In [1]:
```
##Importing all the required libraries
import random
import pickle
import matplotlib.pyplot as plt
```

In [2]:
```
# Loading weights
file_weights = 'weights'
file = open(file_weights, 'rb')
weights = pickle.load(file)
file.close()
```

In [3]:
```
#Loading prices
file_prices = 'prices'
file = open(file_prices, 'rb')
prices = pickle.load(file)
file.close()
```

In [4]:
```
#Loading carts
file_carts = 'carts'
file = open(file_carts, 'rb')
carts = pickle.load(file)
file.close()
```

In [5]:
```
#Loading adjacency lists
file_adj_list = 'adj_list'
file = open(file_adj_list, 'rb')
list_of_adj_lists= pickle.load(file)
file.close()
```

In [6]:
```
#Loading f values
file_f = 'f'
file = open(file_f, 'rb')
f = pickle.load(file)
file.close()
```

In [7]:
```
#Loading g values
file_g = 'g'
file = open(file_g, 'rb')
g = pickle.load(file)
file.close()
```

In [12]:
```
#In node greedy price of 1 item must be changed at a time. Since th
reshold is of 7%
#Change in price is generated re=andomly
delta_u=random.uniform(0,0.07)
print(delta_u)
```

```
0.0031954555355915683
```

In [13]:
```
#Creating a list of dictionaries.
#A dictionary corresponding to each experiment is created
#key->product
#value->revenue of the product
temp_dict=dict()
original_revenue_dict_list=[]
for experiment in range(10):
    revenue_dict=temp_dict.copy()
    for product in range(500):
        x=0
        x+=(prices[product]*f[experiment][product])
        for neighbour in list_of_adj_lists[experiment][product]:
            if(product<neighbour):
                x+=(prices[product]*g[experiment][(product,neighbou
r)])
            else:
                x+=(prices[product]*g[experiment][(neighbour,produc
t)])
        revenue_dict[product]=x
    original_revenue_dict_list.append(revenue_dict)
```

In [14]:
```
#Implementing Heuristic 1 (Node Greedy)
#One by one price of each item is updated

revenue_list=[]
for i in range(200):#no. of iterations=200 since we are changing pr
ice of upto 200 items
    print(i)
    max_revenue=-1
    flag=-1
    for product in range(500):#Changing price of 1 item at a time
        product_revenue=0
        for experiment in range(10):
            original_price=prices[product]
            new_price=original_price*(1+delta_u)
            temp_dict=original_revenue_dict_list[experiment]
            temp_revenue=new_price*f[experiment][product]*(1-delta_
u)
            for neighbours in list_of_adj_lists[experiment][product
]:
                if(product<neighbours):
                    temp_revenue+=new_price*g[experiment][(product,
neighbours)]*(1-weights[product]*delta_u)
                else:
                    temp_revenue+=new_price*g[experiment][(neighbou
rs,product)]*(1-weights[product]*delta_u)
            product_revenue+=sum(temp_dict.values())-temp_dict[prod
uct]+temp_revenue #Updated revenue is calculated with the new price
of the item
        product_revenue/=10 #Average across all the 10 networks
        if(product_revenue>max_revenue): #We need to change the pri
ce of the item for which we get the maximum revenue
            max_revenue=product_revenue
            flag=product
    #Updating the price and revenue because of the product selected
    prices[flag]*=(1+delta_u)
    for experiment in range(10):
        original_revenue_dict_list[experiment][flag]=prices[flag]*(
(f[experiment][flag]*(1-delta_u)))
        for neighbours in list_of_adj_lists[experiment][flag]:
            if(flag<neighbours):
                original_revenue_dict_list[experiment][flag]+=price
s[flag]*g[experiment][(flag,neighbours)]*(1-weights[flag]*delta_u)
            else:
                original_revenue_dict_list[experiment][flag]+=price
s[flag]*g[experiment][(neighbours,flag)]*(1-weights[flag]*delta_u)
    print(max_revenue)
    revenue_list.append(max_revenue)
```

```
0
5483059.084058963
1
5483116.603621125
2
```

```
5483174.306984493
3
5483232.19473639
4
5483290.267466025
5
5483348.525764487
6
5483406.970224751
7
5483465.601441687
8
5483524.420012071
9
5483583.426534583
10
5483642.621609812
11
5483702.005840272
12
5483761.5798304
13
5483821.344186565
14
5483881.299517071
15
5483941.446432172
16
5484001.785544066
17
5484062.317466908
18
5484123.042816818
19
5484183.962211884
20
5484245.076272167
21
5484306.385619713
22
5484367.890878553
23
5484429.592674713
24
5484491.4916362185
25
5484553.588393104
26
5484615.883577414
27
5484678.377823217
28
5484741.071766602
```

```
29
5484803.966045697
30
5484867.061300663
31
5484930.358173711
32
5484993.857309103
33
5485057.559353159
34
5485121.4649542635
35
5485185.574762875
36
5485249.889431528
37
5485314.409614847
38
5485379.135969542
39
5485444.069154426
40
5485509.209830415
41
5485574.558660538
42
5485640.116309941
43
5485705.883445898
44
5485771.860737815
45
5485838.048857232
46
5485904.448477843
47
5485971.060275489
48
5486037.884928173
49
5486104.923116064
50
5486172.175521502
51
5486239.642829011
52
5486307.325725303
53
5486375.22489928
54
5486443.341042047
55
```

```
5486511.674846921
56
5486580.22700943
57
5486648.998227326
58
5486717.989200591
59
5486787.2006314425
60
5486856.633224344
61
5486926.28768601
62
5486996.164725411
63
5487066.265053784
64
5487136.589384638
65
5487207.138433766
66
5487277.912919244
67
5487348.913561443
68
5487420.141083035
69
5487491.596209006
70
5487563.279666656
71
5487635.192185608
72
5487707.334497816
73
5487779.707337574
74
5487852.311441524
75
5487925.147548661
76
5487998.216400338
77
5488071.518740283
78
5488145.055314595
79
5488218.826871761
80
5488292.834162658
81
5488367.077940561
```

```
82
5488441.558961156
83
5488516.277982541
84
5488591.2357652355
85
5488666.433072193
86
5488741.8706688
87
5488817.549322894
88
5488893.469804761
89
5488969.632887151
90
5489046.039345285
91
5489122.689956861
92
5489199.5855020555
93
5489276.7267635465
94
5489354.114526507
95
5489431.749578625
96
5489509.632710099
97
5489587.764713658
98
5489666.1463845605
99
5489744.778520605
100
5489823.661922144
101
5489902.797392087
102
5489982.185735904
103
5490061.8277616445
104
5490141.724279936
105
5490221.876104
106
5490302.284049654
107
5490382.948935322
108
```

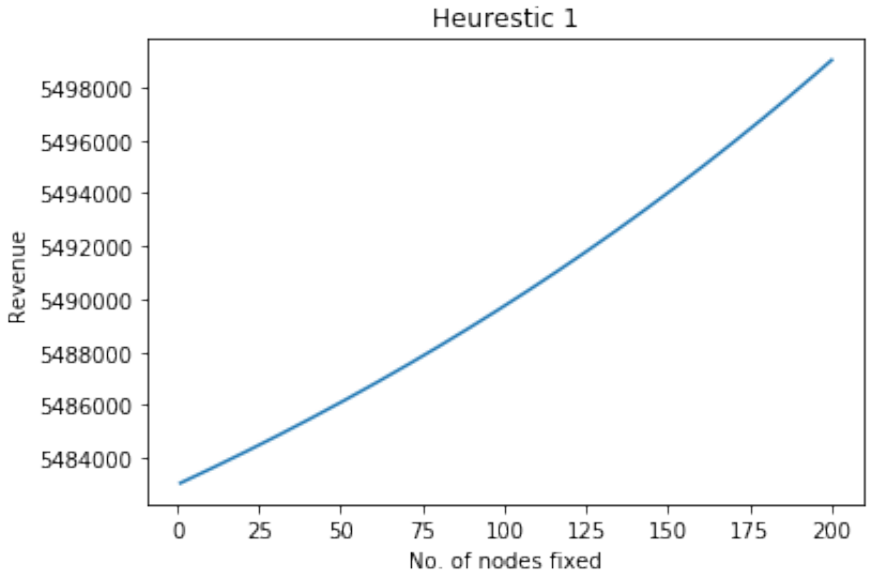
```
5490463.871582047
109
5490545.052813491
110
5490626.49345595
111
5490708.194338361
112
5490790.156292307
113
5490872.380152035
114
5490954.866754451
115
5491037.616939136
116
5491120.631548357
117
5491203.91142707
118
5491287.457422933
119
5491371.270386312
120
5491455.351170287
121
5491539.70063067
122
5491624.319626002
123
5491709.209017572
124
5491794.369669416
125
5491879.802448339
126
5491965.508223907
127
5492051.487868471
128
5492137.742257165
129
5492224.272267924
130
5492311.078781484
131
5492398.162681399
132
5492485.524854043
133
5492573.166188625
134
5492661.087577196
```

```
135
5492749.289914654
136
5492837.7740987595
137
5492926.5410301415
138
5493015.5916123055
139
5493104.926751645
140
5493194.547357449
141
5493284.454341916
142
5493374.648620153
143
5493465.131110196
144
5493555.902733012
145
5493646.964412515
146
5493738.317075564
147
5493829.9616519855
148
5493921.899074577
149
5494014.130279114
150
5494106.656204362
151
5494199.477792093
152
5494292.59598708
153
5494386.011737117
154
5494479.72599303
155
5494573.7397086825
156
5494668.053840982
157
5494762.669349897
158
5494857.587198464
159
5494952.808352796
160
5495048.333782094
161
```

```
5495144.164458652
162
5495240.3013578765
163
5495336.745458287
164
5495433.497741533
165
5495530.559192397
166
5495627.930798814
167
5495725.613551867
168
5495823.608445816
169
5495921.916478089
170
5496020.538649309
171
5496119.475963293
172
5496218.729427064
173
5496318.300050864
174
5496418.188848167
175
5496518.396835679
176
5496618.92503336
177
5496719.774464427
178
5496820.946155366
179
5496922.441135945
180
5497024.260439221
181
5497126.4051015545
182
5497228.876162614
183
5497331.674665393
184
5497434.801656216
185
5497538.258184753
186
5497642.045304028
187
5497746.1640704265
```

```
188
5497850.615543715
189
5497955.400787041
190
5498060.520866952
191
5498165.976853406
192
5498271.769819775
193
5498377.900842863
194
5498484.371002916
195
5498591.181383633
196
5498698.333072172
197
5498805.827159166
198
5498913.664738737
199
5499021.846908498
```

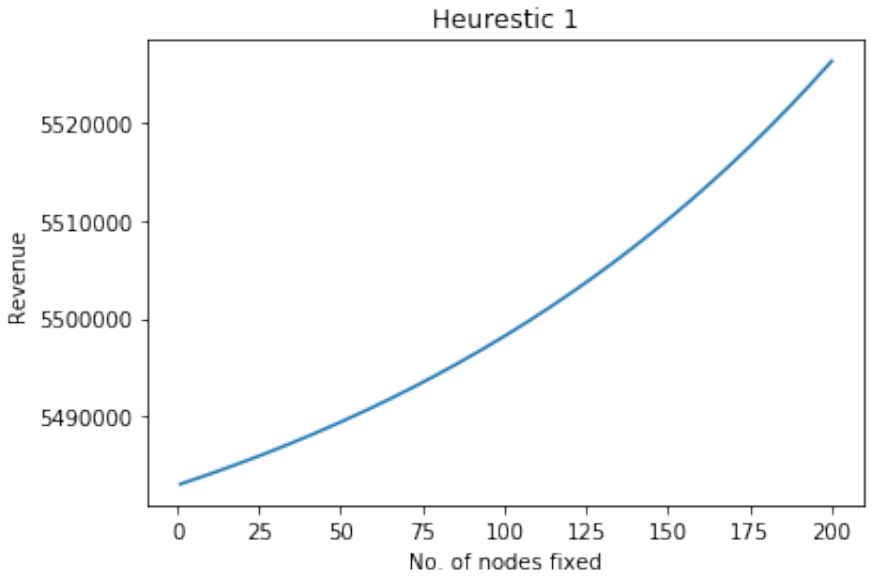
In [15]:
```
#Plotting the line graph
x_axis=[i+1 for i in range(200)]
plt.plot(x_axis,revenue_list)
plt.xlabel("No. of nodes fixed")
plt.ylabel("Revenue")
plt.title("Heurestic 1")
plt.show()
```



In [16]:
```
#Storing the results in a pickle file
file_result = 'result'
outfile7 = open(file_result,'wb')
pickle.dump(revenue_list,outfile7)
outfile7.close()
```

In [18]:
```
# Loading results plotted in final plot
file_results = 'result_final'
file = open(file_results , 'rb')
y_axis = pickle.load(file)
file.close()
```

In [19]:
```
#Plotting the line graph used in final plot
x_axis=[i+1 for i in range(200)]
plt.plot(x_axis,y_axis)
plt.xlabel("No. of nodes fixed")
plt.ylabel("Revenue")
plt.title("Heurestic 1")
plt.show()
```



In [ ]:
```

```