# Hierarchical Clustering

Hierarchical clustering is one of the popular and easy to understand clustering technique. This clustering technique is divided into two types:

1. Agglomerative
2. Divisive

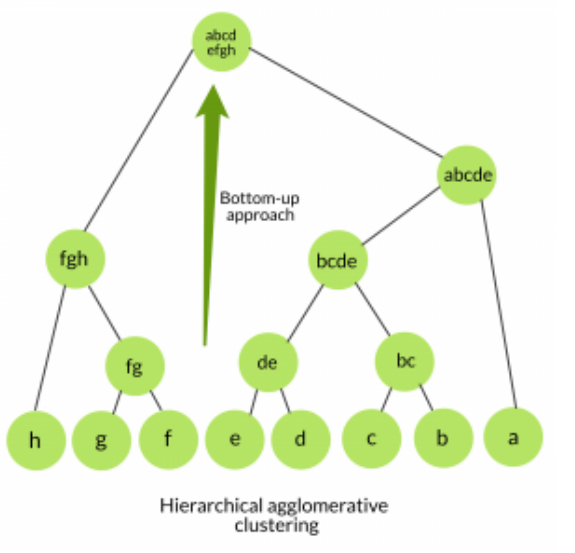
**Agglomerative Clustering:** Also known as bottom-up approach or hierarchical agglomerative clustering (HAC). A structure that is more informative than the unstructured set of clusters returned by flat clustering. This clustering algorithm does not require us to prespecify the number of clusters. Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerates pairs of clusters until all clusters have been merged into a single cluster that contains all data.

## Algorithm:

### Steps to Perform Agglomerative Hierarchical Clustering

We are going to explain the most used and important Hierarchical clustering i.e. agglomerative. The steps to perform the same is as follows −

- **Step 1** − Treat each data point as single cluster. Hence, we will be having, say K clusters at start. The number of data points will also be K at start.
- **Step 2** − Now, in this step we need to form a big cluster by joining two closet datapoints. This will result in total of K-1 clusters.
- **Step 3** − Now, to form more clusters we need to join two closet clusters. This will result in total of K-2 clusters.
- **Step 4** − Now, to form one big cluster repeat the above three steps until K would become 0 i.e. no more data points left to join.
- **Step 5** − At last, after making one single big cluster, dendrograms will be used to divide into multiple clusters depending upon the problem.



Hierarchical agglomerative clustering

Python implementation of the above algorithm using scikit-learn library:

**Role of Dendrograms in Agglomerative Hierarchical Clustering**

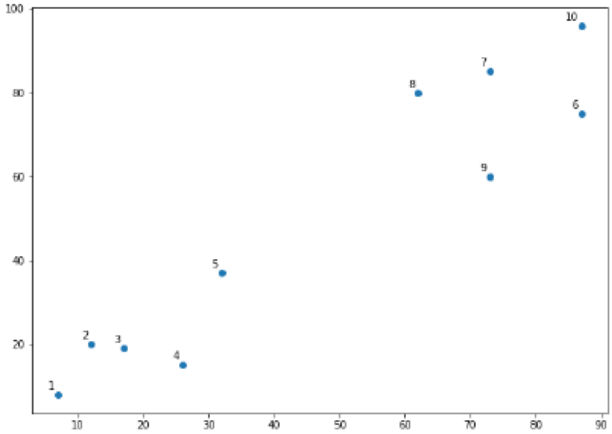
As we discussed in the last step, the role of dendrogram starts once the big cluster is formed. Dendrogram will be used to split the clusters into multiple cluster of related data points depending upon our problem. It can be understood with the help of following example −

**Example 1**

To understand, let us start with importing the required libraries as follows −

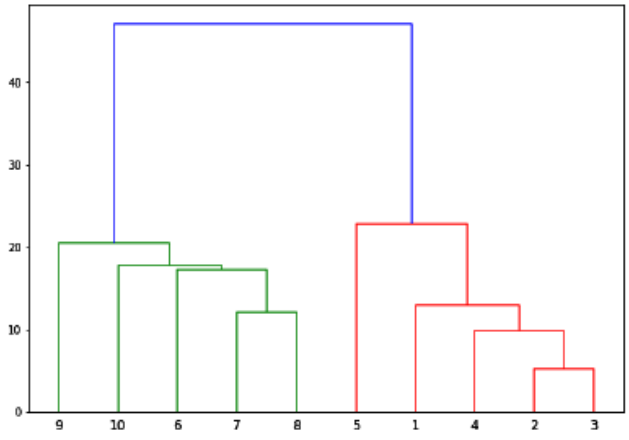
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

```
X = np.array(
   [[7,8],[12,20],[17,19],[26,15],[32,37],[87,75],[73,85], [62,80],[73,60],[87,96],])
labels = range(1, 11)
plt.figure(figsize = (10, 7))
plt.subplots_adjust(bottom = 0.1)
plt.scatter(X[:,0],X[:,1], label = 'True Position')
for label, x, y in zip(labels, X[:, 0], X[:, 1]):
   plt.annotate(
      label,xy = (x, y), xytext = (-3, 3),textcoords = 'offset points', ha = 'right', va = 'bottom')
plt.show()
```

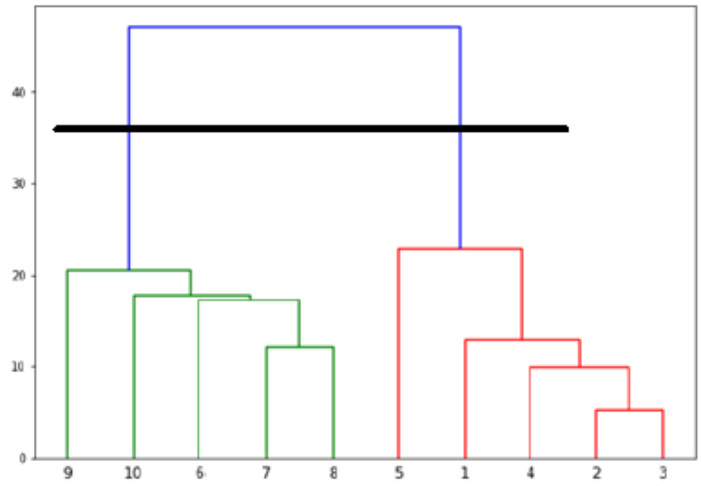



From the above diagram, it is very easy to see that we have two clusters in out datapoints but in the real world data, there can be thousands of clusters. Next, we will be plotting the dendrograms of our datapoints by using Scikt library –

```
from scipy.cluster.hierarchy import dendrogram, linkage
from matplotlib import pyplot as plt
linked = linkage(X, 'single')
labelList = range(1, 11)
plt.figure(figsize = (10, 7))
dendrogram(linked, orientation = 'top',labels = labelList,
   distance_sort ='descending',show_leaf_counts = True)
plt.show()
```




Now, once the big cluster is formed, the longest vertical distance is selected. A vertical line is then drawn through it as shown in the following diagram. As the horizontal line crosses the blue line at two points, the number of clusters would be two.

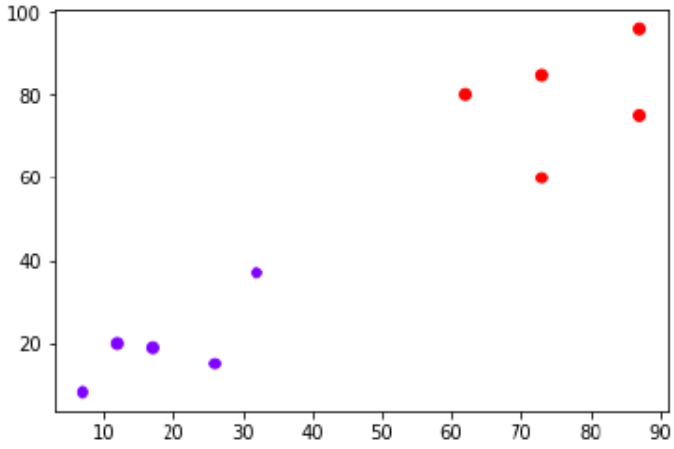



Next, we need to import the class for clustering and call its fit_predict method to predict the cluster. We are importing AgglomerativeClustering class of sklearn.cluster library –

```
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters = 2, affinity = 'euclidean', linkage = 'ward')
cluster.fit_predict(X)
```

Next, plot the cluster with the help of following code –

```
plt.scatter(X[:,0],X[:,1], c = cluster.labels_, cmap = 'rainbow')
```

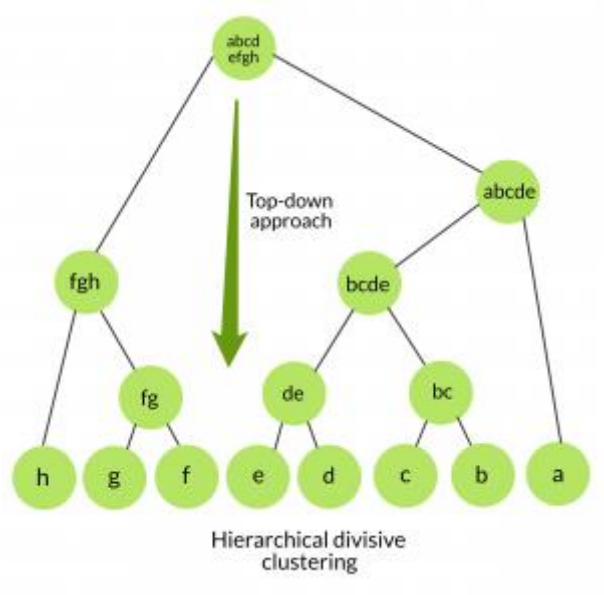



The above diagram shows the two clusters from our datapoints.

**Divisive Clustering :** Also known as top-down approach. This algorithm also does not require to prespecify the number of clusters. Top-down clustering requires a method for splitting a cluster that contains the whole data and proceeds by splitting clusters recursively until individual data have been splitted into singleton cluster.

**Algorithm:**

```
given a dataset (d1, d2, d3, ....dN) of size N
at the top we have all data in one cluster
the cluster is split using a flat clustering method eg. K-Means etc
repeat
choose the best cluster among all the clusters to split
split that cluster by the flat clustering algorithm
untill each data is in its own singleton cluster
```




**Hierarchical Agglomerative *vs* Divisive clustering –**

- Divisive clustering is more *complex* as compared to agglomerative clustering, as in case of divisive clustering we need a flat clustering method as “subroutine” to split each cluster until we have each data having its own singleton cluster.

- Divisive clustering is more *efficient* if we do not generate a complete hierarchy all the way down to individual data leaves. Time complexity of a naive agglomerative clustering is **$O(n^3)$** because we exhaustively scan the N x N matrix dist_mat for the lowest distance in each of N-1 iterations. Using priority queue data structure we can reduce this complexity to **$O(n^2 \log n)$**. By using some more optimizations it can be brought down to **$O(n^2)$**. Whereas for divisive clustering given a fixed number of top levels, using an efficient flat algorithm like K-Means, divisive algorithms are linear in the number of patterns and clusters.

- Divisive algorithm is also more *accurate*. Agglomerative clustering makes decisions by considering the local patterns or neighbor points without initially taking into account the global distribution of data. These early decisions cannot be undone. whereas divisive clustering takes into consideration the global distribution of data when making top-level partitioning decisions.