In[22]:=

```
Needs["Combinatorica`"]
Needs["GraphUtilities`"]
```

In[24]:=

```
nodeEventProb = 0.7; (* Node-event probability *)
edgeEventProb = 0.95; (* Edge-event probability *)
duplEventProb = 0.75; (* Duplication-event probability (duplication or fusion) *)

(* Conditional node-addition probability (given a node event). Conditional probability
  of node removal is 1-nodeAddProb *)
nodeAddProb = 1;

(* Conditional edge-addition probability (given an edge event). 1-
 edgeAddProb is the conditional probability that an edge is removed *)
edgeAddProb = 0.9;

(* Conditional node-duplication probability (given a duplication/fusion event). Conditional node-
 merging probability is 1-duplProb *)
duplProb = 1;
```

In[30]:=

```
(* Execute the function decideNodeEvent on the specified graph,
use the resulting graph as input to decideEdgeEvent,
use the resulting graph as input to decideDuplEvent *)
runEvents[graph_] := Composition[decideDuplEvent, decideEdgeEvent, decideNodeEvent][graph]

(* Call runNodeEvent and return the results or return the graph unchanged. Either way,
it feeds into decideEdgeEvent, and so on  *)
decideNodeEvent[graph_] :=
 RandomChoice[{nodeEventProb, (1 - nodeEventProb)} → {runNodeEvent[graph], graph}]
decideEdgeEvent[graph_] :=
 RandomChoice[{edgeEventProb, (1 - edgeEventProb)} → {runEdgeEvent[graph], graph}]
decideDuplEvent[graph_] :=
 RandomChoice[{duplEventProb, (1 - duplEventProb)} → {runDuplEvent[graph], graph}]

(* Given that a node event will occur this iteration, choose between adding or removing  *)
runNodeEvent[graph_] :=
 RandomChoice[{nodeAddProb, (1 - nodeAddProb)} → {addNode[graph], removeNode[graph]}]
runEdgeEvent[graph_] :=
 RandomChoice[{edgeAddProb, (1 - edgeAddProb)} → {addEdge[graph], removeEdge[graph]}]
runDuplEvent[graph_] :=
 RandomChoice[{duplProb, (1 - duplProb)} → {duplicateNode[graph], mergeNodes[graph]}]
```

In[37]:=

```
addNode[graph_] := AddVertex[graph]
removeNode[graph_] := If[V[graph] > 1,
   (*then*)DeleteVertex[graph, RandomChoice[VertexList[graph]]],
   (*else*)graph
  ]

(* Adds an edge between two existing vertices
 (RandomSample will choose two different items from the range of 1 through #nodes) *)
addEdge[graph_] := If[V[graph] > 1,
   (*then*)MakeSimple[AddEdges[graph, {{RandomSample[Range[1, nodeCount[graph]], 2]}}]],
   (*else*)graph
  ]
removeEdge[graph_] := If[M[graph] > 1,
   (*then*)DeleteEdge[graph, RandomChoice[Edges[graph]]],
   (*else*)graph
  ]

duplicateNode[graph_] := If[V[graph] > 5,
   (*then*)Module[{chosenNode = RandomChoice[VertexList[graph]]},
    neighbors = Neighborhood[graph, chosenNode, 1];
    (* Neighborhood[g,v,k] returns the subset of vertices in g that are at a distance
     of k or less from vertex v *)
    neighbors = DeleteCases[neighbors, chosenNode];
    (* Don't include the chosen node itself in the list of its neighbors *)
    graph = AddVertex[graph];
    newNode = nodeCount[graph];
    attachNode[neighbor_] := {{newNode, neighbor}};
    newEdges = Map[attachNode, neighbors]; (* Build a list of all the new edges to be added *)
    graph = MakeSimple[AddEdges[graph, newEdges]]
   ],
   (*else*)graph
  ]
mergeNodes[graph_] := If[V[graph] > 5,
   (*then*)Contract[graph, RandomSample[Range[1, nodeCount[graph]], 2]],
   (*else*)graph
  ]
```

In[43]:=

```
SeedRandom[123];
maxNodes = 250;
myGraph = MakeGraph[{1}, False, Type→Undirected]; (* One starting vertex,
Edge rule is always False at outset, undirected graph *)
nodeCount[graph_] := V[graph]
(* Delayed evaluation: V[] gives the number of vertices in the graph *)

iterationCount = 0;
While[nodeCount[myGraph] < maxNodes && iterationCount < 5000,
 myGraph = runEvents[myGraph];
 iterationCount = iterationCount + 1;
]
Print["myGraph: ", myGraph]
Print["Iterations to generate myGraph: ", iterationCount]

mySeq = DegreeSequence[myGraph]; (* Gives the sorted degree sequence of the graph *)
Print["myGraph degree sequence: ", mySeq]

myGraphNodes = V[myGraph];
myGraphEdges = M[myGraph];
(* Calculate the fraction of how many of the possible distinct edges in myGraph exist *)
randEdgeProb = (2 * myGraphEdges) / (myGraphNodes * (myGraphNodes - 1));
Print["Edge probability for random graph: ", randEdgeProb, "  ~  ", randEdgeProb // N]

(* RandomGraph[n,p] Constructs a random labeled graph on n vertices with an edge
 probability of p. *)
randomGraph = RandomGraph[maxNodes, randEdgeProb];
Print["randomGraph: ", randomGraph]
randomSeq = DegreeSequence[randomGraph];
Print["randomGraph degree sequence: ", randomSeq]
```

myGraph: - Graph: < 612, 251, Undirected > -

Iterations to generate myGraph: 149

myGraph degree sequence:

{20, 19, 19, 19, 19, 19, 17, 17, 15, 15, 15, 15, 15, 15, 14, 14, 14, 14, 14, 13, 13, 13, 13, 13, 13, 12,
12, 12, 12, 12, 12, 12, 12, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 10, 10, 10, 10, 10,
10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 9, 9, 9, 9, 9, 9, 8, 8, 8, 8, 8, 8, 8, 7, 7, 7, 7, 7, 7, 7,
7, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

Edge probability for random graph: $\frac{612}{31375}$ ~ 0.019506

randomGraph: - Graph: < 580, 250, Undirected > -

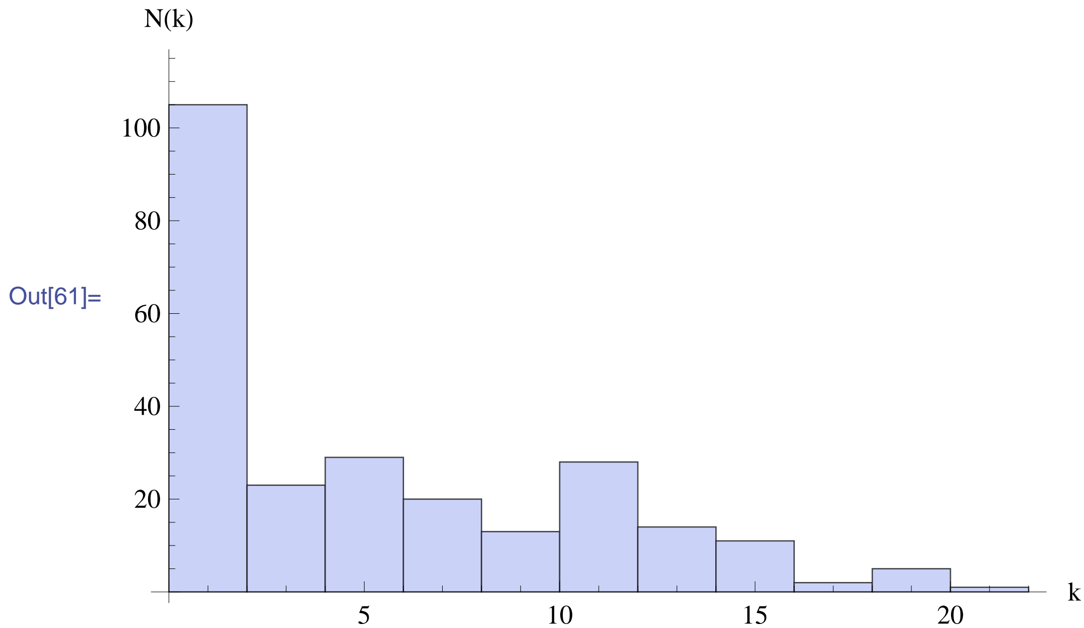
randomGraph degree sequence:

{12, 10, 10, 10, 10, 9, 9, 9, 9, 9, 9, 8, 8, 8, 8, 8, 8, 8, 8, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}

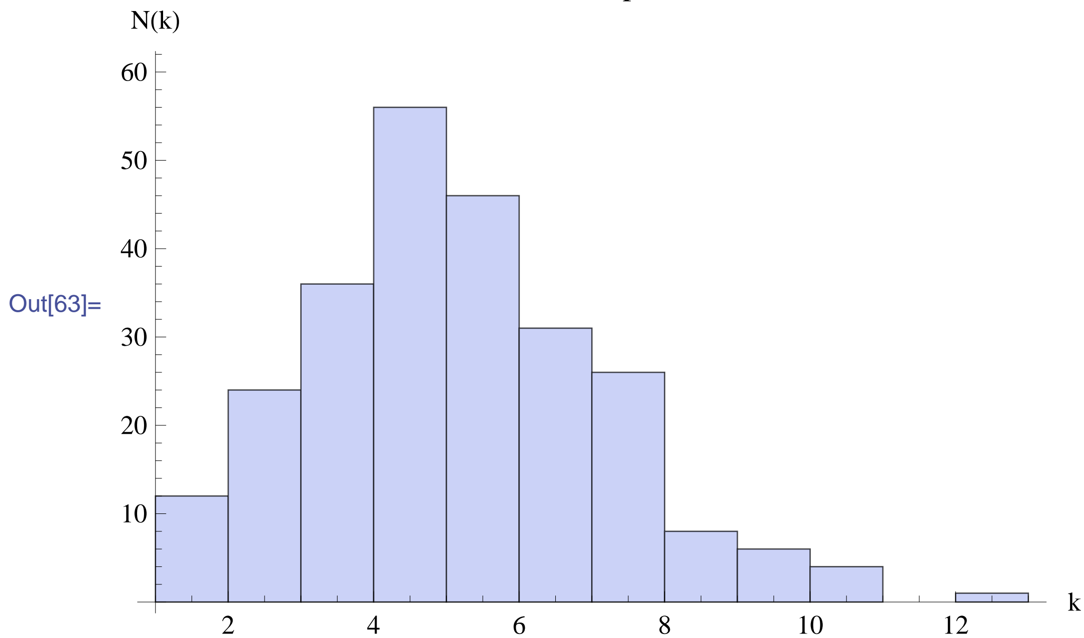
In[61]:=
```
Histogram[mySeq, Automatic, "Count", PlotRange→All, ImageSize→Medium, AxesLabel →{"k", "N(k)"},
 PlotLabel →"Growth process network, maxNodes = 250"]
Export["myGraphHistogram.png", %];

Histogram[randomSeq, Automatic, "Count", PlotRange→All, ImageSize→Medium,
 AxesLabel →{"k", "N(k)"}, PlotLabel →"Mathematica RandomGraph, maxNodes = 250"]
Export["randomGraphHistogram.png", %];
```

Out[61]=



Out[63]=

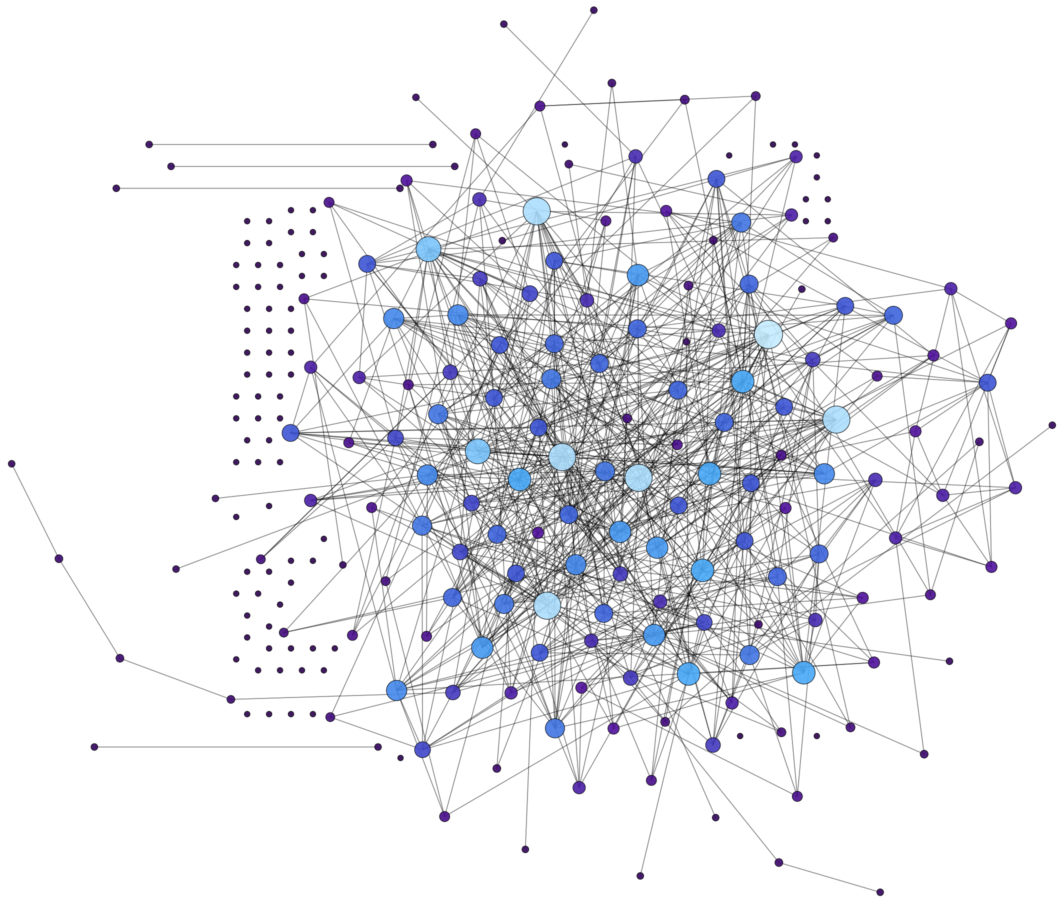


In[65]:=

```
GraphPlot[myGraph,
 Method -> {"SpringElectricalEmbedding", "RepulsiveForcePower" → -8, "InferentialDistance" → .5},
 EdgeLabeling → Automatic,
 VertexLabeling → Tooltip,
 PackingMethod → {"ClosestPackingCenter", "Padding" → 1},
 VertexRenderingFunction →
  ({ColorData["DeepSeaColors"][(Degrees[myGraph][[#2]] / mySeq[[1]])], Opacity[0.9],
     EdgeForm[{Thin, Black}], Disk[#1, .01 + .04 * (Degrees[myGraph][[#2]] / mySeq[[1]])], Black} &),
 EdgeRenderingFunction → ({Opacity[0.3], AbsoluteThickness[0.3], Line[#1]} &),
 ImageSize → Full,
 ImageMargins → 5]
Export["myGraph.png", %];

GraphPlot[randomGraph,
 Method -> {"SpringElectricalEmbedding", "RepulsiveForcePower" → -8, "InferentialDistance" → .5},
 EdgeLabeling → Automatic,
 VertexLabeling → Tooltip,
 PackingMethod → {"ClosestPackingCenter", "Padding" → 1},
 VertexRenderingFunction →
  ({ColorData["DeepSeaColors"][(Degrees[randomGraph][[#2]] / randomSeq[[1]])], Opacity[0.9],
     EdgeForm[{Thin, Black}], Disk[#1, .04 * (Degrees[randomGraph][[#2]] / randomSeq[[1]])],
     Black} &),
 EdgeRenderingFunction → ({Opacity[0.3], AbsoluteThickness[0.3], Line[#1]} &),
 ImageSize → Full,
 ImageMargins → 5]
Export["randomGraph.png", %];
```

Out[65]=



Out[67]=