PART 1: Time Complexty is O(N^2)

In [14]:
```
A = [190, 220, 410, 580, 640, 770, 950, 1100, 1350]
def optimal(List):
    path=[]
    penalties=[]
    for i in range (len(List)):
        penalties.append(pow((200 - List[i]), 2))
        path.append(i)
        for j in range(i):
            temp=penalties[j]+pow((200 - (List[i] - List[j])), 2)
            if temp<penalties[i]:
                penalties[i]=temp
                path[i]=j+1
    finalPath = []
    index=len(path)-1
    while index >= 0:
        finalPath.append(index + 1);
        index = path[index] - 1;

    finalPath.reverse()
    print('Optimal Sequence :',finalPath)
    print('Penalties :', penalties)


optimal(A)
```

```
Optimal Sequence : [1, 3, 4, 6, 7, 8, 9]
Penalties : [100, 400, 500, 1400, 1400, 1500, 1900, 4400, 6900]
```

Part 2: Time Complexty is O(N)

In [1]:

```
#A utility function to check whether a word is present in dictionary or not.
def Dictionary(word):
    dictionary=["the","flower","in","from","high","i", "like","and","go","i","like","ice","cream"]
    size=len(dictionary)/len(dictionary[0])
    for i in range(int(size)):
        if dictionary[i]==word:
            return 1
    return 0
#Returns true if string can be segmented into space separated
#words, otherwise returns false
def Dict(str):
    size=len(str)
    if size==0:
        return 1
    wb=[]
    for x in range(size+1):
        wb.append(0)
# Create the DP table to store results of subroblems. The value wb[i]
# will be true if str[0..i-1] can be segmented into dictionary words,
# otherwise false.
    for i in range(1,size+1):
# if wb[i] is false, then check if current prefix can make it true.
# Current prefix is "str.substr(0, i)"
        temp1=Dictionary(str[0:i])
        if wb[i]==0:
            if temp1==1:
                wb[i]=1;
#wb[i] is true, then check for all substrings starting from
# (i+1)th character and store their results.
        if wb[i]==1:
#If we reached the last prefix
            if i==size:
                return 1
            for j in range(i+1,size):
#Update wb[j] if it is false and can be updated
#Note the parameter passed to dictionaryContains() is
#substring starting from index 'i' and length 'j-i'
                temp2=Dictionary(str[i:j-1])
                if wb[j]==0:
                    if temp2==1:
                        wb[j]=1
                if j==size:
                    if wb[j]==1:
                        return 1
    return 0
```

Part 3: Time Complexty is O(NlogN)

```
class Solution(object):
    def mergeKLists(self, lists):
        amount = len(lists)
        interval = 1
        while interval < amount:
            for i in range(0, amount - interval, interval * 2):
                lists[i] = self.merge2Lists(lists[i], lists[i + interval])
            interval *= 2
        return lists[0] if amount > 0 else lists

    def merge2Lists(self, l1, l2):
        head = point = ListNode(0)
        while l1 and l2:
            if l1.val <= l2.val:
                point.next = l1
                l1 = l1.next
            else:
                point.next = l2
                l2 = l1
                l1 = point.next.next
            point = point.next
        if not l1:
            point.next=l2
        else:
            point.next=l1
        return head.next
```

Part 4: Time Complexty is O(Nlog)

```
#function to delete zeros given list
def delete_zeros(lis):
    output=[]
    for i in lis:
        if i>0:
            output.append(i)
    return output
#function taking a graph of all friends relations
def invited(graph):
    L=[] #to hold invited list
    for a in graph:#Firstly added all friends in invited list,in next steps they will be eliminated.
        L.append(a)

    for node in range(1,len(L)):
        if len(graph[node+1])<5:#friends have not have 5 more friends,they will be zero
            for i in range(1,len(L)+1):
                for j in range(len(graph[i])):
                    if graph[i][j]==L[node]:
                        graph[i][j]=0
                graph[i]=delete_zeros(graph[i])#eliminate the zeros in graph
            L[node]=0

    for node in range(len(L)):
        if len(graph[node+1])<5:
            L[node]=0
            del (graph[node+1])
    L=delete_zeros(L)#delete zeros in invited List (divide)
    a=len(L)
    index=-1
    for node in L:
        index+=1
        if len(graph[node])>a-5:#friend have more than relations (all-5)
            for j in range(len(graph[node])):
                graph[node][j]=0
            graph[node]=delete_zeros(graph[node])#eliminate the zeros here
            L[index]=0#change with 0 node we would eliminate
    index2=-1
    for node in L:
        index2+=1
        if len(graph[node+1])>a-5:
            L[index2]=0
            del (graph[node+1])
    L=delete_zeros(L)

    return print("Alice can invite",len(L),"friends.The list of the invited friends is",L)
```

Part 5: Time Complexty is O(N)

In [1]:

```
def sat(m):
    L=[] #put variables in list
    control=0#to control whether its satisfied
    for current in m:#to compare al variable with one of them
        for node in m:#during list length
            if current==node:#if tey are equals continue
                control+=1#increase 1 control
                if control>=len(m):#if it is on last variable
                    return 1#it measn all variable comparisons finished so return true
            return 0#else return 0
```