Group Details:

Tejasri Dontham - 81715873

Yamini Palnati - 98601457

ASSIGNMENT 1

In this project, we are implementing the functions in c++ language.

Data structure for storing Graph:

Graph.h file

```
Used adjacency list for implementation because it takes less space and comparatively with
adjacency matrix there will be few less comparisons.

class Graph
{
    int V; //-> no of nodes(customers)
    vector <int>* adj; //used array of vectors

};
```

Graph.cpp

It contains the basic functions of graph like adding edges in graph, printing the graph.

Graph_Operations.cpp

It contains the graph operations.

```
void connectedComponents(ofstream& outfile); - Used DFS traversal and it passes through
all the elements that are connected.

void one_cycle(ofstream& outfile); - While doing dfs if the node is visited previously (
or for the second time) we say it as found cycle and store this node as cyclestart and we
will go back till to printcycle.

vector<int> DijkstraSP(int& start, ofstream& outfile);- used dijkshtra algorithm with
priority queues and calculated shortest path.
```

Graph_simulator.h && Graph_simulator.cpp

```
void generateConnectedGraph(int n, ofstream& outfile);
void generateEmptyGraph(int n, ofstream& outfile);
void generateHeap(int n, ofstream& outfile);
void generateTruncatedHeap(int n, int m, ofstream& outfile);
void generateEquivalenceModKGraph(int n, int k, ofstream& outfile);
void generateNcycleGraph(int n, ofstream& outfile);
```

This file contains the 6 functions to generate the simulator graphs required to test the graph operations test cases.

Used mathematical conditions to generate the graph.

Simulated test.cpp

It has the main method to call the simulated graph functions:

For each type of graph, we called all the 3 operations, and the output is stored in **SimulatesTestResults.txt** included in zip file

CPU and peak memory usage report is saved as **SimulatedTestReports.DIAGSSESSION** in zip.

Also pasting output here:

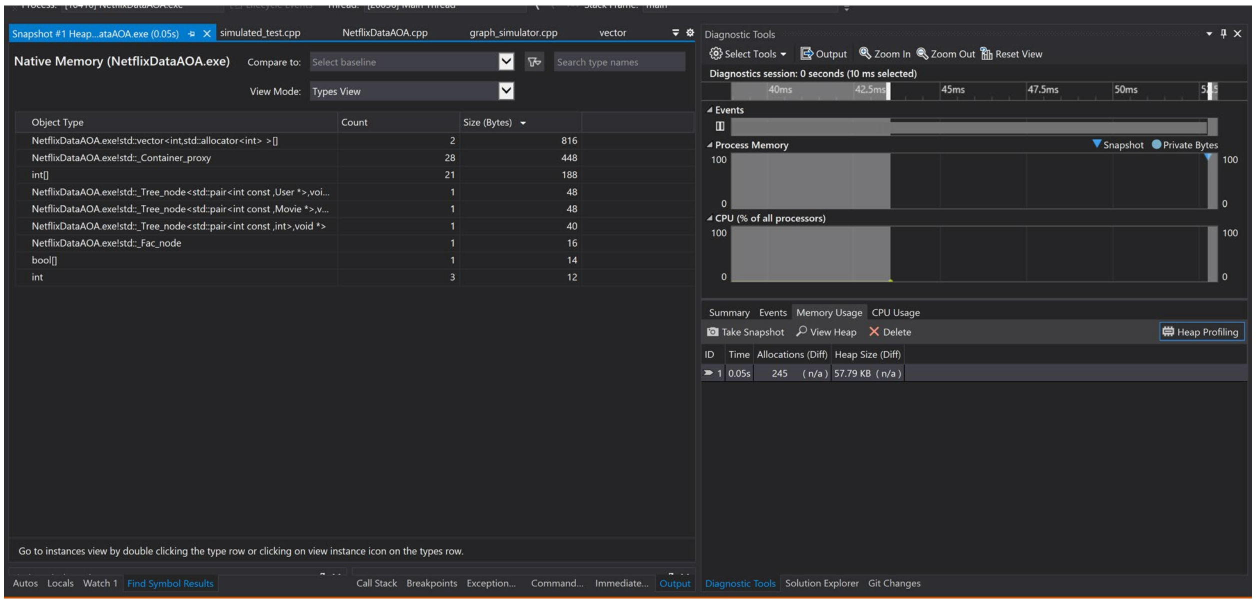
```
//Tested creating a graph with pre-compiled values. For n=14

 Adjacency list of vertex 0

 head

 Adjacency list of vertex 1

 head -> 2

 Adjacency list of vertex 2

 head -> 1-> 3

 Adjacency list of vertex 3

 head -> 2-> 4-> 5

 Adjacency list of vertex 4

 head -> 3-> 6-> 7

 Adjacency list of vertex 5

 head -> 6-> 3-> 9

 Adjacency list of vertex 6

 head -> 4-> 5-> 10

 Adjacency list of vertex 7

 head -> 4-> 8

 Adjacency list of vertex 8

 head -> 7

 Adjacency list of vertex 9

 head -> 5

 Adjacency list of vertex 10
 head -> 6-> 11

 Adjacency list of vertex 11

 head -> 10-> 12-> 13

 Adjacency list of vertex 12

 head -> 11-> 13

 Adjacency list of vertex 13

 head -> 11-> 12

 --------------------------------------------

 Following are connected components

0

1 2 3 4 6 5 9 10 11 12 13 7 8

 --------------------------------------------

 finding cycle

Cycle found: 3 5 6 4 3

 --------------shortest path-----------------------------


Getting the shortest path from 1 to all other nodes.

Printing the shortest paths for node 1.

There is no path from 1 to node 0

The distance from node 1 to node 1 is: 0 and the path is 1

The distance from node 1 to node 2 is: 1 and the path is 1 - 2

The distance from node 1 to node 3 is: 2 and the path is 1 - 2 - 3

The distance from node 1 to node 4 is: 3 and the path is 1 - 2 - 3 - 4

The distance from node 1 to node 5 is: 3 and the path is 1 - 2 - 3 - 5

The distance from node 1 to node 6 is: 4 and the path is 1 - 2 - 3 - 4 - 6

The distance from node 1 to node 7 is: 4 and the path is 1 - 2 - 3 - 4 - 7

The distance from node 1 to node 8 is: 5 and the path is 1 - 2 - 3 - 4 - 7 - 8

The distance from node 1 to node 9 is: 4 and the path is 1 - 2 - 3 - 5 - 9

The distance from node 1 to node 10 is: 5 and the path is 1 - 2 - 3 - 4 - 6 - 10

The distance from node 1 to node 11 is: 6 and the path is 1 - 2 - 3 - 4 - 6 - 10 - 11

The distance from node 1 to node 12 is: 7 and the path is 1 - 2 - 3 - 4 - 6 - 10 - 11 -
12
```

```
The distance from node 1 to node 13 is: 7 and the path is 1 - 2 - 3 - 4 - 6 - 10 - 11 -
13
```

```
 --------------testing simulator functions------------------------------
```

//An n-cycle: There is one connected component, every shortest path has length at most n/2, and there is a unique cycle of length n.

```
 --------------testing with n cycle------------------------------

 Adjacency list of vertex 0
 head -> 1-> 10
 Adjacency list of vertex 1
 head -> 0-> 2
 Adjacency list of vertex 2
 head -> 1-> 3
 Adjacency list of vertex 3
 head -> 2-> 4
 Adjacency list of vertex 4
 head -> 3-> 5
 Adjacency list of vertex 5
 head -> 4-> 6
 Adjacency list of vertex 6
 head -> 5-> 7
 Adjacency list of vertex 7
 head -> 6-> 8
 Adjacency list of vertex 8
 head -> 7-> 9
 Adjacency list of vertex 9
 head -> 8-> 10
 Adjacency list of vertex 10
 head -> 0-> 9
 --------------------------------------------


 Following are connected components
0 1 2 3 4 5 6 7 8 9 10

 --------------------------------------------


 finding cycle
Cycle found: 0 10 9 8 7 6 5 4 3 2 1 0

 --------------shortest path-----------------------------

Getting the shortest path from 1 to all other nodes.

Printing the shortest paths for node 1.
The distance from node 1 to node 0 is: 1 and the path is 1 - 0
The distance from node 1 to node 1 is: 0 and the path is 1
The distance from node 1 to node 2 is: 1 and the path is 1 - 2
The distance from node 1 to node 3 is: 2 and the path is 1 - 2 - 3
The distance from node 1 to node 4 is: 3 and the path is 1 - 2 - 3 - 4
The distance from node 1 to node 5 is: 4 and the path is 1 - 2 - 3 - 4 - 5
The distance from node 1 to node 6 is: 5 and the path is 1 - 2 - 3 - 4 - 5 - 6
The distance from node 1 to node 7 is: 5 and the path is 1 - 0 - 10 - 9 - 8 - 7
The distance from node 1 to node 8 is: 4 and the path is 1 - 0 - 10 - 9 - 8
The distance from node 1 to node 9 is: 3 and the path is 1 - 0 - 10 - 9
The distance from node 1 to node 10 is: 2 and the path is 1 - 0 - 10
```



Native Memory (NetflixDataAOA.exe)

Compare to: Select baseline

View Mode: Types View

Object Type	Count	Size (Bytes)
NetflixDataAOA.exe!std::vector<int,std::allocator<int> >[]	2	816
NetflixDataAOA.exe!std::_Container_proxy	28	448
int[]	21	188
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,User *>,voi...	1	48
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,Movie *>,v...	1	48
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,int>,void *>	1	40
NetflixDataAOA.exe!std::_Fac_node	1	16
bool[]	1	14
int	3	12

Go to instances view by double clicking the type row or clicking on view instance icon on the types row.

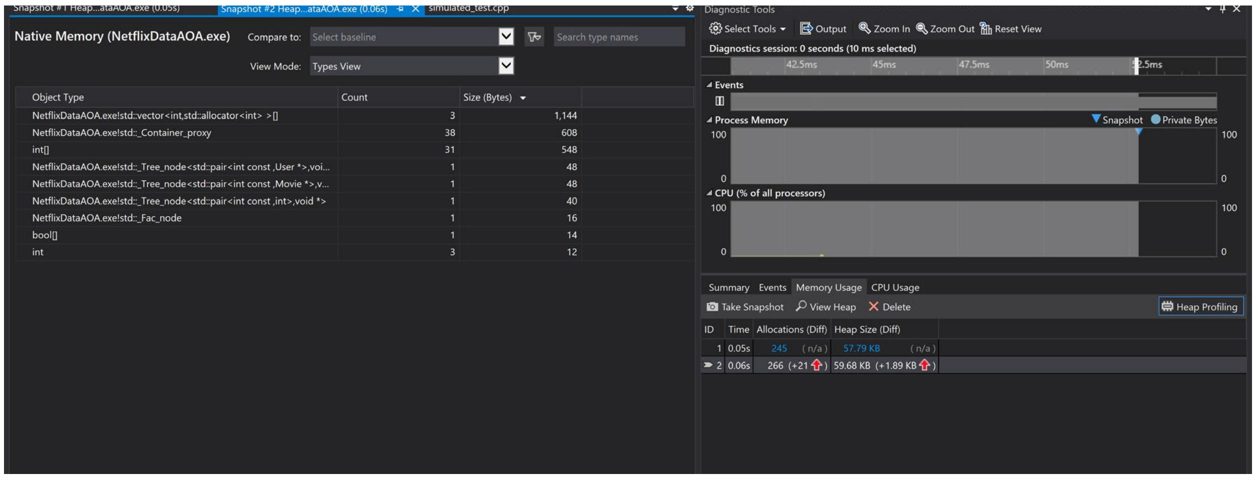
ID	Time	Allocations (Diff)	Heap Size (Diff)
1	0.05s	245 (n/a)	57.79 KB (n/a)

```
--------------testing with connected graph-----------------------------
```

// A complete graph on n vertices: There is one connected component, every shortest path has unit length, and there are many cycles.

```
Adjacency list of vertex 0

head -> 1-> 2-> 3-> 4-> 5-> 6-> 7-> 8-> 9

Adjacency list of vertex 1

head -> 0-> 2-> 3-> 4-> 5-> 6-> 7-> 8-> 9

Adjacency list of vertex 2

head -> 0-> 1-> 3-> 4-> 5-> 6-> 7-> 8-> 9

Adjacency list of vertex 3

head -> 0-> 1-> 2-> 4-> 5-> 6-> 7-> 8-> 9

Adjacency list of vertex 4

head -> 0-> 1-> 2-> 3-> 5-> 6-> 7-> 8-> 9

Adjacency list of vertex 5

head -> 0-> 1-> 2-> 3-> 4-> 6-> 7-> 8-> 9

Adjacency list of vertex 6

head -> 0-> 1-> 2-> 3-> 4-> 5-> 7-> 8-> 9

Adjacency list of vertex 7

head -> 0-> 1-> 2-> 3-> 4-> 5-> 6-> 8-> 9

Adjacency list of vertex 8
```

```
 head -> 0-> 1-> 2-> 3-> 4-> 5-> 6-> 7-> 9
 Adjacency list of vertex 9
 head -> 0-> 1-> 2-> 3-> 4-> 5-> 6-> 7-> 8
 --------------------------------------------

 Following are connected components
0 1 2 3 4 5 6 7 8 9

 --------------------------------------------

 finding cycle
Cycle found: 0 2 1 0

 --------------shortest path-----------------------------

Getting the shortest path from 1 to all other nodes.

Printing the shortest paths for node 1.
The distance from node 1 to node 0 is: 1 and the path is 1 - 0
The distance from node 1 to node 1 is: 0 and the path is 1
The distance from node 1 to node 2 is: 1 and the path is 1 - 2
The distance from node 1 to node 3 is: 1 and the path is 1 - 3
The distance from node 1 to node 4 is: 1 and the path is 1 - 4
The distance from node 1 to node 5 is: 1 and the path is 1 - 5
The distance from node 1 to node 6 is: 1 and the path is 1 - 6
The distance from node 1 to node 7 is: 1 and the path is 1 - 7
The distance from node 1 to node 8 is: 1 and the path is 1 - 8
The distance from node 1 to node 9 is: 1 and the path is 1 - 9
```



Native Memory (NetflixDataAOA.exe) Compare to: Select baseline

View Mode: Types View

Object Type	Count	Size (Bytes)
NetflixDataAOA.exe!std::vector<int,std::allocator<int> >[]	3	1,144
NetflixDataAOA.exe!std::_Container_proxy	38	608
int[]	31	548
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,User *>,voi...	1	48
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,Movie *>,v...	1	48
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,int>,void *>	1	40
NetflixDataAOA.exe!std::_Fac_node	1	16
bool[]	1	14
int	3	12

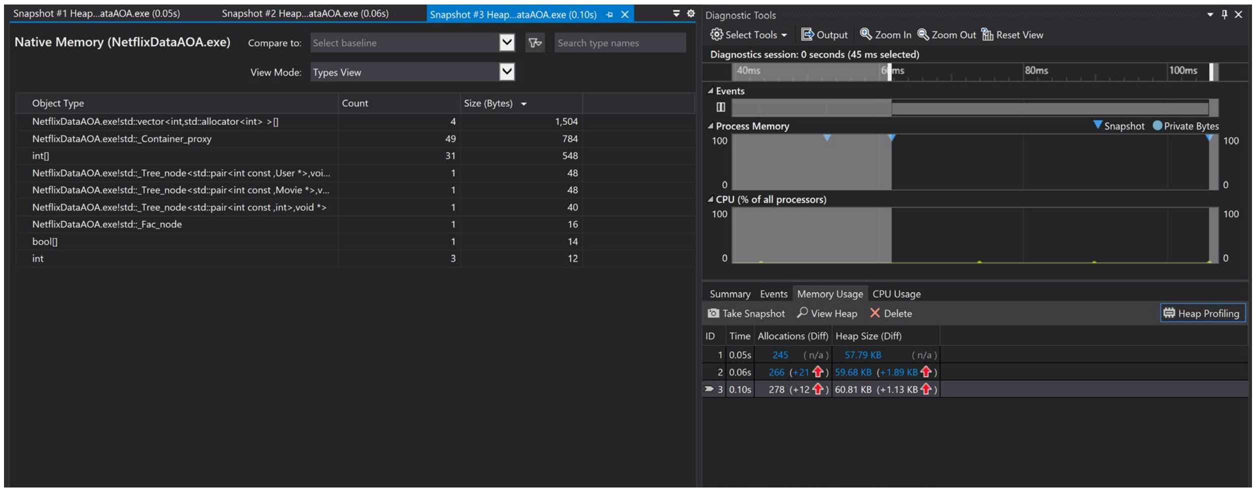
Diagnostics session: 0 seconds (10 ms selected)

ID	Time	Allocations (Diff)	Heap Size (Diff)
1	0.05s	245 (n/a)	57.79 KB (n/a)
2	0.06s	266 (+21 ↑)	59.68 KB (+1.89 KB ↑)

// . An empty graph on n vertices: The vertices are integers from 0 through n – 1. There are no edges. There are n connected components, no paths, and no cycles

```
--------------testing with empty graph-----------------------------

Adjacency list of vertex 0

head

Adjacency list of vertex 1

head

Adjacency list of vertex 2

head

Adjacency list of vertex 3

head

Adjacency list of vertex 4

head

Adjacency list of vertex 5

head

Adjacency list of vertex 6

head

Adjacency list of vertex 7

head

Adjacency list of vertex 8

head

Adjacency list of vertex 9

head
```

```
 Adjacency list of vertex 10

 head

 -------------------------------------------


 Following are connected components

0

1

2

3

4

5

6

7

8

9

10


 -------------------------------------------


 finding cycle

Acyclic


 --------------shortest path-----------------------------


Getting the shortest path from 1 to all other nodes.


Printing the shortest paths for node 1.

There is no path from 1 to node 0

The distance from node 1 to node 1 is: 0 and the path is 1

There is no path from 1 to node 2

There is no path from 1 to node 3

There is no path from 1 to node 4
There is no path from 1 to node 5
There is no path from 1 to node 6
There is no path from 1 to node 7
There is no path from 1 to node 8
There is no path from 1 to node 9
There is no path from 1 to node 10
```



Native Memory (NetflixDataAOA.exe)

Object Type	Count	Size (Bytes)
NetflixDataAOA.exe!std::vector<int,std::allocator<int> >[]	4	1,504
NetflixDataAOA.exe!std::_Container_proxy	49	784
int[]	31	548
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,User *>,voi...	1	48
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,Movie *>,v...	1	48
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,int>,void *>	1	40
NetflixDataAOA.exe!std::_Fac_node	1	16
bool[]	1	14
int	3	12

Diagnostics session: 0 seconds (45 ms selected)

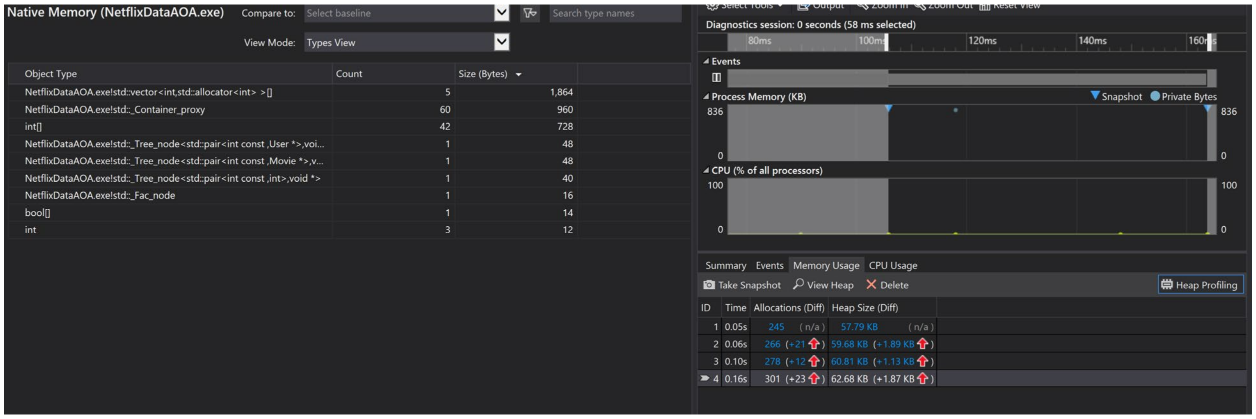
Process Memory

CPU (% of all processors)

ID	Time	Allocations (Diff)	Heap Size (Diff)
1	0.05s	245 (n/a)	57.79 KB (n/a)
2	0.06s	266 (+21 ⬆)	59.68 KB (+1.89 KB ⬆)
3	0.10s	278 (+12 ⬆)	60.81 KB (+1.13 KB ⬆)

// A heap: The vertices are integers from 0 through n – 1. The neighbors of a vertex v are (v – 1)/2, 2v + 1, and 2v + 2, provided those numbers are in the range for vertices. There is one connected component, the paths are short, and there are no cycles.

```
 --------------testing with heap ----------------------------


 Adjacency list of vertex 0
 head -> 0-> 0-> 10-> 1-> 2
 Adjacency list of vertex 1
 head -> 0-> 10-> 3-> 4
 Adjacency list of vertex 2
 head -> 0-> 10-> 5-> 6
 Adjacency list of vertex 3
 head -> 1-> 10-> 7-> 8
 Adjacency list of vertex 4
 head -> 1-> 10-> 9-> 10
 Adjacency list of vertex 5
 head -> 2-> 10
 Adjacency list of vertex 6
 head -> 2-> 10
 Adjacency list of vertex 7
 head -> 3-> 10
 Adjacency list of vertex 8
 head -> 3-> 10
 Adjacency list of vertex 9
 head -> 4-> 10
 Adjacency list of vertex 10
 head -> 0-> 1-> 2-> 3-> 4-> 5-> 6-> 7-> 8-> 9-> 4-> 10-> 10
 --------------------------------------------

 Following are connected components
0 10 1 3 7 8 4 9 2 5 6

 --------------------------------------------

 finding cycle
Acyclic

 --------------shortest path------------------------------

Getting the shortest path from 1 to all other nodes.

Printing the shortest paths for node 1.
The distance from node 1 to node 0 is: 1 and the path is 1 - 0
The distance from node 1 to node 1 is: 0 and the path is 1
The distance from node 1 to node 2 is: 2 and the path is 1 - 0 - 2
The distance from node 1 to node 3 is: 1 and the path is 1 - 3
The distance from node 1 to node 4 is: 1 and the path is 1 - 4
The distance from node 1 to node 5 is: 2 and the path is 1 - 10 - 5

The distance from node 1 to node 6 is: 2 and the path is 1 - 10 - 6

The distance from node 1 to node 7 is: 2 and the path is 1 - 3 - 7

The distance from node 1 to node 8 is: 2 and the path is 1 - 3 - 8

The distance from node 1 to node 9 is: 2 and the path is 1 - 4 - 9

The distance from node 1 to node 10 is: 1 and the path is 1 - 10
```



Native Memory (NetflixDataAOA.exe) Compare to: Select baseline

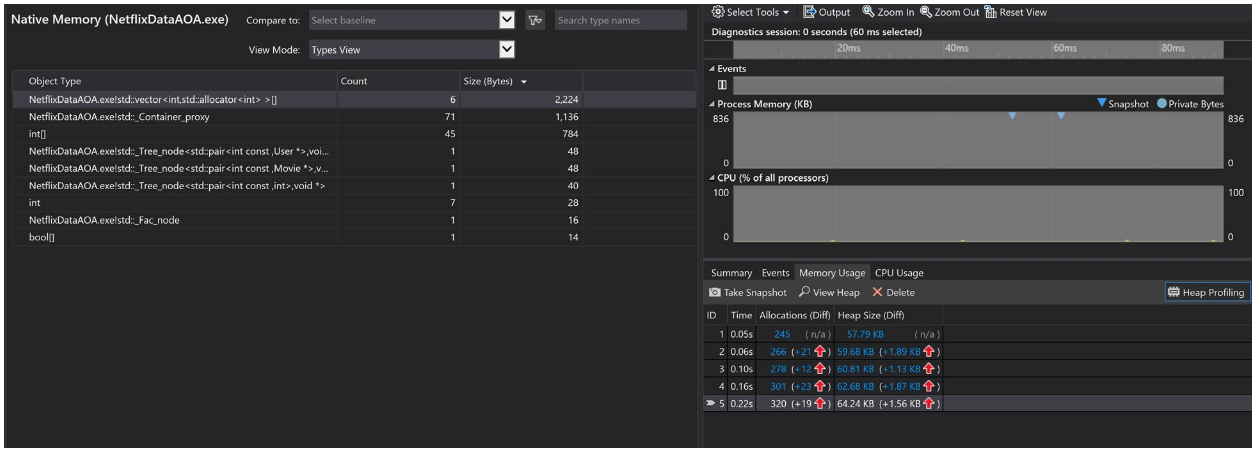
View Mode: Types View

Object Type	Count	Size (Bytes)
NetflixDataAOA.exe!std::vector<int,std::allocator<int> >[]	5	1,864
NetflixDataAOA.exe!std::_Container_proxy	60	960
int[]	42	728
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,User *>,voi...	1	48
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,Movie *>,v...	1	48
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,int>,void *>	1	40
NetflixDataAOA.exe!std::_Fac_node	1	16
bool[]	1	14
int	3	12

ID	Time	Allocations (Diff)	Heap Size (Diff)
1	0.05s	245 (n/a)	57.79 KB (n/a)
2	0.06s	266 (+21 ⇧)	59.68 KB (+1.89 KB ⇧)
3	0.10s	278 (+12 ⇧)	60.81 KB (+1.13 KB ⇧)
4	0.16s	301 (+23 ⇧)	62.68 KB (+1.87 KB ⇧)

// A truncated heap: The vertices are integers from m through n – 1. The edge relationship is the same as for the heap. There are n – 1 – 2m edges, m + 1 connected components, and no cycles. The paths, when they exist, are short

```
 --------------testing with truncated heap -----------------------------


 Adjacency list of vertex 0

 head

 Adjacency list of vertex 1

 head

 Adjacency list of vertex 2

 head

 Adjacency list of vertex 3

 head

 Adjacency list of vertex 4

 head -> 10-> 9-> 10

 Adjacency list of vertex 5

 head -> 10

 Adjacency list of vertex 6

 head -> 10
 Adjacency list of vertex 7

 head -> 10

 Adjacency list of vertex 8

 head -> 10

 Adjacency list of vertex 9

 head -> 4-> 10

 Adjacency list of vertex 10

 head -> 4-> 5-> 6-> 7-> 8-> 9-> 4-> 10-> 10

 --------------------------------------------


 Following are connected components

0

1

2

3

4 10 5 6 7 8 9


 --------------------------------------------


 finding cycle

Cycle found: 4 9 10 4


 --------------shortest path-----------------------------


Getting the shortest path from 6 to all other nodes.


Printing the shortest paths for node 6.

There is no path from 6 to node 0

There is no path from 6 to node 1

There is no path from 6 to node 2

There is no path from 6 to node 3

The distance from node 6 to node 4 is: 2 and the path is 6 - 10 - 4
The distance from node 6 to node 5 is: 2 and the path is 6 - 10 - 5

The distance from node 6 to node 6 is: 0 and the path is 6

The distance from node 6 to node 7 is: 2 and the path is 6 - 10 - 7

The distance from node 6 to node 8 is: 2 and the path is 6 - 10 - 8

The distance from node 6 to node 9 is: 2 and the path is 6 - 10 - 9

The distance from node 6 to node 10 is: 1 and the path is 6 - 10
```



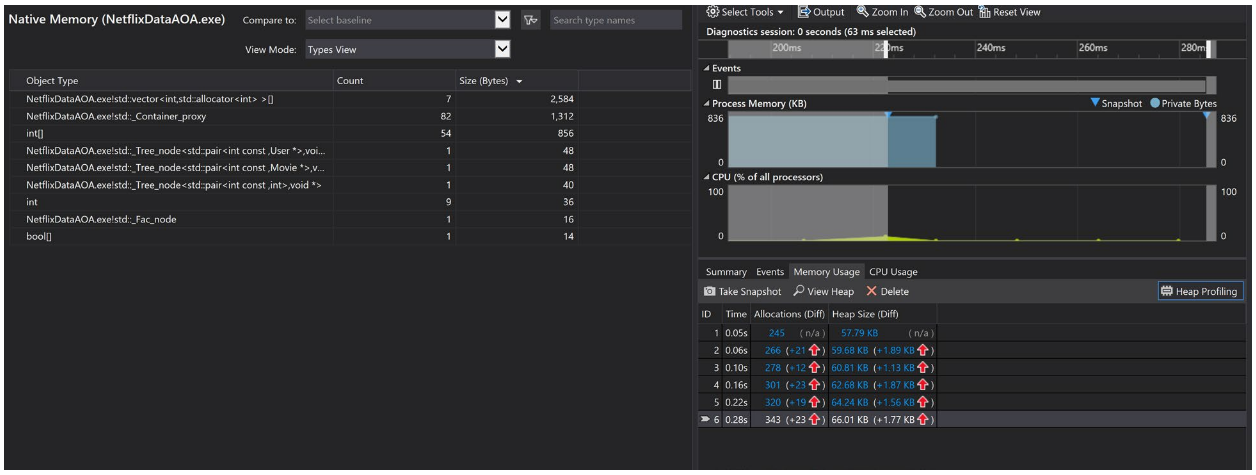
Object Type	Count	Size (Bytes)
NetflixDataAOA.exe!std::vector<int,std::allocator<int> >[]	6	2,224
NetflixDataAOA.exe!std::_Container_proxy	71	1,136
int[]	45	784
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,User *>,voi...	1	48
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,Movie *>,v...	1	48
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,int>,void *>	1	40
int	7	28
NetflixDataAOA.exe!std::_Fac_node	1	16
bool[]	1	14

ID	Time	Allocations (Diff)	Heap Size (Diff)
1	0.05s	245 (n/a)	57.79 KB (n/a)
2	0.06s	266 (+21 ↑)	59.68 KB (+1.89 KB ↑)
3	0.10s	278 (+12 ↑)	60.81 KB (+1.13 KB ↑)
4	0.16s	301 (+23 ↑)	62.68 KB (+1.87 KB ↑)
5	0.22s	320 (+19 ↑)	64.24 KB (+1.56 KB ↑)

// Equivalence mod k: The vertices are integers from 0 to n – 1, where k ≤ n. The vertices u and v are connected by an edge if u – v is evenly divisible by k. There are k components, and each component is a complete graph.

```
 --------------testing with EquivalenceModKGraph ----------------------------

 Adjacency list of vertex 0

 head -> 4-> 8

 Adjacency list of vertex 1

 head -> 5-> 9

 Adjacency list of vertex 2

 head -> 6-> 10

 Adjacency list of vertex 3

 head -> 7

 Adjacency list of vertex 4

 head -> 0-> 8

 Adjacency list of vertex 5

 head -> 1-> 9

 Adjacency list of vertex 6

 head -> 2-> 10
 Adjacency list of vertex 7
 head -> 3
 Adjacency list of vertex 8
 head -> 0-> 4
 Adjacency list of vertex 9
 head -> 1-> 5
 Adjacency list of vertex 10
 head -> 2-> 6
 --------------------------------------------

 Following are connected components
0 4 8
1 5 9
2 6 10
3 7

 --------------------------------------------

 finding cycle
Cycle found: 0 8 4 0

 --------------shortest path-----------------------------

Getting the shortest path from 6 to all other nodes.

Printing the shortest paths for node 6.
There is no path from 6 to node 0
There is no path from 6 to node 1
The distance from node 6 to node 2 is: 1 and the path is 6 - 2
There is no path from 6 to node 3
There is no path from 6 to node 4
There is no path from 6 to node 5
```

```
The distance from node 6 to node 6 is: 0 and the path is 6

There is no path from 6 to node 7

There is no path from 6 to node 8

There is no path from 6 to node 9

The distance from node 6 to node 10 is: 1 and the path is 6 - 10
```



Native Memory (NetflixDataAOA.exe) Compare to: Select baseline View Mode: Types View

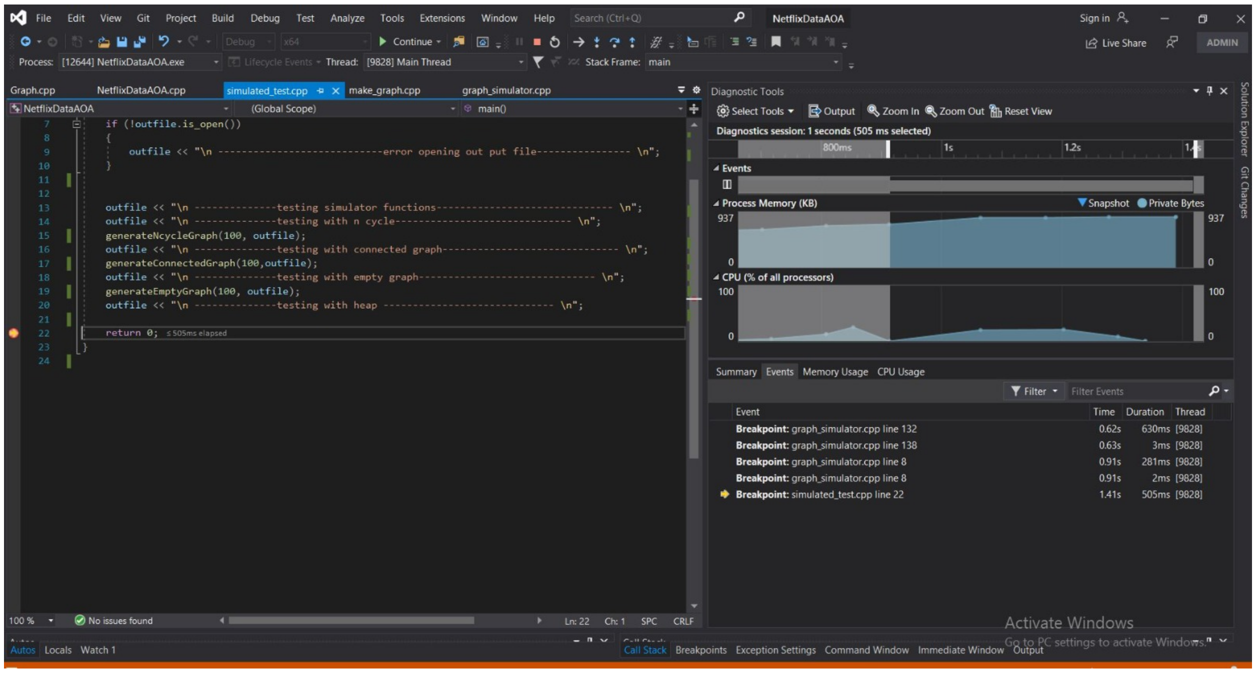
Object Type	Count	Size (Bytes)
NetflixDataAOA.exe!std::vector<int,std::allocator<int> >[]	7	2,584
NetflixDataAOA.exe!std::_Container_proxy	82	1,312
int[]	54	856
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,User *>,voi...	1	48
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,Movie *>,v...	1	48
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,int>,void *>	1	40
int	9	36
NetflixDataAOA.exe!std::_Fac_node	1	16
bool[]	1	14

Diagnostics session: 0 seconds (63 ms selected)

ID	Time	Allocations (Diff)	Heap Size (Diff)
1	0.05s	245 (n/a)	57.79 KB (n/a)
2	0.06s	266 (+21)	59.68 KB (+1.89 KB)
3	0.10s	278 (+12)	60.81 KB (+1.13 KB)
4	0.16s	301 (+23)	62.68 KB (+1.87 KB)
5	0.22s	320 (+19)	64.24 KB (+1.56 KB)
6	0.28s	343 (+23)	66.01 KB (+1.77 KB)

Simulated runtime and peak memory analysis:

When n=100 -- 505 ms



Process: [12644] NetflixDataAOA.exe Thread: [9828] Main Thread Stack Frame: main

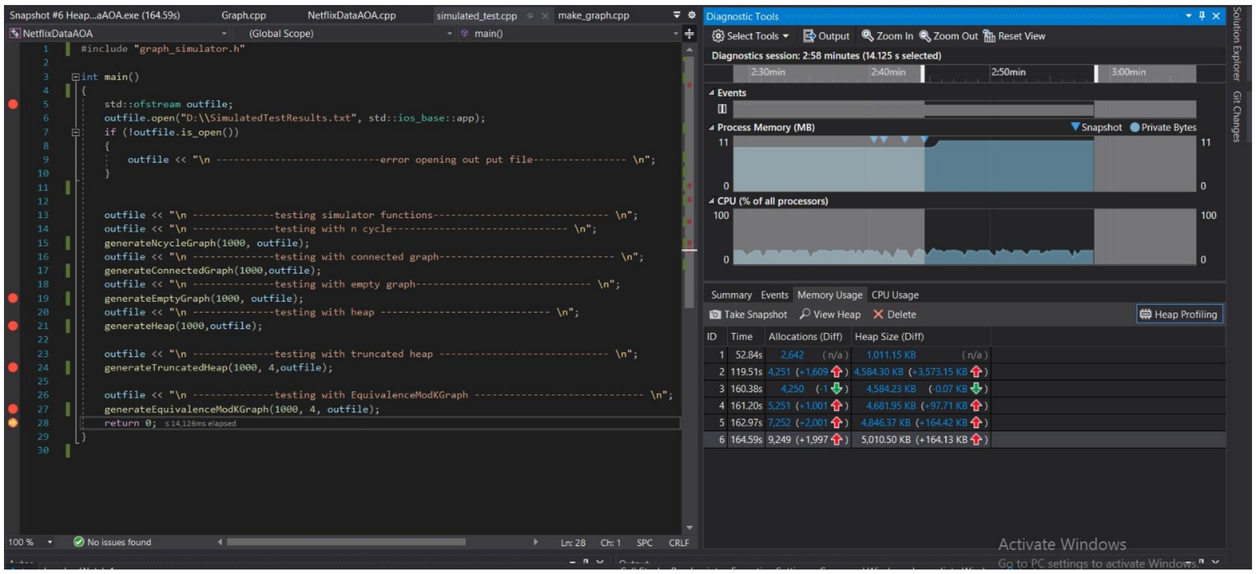
simulated_test.cpp

```
    if (!outfile.is_open())
    {
        outfile << "\n ---------------------------error opening out put file--------------- \n";
    }


    outfile << "\n -------------testing simulator functions----------------------------- \n";
    outfile << "\n -------------testing with n cycle---------------------------- \n";
    generateNcycleGraph(100, outfile);
    outfile << "\n -------------testing with connected graph---------------------------- \n";
    generateConnectedGraph(100,outfile);
    outfile << "\n -------------testing with empty graph---------------------------- \n";
    generateEmptyGraph(100, outfile);
    outfile << "\n -------------testing with heap --------------------------- \n";

    return 0;  ≤ 505ms elapsed
}
```

Diagnostics session: 1 seconds (505 ms selected)

Event	Time	Duration	Thread
Breakpoint: graph_simulator.cpp line 132	0.62s	630ms	[9828]
Breakpoint: graph_simulator.cpp line 138	0.63s	3ms	[9828]
Breakpoint: graph_simulator.cpp line 8	0.91s	281ms	[9828]
Breakpoint: graph_simulator.cpp line 8	0.91s	2ms	[9828]
Breakpoint: simulated_test.cpp line 22	1.41s	505ms	[9828]

When n=1000 – 2.58 mins

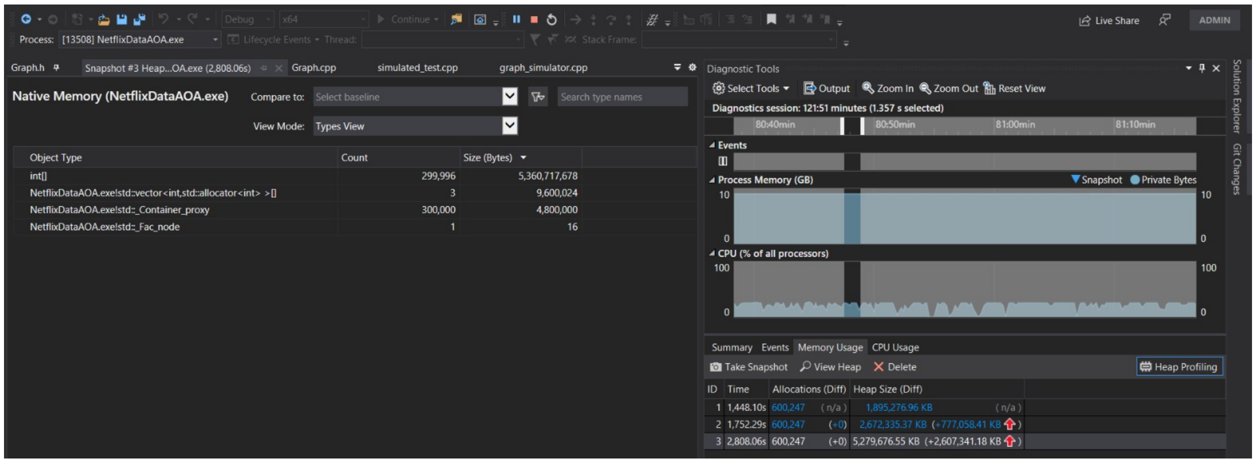


```cpp
#include "graph_simulator.h"

int main()
{
    std::ofstream outfile;
    outfile.open("D:\\SimulatedTestResults.txt", std::ios_base::app);
    if (!outfile.is_open())
    {
        outfile << "\n --------------------------error opening out put file--------------- \n";
    }


    outfile << "\n -------------testing simulator functions---------------------------- \n";
    outfile << "\n -------------testing with n cycle---------------------------- \n";
    generateNcycleGraph(1000, outfile);
    outfile << "\n -------------testing with connected graph---------------------------- \n";
    generateConnectedGraph(1000,outfile);
    outfile << "\n -------------testing with empty graph---------------------------- \n";
    generateEmptyGraph(1000, outfile);
    outfile << "\n -------------testing with heap ---------------------------- \n";
    generateHeap(1000,outfile);

    outfile << "\n -------------testing with truncated heap ---------------------------- \n";
    generateTruncatedHeap(1000, 4,outfile);

    outfile << "\n -------------testing with EquivalenceModKGraph ---------------------------- \n";
    generateEquivalenceModKGraph(1000, 4, outfile);
    return 0;  ≤ 14,126ms elapsed
}
```

Diagnostics session: 2:58 minutes (14.125 s selected)

ID	Time	Allocations (Diff)	Heap Size (Diff)
1	52.84s	2,642 (n/a)	1,011.15 KB (n/a)
2	119.51s	4,251 (+1,609 ⬆)	4,584.30 KB (+3,573.15 KB ⬆)
3	160.38s	4,250 (-1 ⬇)	4,584.23 KB (-0.07 KB ⬇)
4	161.20s	5,251 (+1,001 ⬆)	4,681.95 KB (+97.71 KB ⬆)
5	162.97s	7,252 (+2,001 ⬆)	4,846.37 KB (+164.42 KB ⬆)
6	164.59s	9,249 (+1,997 ⬆)	5,010.50 KB (+164.13 KB ⬆)

N=10000: 121.51 mins



Process: [13508] NetflixDataAOA.exe

Native Memory (NetflixDataAOA.exe)

Object Type	Count	Size (Bytes)
int[]	299,996	5,360,717,678
NetflixDataAOA.exe!std::vector<int,std::allocator<int> >[]	3	9,600,024
NetflixDataAOA.exe!std::_Container_proxy	300,000	4,800,000
NetflixDataAOA.exe!std::_Fac_node	1	16

Diagnostics session: 121:51 minutes (1.357 s selected)

ID	Time	Allocations (Diff)	Heap Size (Diff)
1	1,448.10s	600,247 (n/a)	1,895,276.96 KB (n/a)
2	1,752.29s	600,247 (+0)	2,672,335.37 KB (+777,058.41 KB ⬆)
3	2,808.06s	600,247 (+0)	5,279,676.55 KB (+2,607,341.18 KB ⬆)

Make graph.cpp

This file contains the construction of graph based on the adjacency criteria.

Criteria1:

Our first criteria is to say that 2 customers have a relationship/similar taste when both of them liked at least 4 videos in the recent time

Assumptions: we assume that if the customer have given rating at least 4 then he must have liked the movie and considering movie ratings in the recent year (i.e, took after 2004)

For this to implement, while reading the data files for a movie if the customer gives more than 4 rating in the recent year (2005 or above) we store the movieid+rating as unique key against the customer key.

So at the end we will have customer map where each customer points to all the movies satisfying the above conditions.

And then we iterate over each customer to find the intersection of those unique keys at least 4 in common and for those we call ADD_EDGE API and after constructing graph we are printing the graph(adj list) and connected components in graph , finding the cycle path and finding shortest path from 1st customer to all.

Data structure used is: each customer pointer holds userid and unique review keys.

All the customers reviews are stored in a map whose key is customer_id and contains the unique review keys.

```
class User {
public:
    int user_id;
    vector<int> reviews;

    }
```

Output is printed in **netflixtest1.txt** including in the zip file.

Sample output:

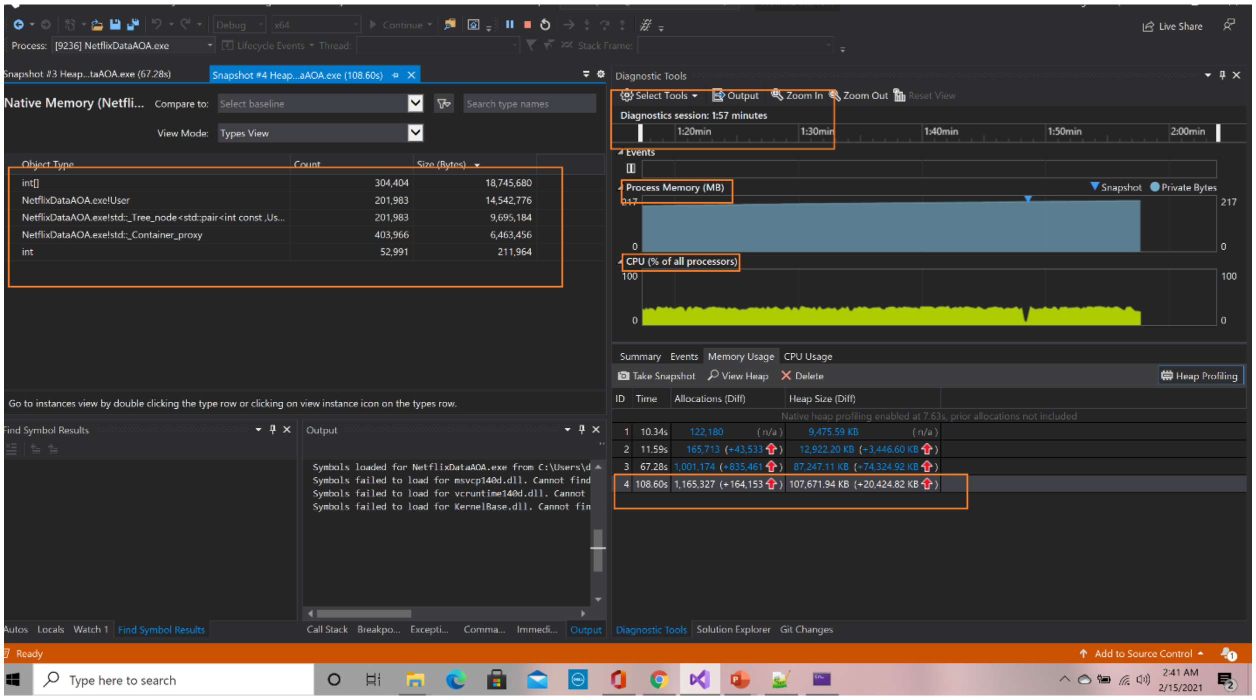
```
Adjacency list of vertex 0

 head -> 1-> 11-> 12-> 17-> 3->....->247-> 254-> 258-> 260-> 286-> 293-> 323-> 332-> 347-> 351-> 376-> 384-> 418-> 442-> 446-> 468-> 483->
486-> 501-> 505-> 506-> 507-> 528-> 541-> 622-> 630-> 632-> 671-> 679-> 693-> 707-> 709-> 722-> 724-> 737-> 745-> 765-> 770-> 771-> 844->
849-> 864-> 868-> 887-> 889-> 915-> 918-> 924-> 932-> 938-> 963-> 966-> 971-> 975-> 981-> 983-> 988-> 998-> 414585-> 414600-> 414624->
414629-> 414631-> 414632-> 414638-> 414660-> 414668-> 414675-> 414681-> 414691-> 414694-> 414702-> 414708-> 414710-> 414723

 Adjacency list of vertex 1

 head -> 0-> 2> 7-> 10-> 11->..........................

.

------------------------------------------

 finding cycle

Cycle found: 0 11 2 1 0

 --------------shortest path-----------------------------
```

```
Getting the shortest path from 1 to all other nodes.

Printing the shortest paths for node 1.

Getting the shortest path from 1 to all other nodes.

Printing the shortest paths for node 1.

The distance from node 1 to node 0 is: 1 and the path is 1 - 0

The distance from node 1 to node 1 is: 0 and the path is 1

The distance from node 1 to node 2 is: 1 and the path is 1 - 2

The distance from node 1 to node 3 is: 2 and the path is 1 - 31 - 3

The distance from node 1 to node 4 is: 2 and the path is 1 - 1246 - 4

The distance from node 1 to node 5 is: 2 and the path is 1 - 41 - 5

The distance from node 1 to node 6 is: 2 and the path is 1 - 17 - 6

The distance from node 1 to node 7 is: 1 and the path is 1 - 7

.

.

.There is no path from 1 to node 414730

The distance from node 1 to node 414731 is: 1 and the path is 1 - 414731

The distance from node 1 to node 414732 is: 2 and the path is 1 - 54 - 414732

The distance from node 1 to node 414733 is: 2 and the path is 1 - 135 - 414733

The distance from node 1 to node 414734 is: 2 and the path is 1 - 13 - 414734

The distance from node 1 to node 414735 is: 1 and the path is 1 – 414735
```

**-→ Here in graph we printed customer indexes and not customer IDS if we want to display as customerIDs graphs while printing we just access V[index] which gives the customerIDs

And for runtime and peak memory analysis we used the visual studio profiling tool

For this criteria it took 235.49 mins and the heap size and objects allocations are shown in below screenshot.



Process: [9236] NetflixDataAOA.exe

Snapshot #3 Heap...taAOA.exe (67.28s) | Snapshot #4 Heap...aAOA.exe (108.60s)

Native Memory (Netfli... Compare to: Select baseline

View Mode: Types View

Object Type	Count	Size (Bytes)
int[]	304,404	18,745,680
NetflixDataAOA.exe!User	201,983	14,542,776
NetflixDataAOA.exe!std::_Tree_node<std::pair<int const ,Us...	201,983	9,695,184
NetflixDataAOA.exe!std::_Container_proxy	403,966	6,463,456
int	52,991	211,964

Go to instances view by double clicking the type row or clicking on view instance icon on the types row.

Output

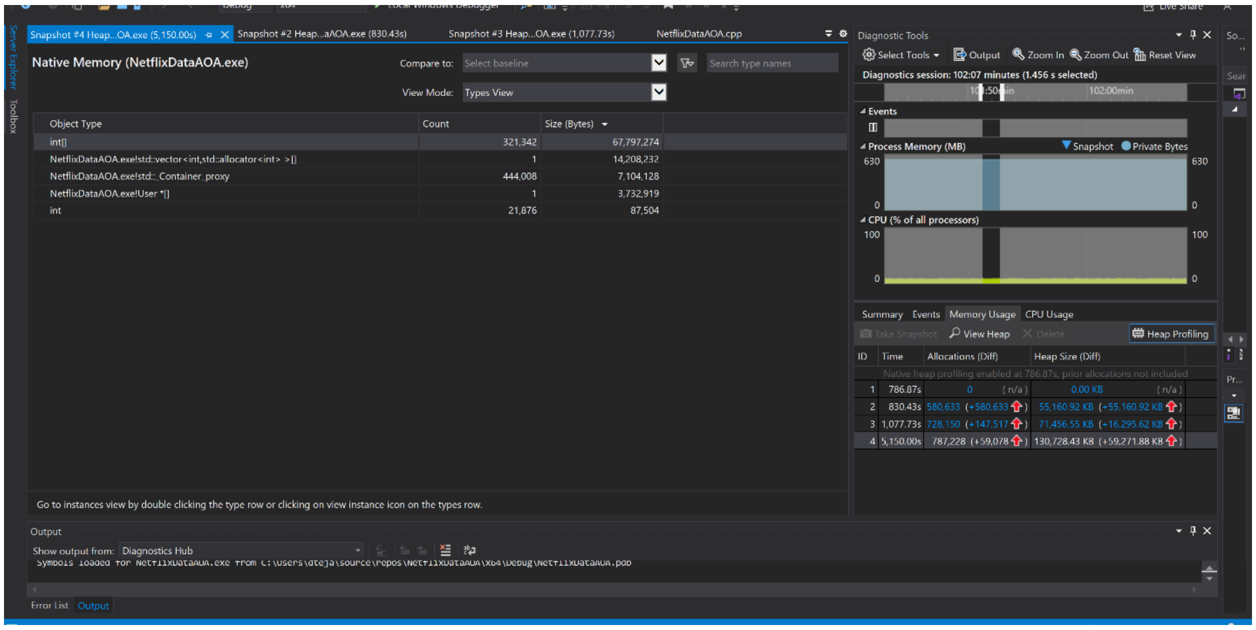
```
Symbols loaded for NetflixDataAOA.exe from C:\Users\d
Symbols failed to load for msvcp140d.dll. Cannot find
Symbols failed to load for vcruntime140d.dll. Cannot
Symbols failed to load for KernelBase.dll. Cannot fin
```

Diagnostic Tools

Diagnostics session: 1:57 minutes

Native heap profiling enabled at 7.63s, prior allocations not included

ID	Time	Allocations (Diff)	Heap Size (Diff)
1	10.34s	122,180 (n/a)	9,475.59 KB (n/a)
2	11.59s	165,713 (+43,533)	12,922.20 KB (+3,446.60 KB)
3	67.28s	1,001,174 (+835,461)	87,247.11 KB (+74,324.92 KB)
4	108.60s	1,165,327 (+164,153)	107,671.94 KB (+20,424.82 KB)



Snapshot #4 Heap...OA.exe (5,150.00s) | Snapshot #2 Heap...aAOA.exe (830.43s) | Snapshot #3 Heap...OA.exe (1,077.73s) | NetflixDataAOA.cpp

Native Memory (NetflixDataAOA.exe) Compare to: Select baseline

View Mode: Types View

Object Type	Count	Size (Bytes)
int[]	321,342	67,797,274
NetflixDataAOA.exe!std::vector<int,std::allocator<int> >[]	1	14,208,232
NetflixDataAOA.exe!std::_Container_proxy	444,008	7,104,128
NetflixDataAOA.exe!User *[]	1	3,732,919
int	21,876	87,504

Go to instances view by double clicking the type row or clicking on view instance icon on the types row.

Output

Show output from: Diagnostics Hub

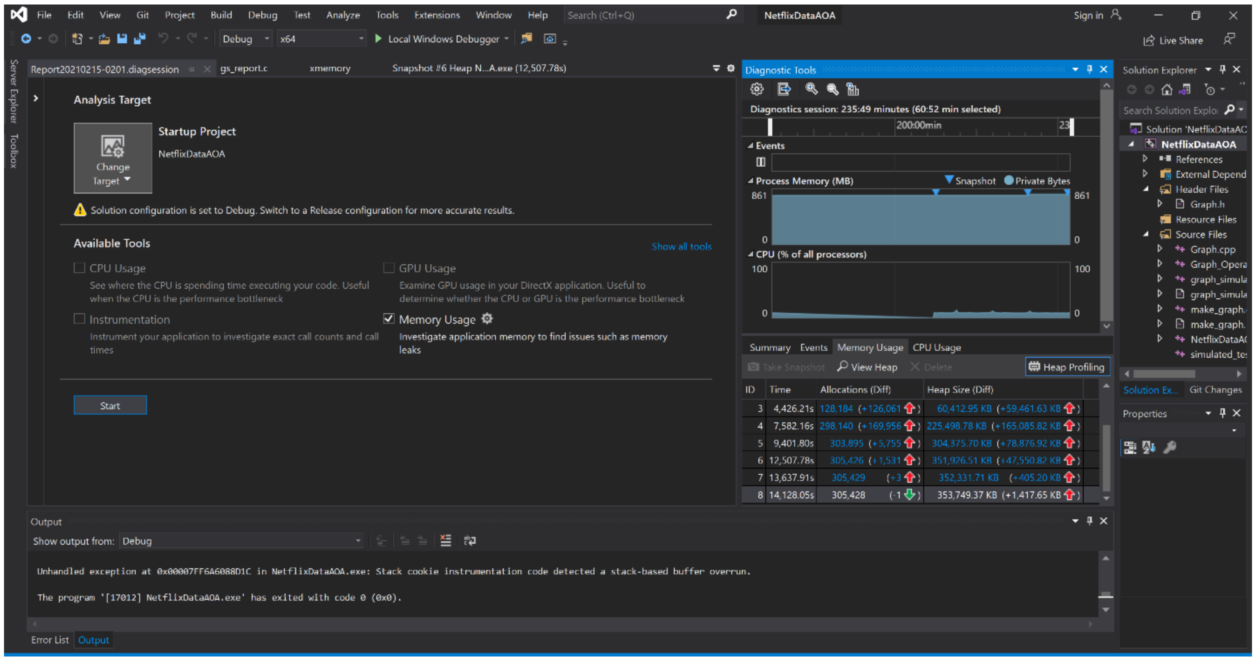
```
Symbols loaded for NetflixDataAOA.exe from C:\Users\dteja\source\repos\NetflixDataAOA\x64\Debug\NetflixDataAOA.pdb
```

Diagnostic Tools

Diagnostics session: 102:07 minutes (1.456 s selected)

Native heap profiling enabled at 786.87s, prior allocations not included

ID	Time	Allocations (Diff)	Heap Size (Diff)
1	786.87s	0 (n/a)	0.00 KB (n/a)
2	830.43s	580,633 (+580,633)	55,160.92 KB (+55,160.92 KB)
3	1,077.73s	728,150 (+147,517)	71,456.55 KB (+16,295.62 KB)
4	5,150.00s	787,228 (+59,078)	130,728.43 KB (+59,271.88 KB)


Analysis Target

Startup Project
NetflixDataAOA

Solution configuration is set to Debug. Switch to a Release configuration for more accurate results.

Available Tools

- ☐ CPU Usage — See where the CPU is spending time executing your code. Useful when the CPU is the performance bottleneck
- ☐ GPU Usage — Examine GPU usage in your DirectX application. Useful to determine whether the CPU or GPU is the performance bottleneck
- ☐ Instrumentation — Instrument your application to investigate exact call counts and call times
- ☑ Memory Usage — Investigate application memory to find issues such as memory leaks

Diagnostics session: 235:49 minutes (60:52 min selected)

ID	Time	Allocations (Diff)	Heap Size (Diff)
3	4,426.21s	128,184 (+126,061 ⬆)	60,412.95 KB (+59,461.63 KB ⬆)
4	7,582.16s	298,140 (+169,956 ⬆)	225,498.78 KB (+165,085.82 KB ⬆)
5	9,401.80s	303,895 (+5,755 ⬆)	304,375.70 KB (+78,876.92 KB ⬆)
6	12,507.78s	305,426 (+1,531 ⬆)	351,926.51 KB (+47,550.82 KB ⬆)
7	13,637.91s	305,429 (+3 ⬆)	352,331.71 KB (+405.20 KB ⬆)
8	14,128.05s	305,428 (-1 ⬇)	353,749.37 KB (+1,417.65 KB ⬆)

Output

Show output from: Debug

```
Unhandled exception at 0x00007FF6A6088D1C in NetflixDataAOA.exe: Stack cookie instrumentation code detected a stack-based buffer overrun.

The program '[17012] NetflixDataAOA.exe' has exited with code 0 (0x0).
```

<u>Adjacency criteria 2:</u>

Criteria2:

Our second criteria is to say that 2 customers have a relationship/similar taste when both of them liked at least a movie.

Assumptions: we assume that if the customer have given rating 5 then he must have liked the movie.

For this to implement, while reading the data files for a movie if the customer gives rating 5 then we add that customer in the movie liked users list.

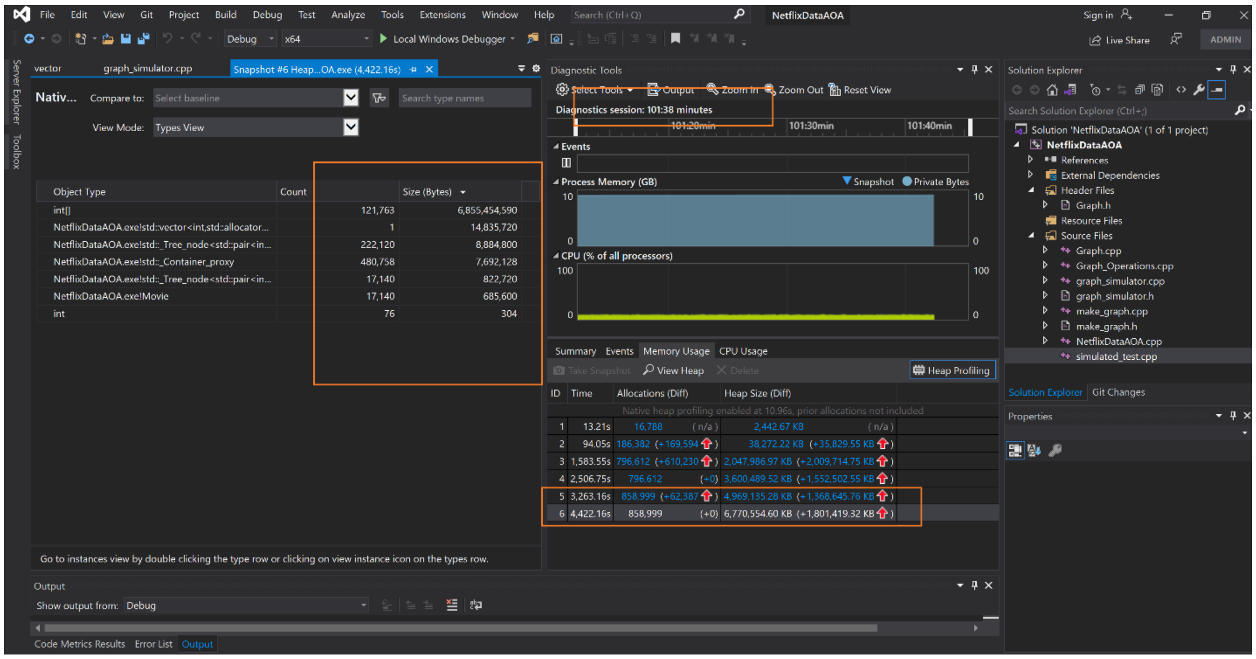
So at the end we will have movies map which says which customers have liked a particular movie.

And then we iterate over each movie to map all the customers in that movie as we can add edge (if there is one movie in common we will add edge).

Output is printed in **netflixtest2.txt** file. Uploaded in the zip file.

Data structure:

```
class Movie {
public:
    int movie_id;
    vector<int> users;
    Movie(int id) {
        movie_id = id;
    }

    void add_user(int userid)
    {
        users.push_back(userid);
    }
};
```



Snapshot #6 Heap...OA.exe (4,422.16s)
Compare to: Select baseline
View Mode: Types View
Diagnostics session: 101:38 minutes
Process Memory (GB)
CPU (% of all processors)

Object Type	Count	Size (Bytes)
int[]	121,763	6,855,454,590
NetflixDataAOA.exe!std::vector<int,std::allocator...	1	14,835,720
NetflixDataAOA.exe!std::_Tree_node<std::pair<in...	222,120	8,884,800
NetflixDataAOA.exe!std::_Container_proxy	480,758	7,692,128
NetflixDataAOA.exe!std::_Tree_node<std::pair<in...	17,140	822,720
NetflixDataAOA.exe!Movie	17,140	685,600
int	76	304

Native heap profiling enabled at 10.96s, prior allocations not included

ID	Time	Allocations (Diff)	Heap Size (Diff)
1	13.21s	16,788 (n/a)	2,442.67 KB (n/a)
2	94.05s	186,382 (+169,594)	38,272.22 KB (+35,829.55 KB)
3	1,583.55s	796,612 (+610,230)	2,047,986.97 KB (+2,009,714.75 KB)
4	2,506.75s	796,612 (+0)	3,600,489.52 KB (+1,552,502.55 KB)
5	3,263.16s	858,999 (+62,387)	4,969,135.28 KB (+1,368,645.76 KB)
6	4,422.16s	858,999 (+0)	6,770,554.60 KB (+1,801,419.32 KB)

Go to instances view by double clicking the type row or clicking on view instance icon on the types row.

Adjacency criteria 3:

Criteria3:

Our third criteria is to say that 2 customers have a relationship/similar taste when both of them liked at least 4 movies with same rating and on same date and he should have given rating recently.

Assumptions: we assume that

i) if the customer have given rating at least 4 then he must have liked the movie and
ii) also considering ratings given only in the recent year (i.e, ratings given after 2004)

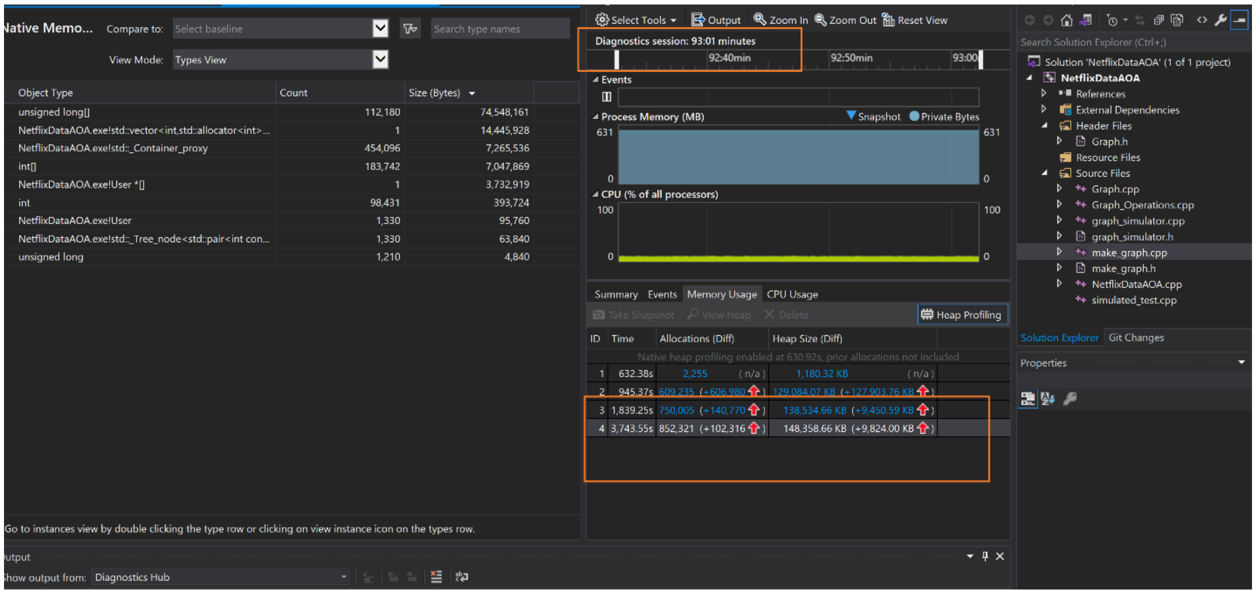
For this to implement, while reading the data files for a movie if the customer gives more than 4 rating in the recent year (2005 or above) we store the **date+movieid** as unique key against the customer key.

So at the end we will have customer map where each customer points to all the movies satisfying the above conditions.

And then we iterate over each customer to find the intersection of those unique keys at least 4 in common and for those we call ADD_EDGE API to construct graph.

For that graph, we are printing the graph (adj list) and connected components in graph , finding the cycle path and finding shortest path from 1st customer to all customers.

- Using same data structure as criteria_1.



Object Type	Count	Size (Bytes)
unsigned long[]	112,180	74,548,161
NetflixDataAOA.exe!std::vector<int,std::allocator<int>...	1	14,445,928
NetflixDataAOA.exe!std::_Container_proxy	454,096	7,265,536
int[]	183,742	7,047,869
NetflixDataAOA.exe!User *[]	1	3,732,919
int	98,431	393,724
NetflixDataAOA.exe!User	1,330	95,760
NetflixDataAOA.exe!std::_Tree_node<std::pair<int con...	1,330	63,840
unsigned long	1,210	4,840

Go to instances view by double clicking the type row or clicking on view instance icon on the types row.

Native heap profiling enabled at 630.92s, prior allocations not included

ID	Time	Allocations (Diff)	Heap Size (Diff)
1	632.38s	2,255 (n/a)	1,180.32 KB (n/a)
2	945.37s	609,235 (+606,980)	129,084.07 KB (+127,903.76 KB)
3	1,839.25s	750,005 (+140,770)	138,534.66 KB (+9,450.59 KB)
4	3,743.55s	852,321 (+102,316)	148,358.66 KB (+9,824.00 KB)

**It took around 340mins but profiling tool ran out of memory after 90 mins of execution

real_test.cpp:

It contains the function calls to adjacency criteria.

In order to test 3 criteria's separately for runtime and memory usage, called the functions individually.

SUMMARY

For simulated data:

For n=100 it took 550ms and memory is 5kb

For n=1000 it took 2.58 mins and memory is 5010.5kb

For n=10000 it took 125 mins and memory is 2607341.18kb

Observed that heap is taking very less time 16 mins for n=10000 and other graphs are taking 0(n^2) time.

All of them are n^2 loops adding edges with mathematical conditions.

<u>For Netflix data :</u>

Each criteria took minimum of 240 mins and minimum memory of 5 mb for filtered data in memory.

Each criteria of ours takes o(n^3) time to process.