5, 10	P, 4, 20	P, 3, 30	P, 2, 40	P, 1, 100

Expected Outcome:

D, 5, 30	D, 6, 100
P, 4, 20	P, 3, 30

2) *Algorithm Pseudocode*:

```
dalis, picassos 🡐 non-empty non-null sorted arrays of paintings for respective artist
galleryPairs 🡐 empty dictionary to hold dali and picasso pairs meeting constraints

createGalleryPairs( )
{
    Error Check: collections and elements non-null

    pIndex 🡐 index of current picassos array beginning at -1
    dIndex 🡐 index of current dalis array beginning at -1
    do
    {
            pIndex 🡐 pIndex + 1
            dIndex 🡐 dIndex + 1

        if isPicassoShorter(pIndex, dIndex) then
        {
            galleryPairs insert key: dalis[dIndex] and value: picassos[pIndex]
        }
        else if isPicassoShorter(pIndex + 1, dIndex) then
        {
            galleryPairs insert key: dalis[dIndex] and value: picassos[pIndex + 1]
            pIndex 🡐 pIndex + 1
        }
    }
    while pIndex < length of picassos - 1 OR dIndex < length of dalis
}


isPicassoShorter( current index in picassos,  current index in dalis)
{
    isShorter 🡐 Boolean denoting if height of picasso at pIndex is smaller than height of dali
            at dIndex
    return isShorter
}
```

3) *Justifications*:

• I wanted to keep my algorithm short and easy to understand, a requirement I feel that I was able to meet. The reasoning behind my chosen data structures (arrays and dictionary) is I can reach a highly optimal time and space complexity that I will explore in my runtime analysis as well as the fact they are simple and easy to convert.

• isPicassoShorter was created for ease of understanding my main algorithm. Although it only replaces what would have only been two conditional statements, the conditional statements would be long and quite harder to understand from a quick glance.

• createGalleryPairs will always find a solution of maximum possible length while meeting the required constraints given. The picasso and dali arrays are already sorted first by price and then height in the event of equality, therefore I need not worry about any element passed over as it would no longer meet the increasing price constraint. If the elements are the specified indexes meet the constraints they should be added to the dictionary with the dali as the key and picasso as its value. As these values are added they will remain sorted which is the goal. If the corresponding elements do not form an eligible pair then it check the next succeeding picasso against that same dali. If those two form a match then the pIndex is incremented so the first case will be met during the next iteration. Finally, if neither of these two cases produce results then dIndex will be incremented hoping to find pairs that work.

4) *Runtime Analysis*:

• The runtime of my program is worst-case $O(n \log n)$ and best-case $O(n)$.

• Because of the data structures I decided to use, indexing an array runs at a time of O(1). However, in worst-case I would need to check all 2n indexes (size of dalis and picassos = n) and best case I would check a fraction of those, assuming non-zero length, so it can be generalized to O(n). Additionally, insertion into a dictionary, and what we would likely wanted to be a sorted map/tree map, is O(logn) time. At worst-case every pair check will be inserted at a time of O(logn) and at best-case no pairs will be inserted.

5) *Operation Examples*:

(Continuing from the final dali and picasso collections in the overview):

<u>First Iteration</u>:

dIndex = 0

D, 2, 10	D, 3, 20	D, 4, 25	D, 5, 30	D, 6, 100

pIndex = 0

P, 5, 10	*P, 4, 20*	P, 3, 30	P, 2, 40	P, 1, 100

finalGalleryPairs = {}

isPicassoShorter(0, 0) → False, height at picassos[0] > height at dalis[0]
isPicassoShorter(1, 0) → False, height at picassos[1] > height at dalis[0]

Second Iteration:

dIndex = 1

D, 2, 10	**D, 3, 20**	D, 4, 25	D, 5, 30	D, 6, 100

pIndex = 1

P, 5, 10	**P, 4, 20**	*P, 3, 30*	P, 2, 40	P, 1, 100

isPicassoShorter(1, 1) → False, height at picassos[1] > height at dalis[1]
isPicassoShorter(2, 1) → False, height at picassos[2] > height at dalis[1]

Third Iteration:

dIndex = 1

D, 2, 10	**D, 3, 20**	D, 4, 25	D, 5, 30	D, 6, 100

pIndex = 1

P, 5, 10	**P, 4, 20**	*P, 3, 30*	P, 2, 40	P, 1, 100

isPicassoShorter(1, 1) → False, height at picassos[1] > height at dalis[1]
isPicassoShorter(2, 1) → False, height at picassos[2] > height at dalis[1]

Fourth Iteration:

dIndex = 2

D, 2, 10	D, 3, 20	**D, 4, 25**	D, 5, 30	D, 6, 100

pIndex = 2

P, 5, 10	P, 4, 20	**P, 3, 30**	*P, 2, 40*	P, 1, 100

isPicassoShorter(2, 2) → True, height at picassos[2] < height at dalis[2]
finalGalleryPairs = { [(D,4,25) , (P,3,30)] }

Fifth Iteration:

dIndex = 3

D, 2, 10	D, 3, 20	D, 4, 25	**D, 5, 30**	D, 6, 100

pIndex = 3

P, 5, 10	P, 4, 20	P, 3, 30	**P, 2, 40**	P, 1, 100

isPicassoShorter(3, 3) → True, height at picassos[3] < height at dalis[3]
finalGalleryPairs = { [(D,4,25) , (P,3,30)], [(D,5,30) , (P,2,40)] }

Six Iteration:

dIndex = 4

D, 2, 10	D, 3, 20	D, 4, 25	D, 5, 30	**D, 6, 100**

pIndex = 4

P, 5, 10	P, 4, 20	P, 3, 30	P, 2, 40	**P, 1, 100**

isPicassoShorter(4, 4) → True, height at picassos[4] < height at dalis[4]
finalGalleryPairs = { [(D,4,25) , (P,3,30)], [(D,5,30) , (P,2,40)], [(D,6,100) , (P,1,100)] }