Algorithm Complexity

In [8]:
```
def get_centroid(data_points, iteration, k, groups):

    centroids =[]

    if iteration == 0:
        #Choosing random centroid in the first iteration.
        for i in range(k):
            centroids.append(data_points[np.random.randint(len(data_points))])
```

O(K)

```
    else:
        #looping on each group to get the mean of each point to get the centroid.
        for g in range(len(groups)):
            group = groups[g]
            temp = []
            #looping on the points inside the sample points ---> ex: sample 1 :[x, y, z,        w,    l]
            for n in range(len(data_points[0])):
                num = 0
```

O(K)

#H ,w ,bmi, length shoulder-arms

O(5) → no. of columns

```
                for i in range(len(group)):

                    sample_index = group[i]
                    num += data_points[sample_index][n]

                    average = (num / (len(group)))
```

O(N) → worst case: all samples may be on the same cluster

```
                # the temp value after each iteration will add the mean of each point # ex: first iter --> [x]
                temp.append(average)
            #at the end each temp array will represent a centroid in the centroids array --> [ [c1] , [c2] , [c3]
            centroids.append(temp)
    return centroids
```

total O(5xKxN) + O(K) → small

K_means function

- **Input:** the function takes the dataframe , k (No. of Clusters) , The no. of iterations.
- **Output:** Array containing the cluster no. that corresponds to the sample.

In [9]:
```
def k_means_clustering(data , k , iter_numbers):
    #converting the dataframe points into list of lists. [[sample1 points] , [sample2 points]]
    data_points = data.values.tolist()
    centroids = []
    groups = [] #representing the clusters in to groups [[g1] , [g2] , [g3]]

    for iteration in range(iter_numbers):
```

O(i)

```
        centroids = get_centroid(data_points,iteration,k,groups)
```

→ O(5xKxN) + O(K)

```
        clusters = []
        for i in range(len(data_points)):
```

O(N)

```
            distances = []

            for j in range(k):
                c = centroids[j]
                p = data_points[i]
```

O(K)

```
                d = 0
                for n in range(len(c)):
                    d +=((c[n]-p[n])**2)
```

O(5) 5 features.

```
                d = math.sqrt(d)
                distances.append(d)


            group_index=distances.index(min(distances))
            clusters.append(group_index)

            ## by the end of this loop we will have the clusters array representing each sample by the array index
            ## means that if the index was zero this represent sample 1 and array[0] = cluster that it belong to.


            groups = []
            unique_indexes = np.unique(clusters)
            for g in unique_indexes:
                pos = list(np.where(np.array(clusters) == g)[0])
                groups.append(pos)
```

O(N)

```
            #representing the clusters in to groups [[g1] , [g2] , [g3]] we will use it to find the new centroid.

    return clusters
```

The K_Means function complexity.

$$\rightarrow O(i)\left[\underbrace{O(5 \times N \times K)}_{\text{Centroid function}} + O(5 \times N \times K) \times O(N)\right]$$

→ simplifying

① ∵ 5 is very small compared to N

N ∵ no. of samples. ∴ Neglected

② K is either equal 3 or 5 ∴ also neglected

③ we Endup $O(i)\left[O(N) + O(N) \times O(N)\right]$

∵ iterations is small compared to N.

∴ Complexity is

$$O(N^2)$$