VISHAL KUMAR MUNKA
81286177
munkav@ufl.edu

1. COMPILATION STEPS

IDE used : Visual studio 2008
Compiler tool: cl.exe

The Program contains three main files mst.cpp, simpleScheme.cpp and fibHeap.cpp and three Header files simpleScheme.hpp, fibHeap.hpp and common.hpp.
A makefile is used to compile the program.
To compile the Program go to the folder in which the files are and type MAKE on the terminal.
This will create a mst executable.
To run it use inputs like :
./mst –r 1000 10
./mst –f input.txt
./mst –s input.txt
NOTE: The input.txt should be in the same folder as the program files.

2. PROGRAM STRUCTURE AND FUNCTION PROTOTYPES

The file mst.cpp is the file with the main function where the input is taken from the user and according to the input either random graph is generated or user graph is taken as input. The mst.cpp also contains the Random graph generation function createRandomGraph() and connected graph checking functions check_connected() and dfs(). The file simpleScheme.cpp contains the functions related to the class simpleScheme. The file simpleScheme.hpp contains the class definition of the simpleScheme. An object of type simpleScheme is instantiated from the main function to run the simpleScheme for Prims Algorithm.
The file fibHeap.cpp contains the function to run Prims algorithm for the fibonacci heap and the functions for fibonacci heap operations such as removeMin(), pairwiseCombine etc.
An object of fibHeap class is instantiated from the main() to run the Prims algorithm for Fibonacci Scheme.
The file common.hpp contains the common header files and the Adjacency List node structue node.

The simpleScheme.cpp is the file that contains the functions related to the simple scheme. This file has a simpleScheme.hpp associated with it.
The Simple Scheme uses an mst_node structure as below

```
struct mst_node  //used in simpleScheme array
{
	int vertex;
	int weight;
	int parent;
	bool visited;
};
```

The fibHeap_C.cpp is the file that contains functions related to the Fibonacci scheme. This file has a fibHeap.c.hpp associated with it.

The Fibonacci Heap uses a fibNode Structure as below

```
struct fibNode  //used for Fibonacci Heap
{
	int degree;
	fibNode *child;
	int weight;
	int vertex;
	int other_end;
	fibNode * right;
	fibNode * left;
	fibNode * parent;
	bool  childcut;
	bool visited;
};
```

THE FUNCTIONS IN MST.CPP ARE :

1. int main(int argc, char * argv[])

User input is handled by this function and according to the input the graph is generated
The Fibonacci and SimpleScheme objects are initialised from this function and according to the user input the Fibonacci and SimpleScheme functions are called.

2. bool createRandomGraph(vector <vector <node*> > &adj_list, int vertices, int density)

This Function creates a RandomGraph based on the number of vertices and density and if the graph is not connected it redraws the Graph again.

3. bool check_connected(vector <vector <node*> > &adj_list, int vertices)

This functions checks if the graph is connected or not by using a dfs function.

4. void dfs(vector <vector <node*> > &adj_list,vector <bool> &visited, int u, int n)

This function run the dfs algorithm on the adjacency list and changes the vector of visited nodes.
If all the nodes in the Vector are visited that means that the graph is connected.

THE FUNCTIONS IN simpleScheme.cpp ARE:

1. void init_min_q(int size , vector <mst_node*> & min_q)

This function initializes the array from which the minimum is extracted each time in a loop of number of vertices. All the nodes in the array are of type mst_node (node structure for simple scheme) and are initialized with value MAX_INT except node 0 which is initialized with value 0.

2. int mst(vector <mst_node*> &min_q, vector <mst_node*>& mst_list, vector <vector <node*> > &adj_list, int vertex)

This function is used for implementing Prims Algorithm using an array.

3.void printData(int total_cost, vector <mst_node*>&mst_list)

This function is used for printing the generated mst.

The file fibHeap.cpp contains the function for running the prims algorithm using Fibonacci Heap and the Fibonacci Heap function

FUNCTIONS IN fibHeap.cpp ARE:

1. int fibMST(vector <vector <node*>> &adj_list, int vertex, vector <fibNode*> &mst_list)

This is the main function for running the Fibonacci Prims Algorithm.

2. void Insert(fibNode *node)
This function is used for inserting nodes in the top level circular list.

3. void Union(fibNode *child)

This function is used for removing all the children of the removed min and
put them in the top level circular list after the min_ptr

4. void decreaseKey(int vertex, int weight, int other_end)

This function is used for decreasing the weight of a particular node.

5. void cut(fibNode* x, fibNode*y)

This function is used for removing the child of a node if its weight is lesser than its parent and putting them in the top level list. If this causes a second cut from the parent, the cascading cut takes place.

6. void cascading_cut(fibNode * y)
This recursive function is used for removing the parents who have lost two children.

7. fibNode * find(int vertex)

This function is used to find a node with a particular vertic in the heap.

8. fibNode * findInChild(int vertex, fibNode* parent)
This recursive function is used to find node corresponding to a vertex in a particualar child.

9. void pairwiseCombine()

This function does pairwise combining of nodes with equal degree in the root list. This function is called after extracting the minimum from the heap

10. void link(fibNode *y, fibNode *x)

This function is used to make y a child of x. This is used in pairwise combine to combine two nodes of equal degree. y is made a child of x if weight of y is more than weight of x.

11. fibNode* removeMin()
This is perhaps the most important function in the Fibonacci Heap. It is used to remove the smallest elememnt from the heap and bring its children to the root level list. This is also responsible for pairwiseCombining of nodes of equal degrees at the root level.

12. void createNodes(int ver)

This function is used to initialize the heap with nodes equal to the number of vertices and give weight equal to MAX_INT(equivalent to infinity) to all the nodes except the zeroth node(weight 0) as we are assuming we are going to start the mst from this vertex.

13. void addNode(int v, int weight)

Creates a new Node.With all data memebers values set to default except weight and vertex

14. void printData(int total_cost, vector <fibNode*> &mst_list)

Utility function used to print the created MST.

3. COMPARISON (Average Reading after 5 readings at each vertex number and density)

The expecatation from the Comparison is that the Fibonacci Heap will work better than simpleScheme for sparse graphs as the complexity for a simpleScheme is O(V^2) and complexity for Fibonacci Heap implementation is O(VlogV + E). In Sparse graph E is near to number of vertices therefore complexity is of O(VlogV). This is better than O(V^2)
For dense graphs, the performance of the simpleScheme is expected to be better than the Fibonacci Heap as complexity for simpleScheme using array is O(V^2) and complexity for Fibonacci Heap will be O(VlogV + V^2) as in dense graphs E is near to V^2.
NOTE: V is no of vertices.

Fibonacci Heap Scheme

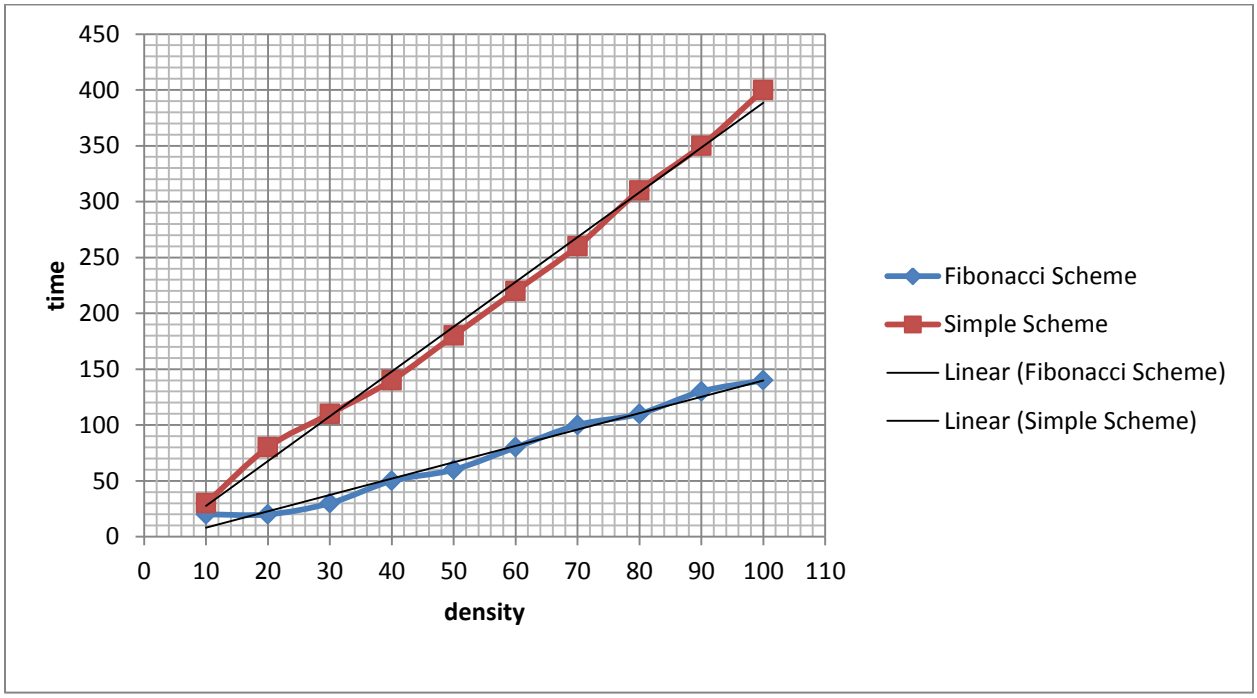
Density	10	20	30	40	50	60	70	80	90	100
1000	20	20	30	50	60	80	100	110	130	140
3000	150	340	540	760	990	1360	1610	1880	2160	2420
5000	550	1190	1860	2550	3260	3980	4680	5400	6100	6830

Simple Heap Scheme

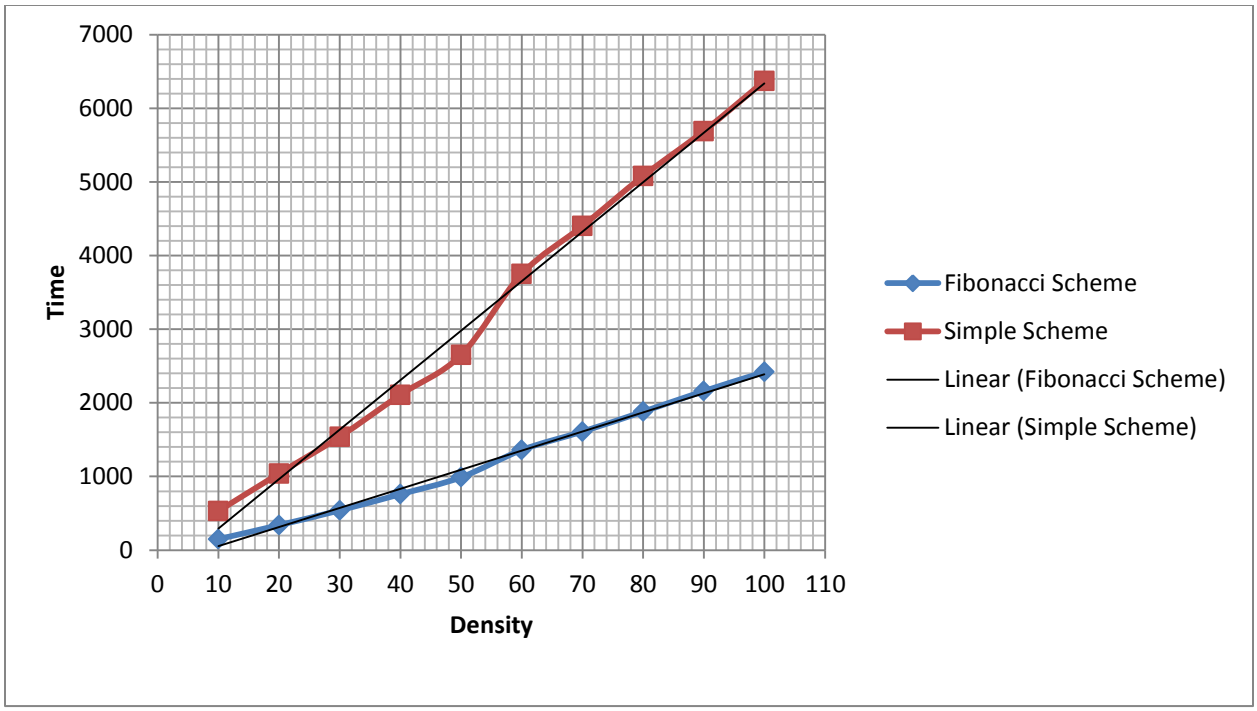
density	10	20	30	40	50	60	70	80	90	100
1000	30	80	110	140	180	220	260	310	350	400
3000	530	1040	1540	2110	2650	3750	4400	5080	5690	6370
5000	2100	3740	5510	7070	8870	3980	11980	13800	15340	17210

The program could was tested till vertex = 8000 and density 50 on cise lab machines using g++.

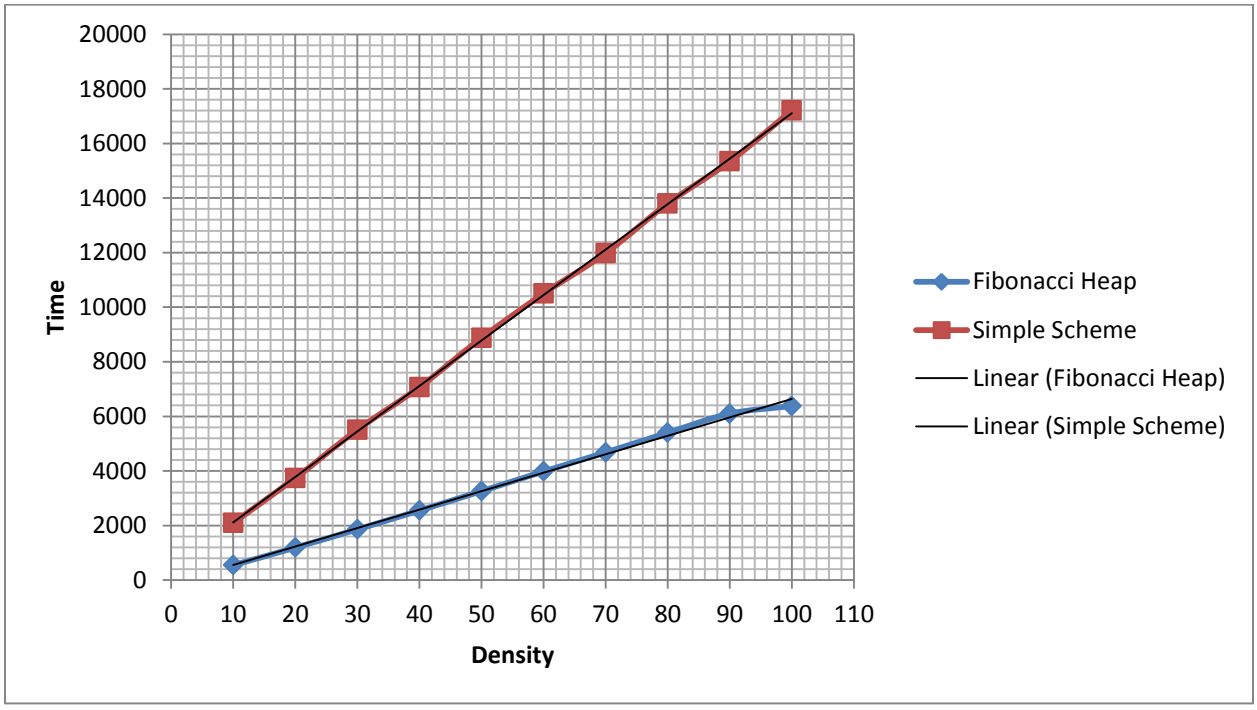
We can clearly see the comparison between Fibonacci and Simple Scheme at different densities using the below graphs



For Vertices n = 1000



For Vertices n = 3000



For Vertices n = 5000

From the Above three graphs we can clearly see that the Fibonacci Heap is performing better than the simple scheme.

We expected the Fibonacci Heap to perform better than the simple scheme for less sparse graphs. The result for this scenario is as expected. For dense graphs we expected the simple scheme to perform better than the Fibonacci Scheme. For my implementation I am not getting this as there are many other factors that influence the Fibonacci Scheme.

For dense graph we expect the Fibonacci heap will be bad in performance as compared to the simple scheme because of the V^2 factor in O(VlogV + V^2). Here we are considering that we will be calling the decrease Key a V^2 number of times for the Prims algorithm but in actuality that is not the case. We are calling decrease key in order of V as the random function produces a lot of very small values. The C++ random function does not produce very uniform values as a result the weight of a vertex is decreased in the beginning to a very small value and does not need to be decreased very often. This reduces the run time of Fibonacci heap at high densities.

NOTE: FOR random graph I am printing the total cost and time at the end so that it is easy to see output for large values.