COP 5536 Fall 2013
Programming Project

Name: Yahui Han UFID: 41442945 Email: siyuan808@ufl.edu

The mst_src_code file contains everything to compile and run this program. It was compiled with g++ running on OS X and Linux.

To compile, just change to directory under mst_src_code and type "make" in the terminal, then the compiler will create the target executable "mst" in this directory according to its "MakeFile".

Function prototypes and program structure.

```
struct Vertex{
    int id; // the index in the vertex array of the graph
    int key; // current key when calculate the mst
    int mstParent; // parent id in the mst
    Color color; // White: not visited, Black: visited
    unordered_map<int, int> edges; //a hash map to store
all the nodes ids and weight that are connected with this
node
};
```

Graph.h

```
    int nVertices; //number of vertices
    Vertex ** vertices; //an array of vertices
    int mstCost; //mst cost after calculating mst.
    private:
    //add an edge into the graph if edge(i,j) not exists
    bool addEdge(int i, int j, int w);
    //initialize the graph and create vertices
    void initialize(int n);
    //dfs visit every node to help check connectivity
    void dfsVisit(int v);
    public:
```

```
    // Three different type of build, to add edges into the graph until  connected
    void build(int n, double d);
    void buildFull(int n);
    void build(const char *fileName);

    // To check whether corrent graph is connected or not
    bool isConnected();
    // Get the weight on edge (u, v)
    int getWeight(int u, int v);
    // reset all the vertices before dfs and prim
    void traversalInitialize();
    // prim's algo to calculate the mst with the help of a
min queue
    void primMST(MinQueue *q);
```

MinQueue
```
// Pure virtual class that declares the interface
any min queue has to offer
    // Get the min and remove it from the queue
    virtual int extractMin() = 0;
    // Decrease the key in node v
    virtual void decreaseKey(int v, int value) = 0;
    // Check is the queue empty
    virtual bool isEmpty() = 0;
```

SimpleQueue: MinQueue
```
// An array implemenatation of min queue

// Fibonacci heap node
struct fnode {
    Type data; // An pointer to a vertex in a graph
    //Used for circular doubly linked list of siblings
    fnode* left;
    fnode* right;

    fnode* child;
    fnode* parent; //Pointer to parent node
    int degree;

    /* True if node has lost a child since it became a
child of its current parent.
     * Set to false by remove min, which is the only
```

```
operation that makes one node a child of another.
	 * Undefined for a root node.
	 */
	bool childCut;
};
```

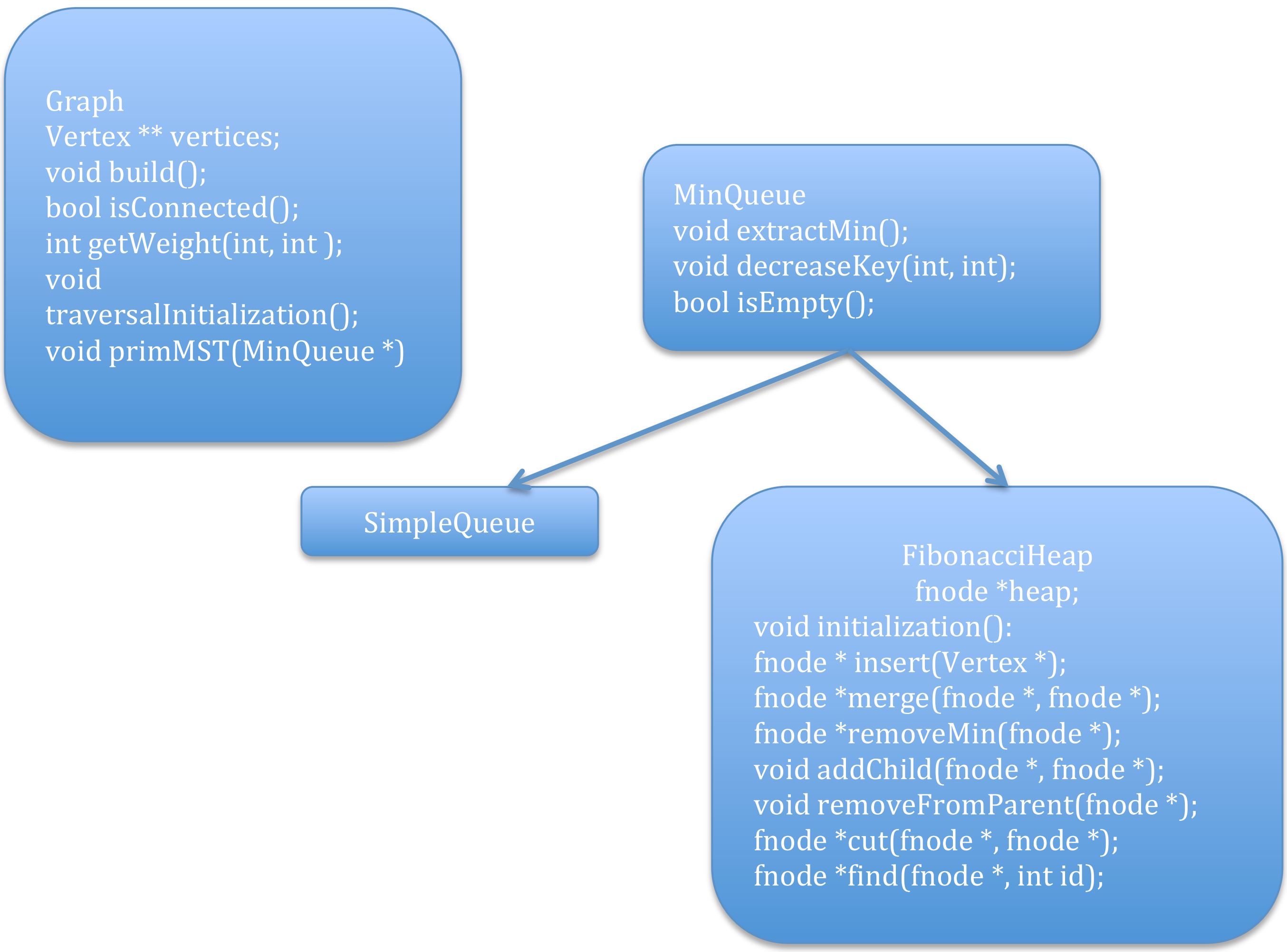
FibonacciHeap: MinQueue //Fibonacci heap implementation of min queue

```
	fnode *heap; // fibonacci heap's min pointer
	// an array help to look up the fnode according to the id of a vertex
	fnode* *f_map;
	// initialize the fib heap with the vertices in graph g
	void initialize(Graph *g);
	// insert a vertex into current fib heap
	fnode *insert(Type v);
	// merge two doubly linked list into one
	fnode *merge(fnode *a, fnode *b);
	// remove min and do the pairwise combine
	fnode *removeMin(fnode *n);
	// add a child fnode into a fnode
	void addChild(fnode *parent, fnode *child);
	// excise all the node in the list from their parent
	void removeAllFromParent(fnode *n);
	// excise fnode n from its parent and re-insert it into the top list
	fnode *cut(fnode *heap, fnode *n);
	// find the fnode according to the id of a vertex
	fnode *find(fnode *heap, int v);
	// delete every fnode and free the memory
	void deleteAll(fnode *n);
```

Main.cpp

```
// calculate the mst of g with simple queue
void mstSimple(Graph *g);
// calculate the mst of g with fibonacci heap
void mstFibonacci(Graph *g)
```

Program structure



Comparison

Expectation:

Since we know in theory prim's algorithm runs in $O(n^2)$ with an array and in O(nlgn + e) with fibonacci heap, where $e=d*n^2$. We may expect that with when the number of nodes is large enough, and the graph is sparse, which means e is much smaller than n^2, Fibonacci heap will be superior to an array. However, when the number of nodes is small, Fibonacci will be overkill and can't compensate the time to establish it, and also when the graph is dense, which means e is close to n^2, Fibonacci won't have a advantage over array in this situation either.

Experiment result:

(in microsecond)

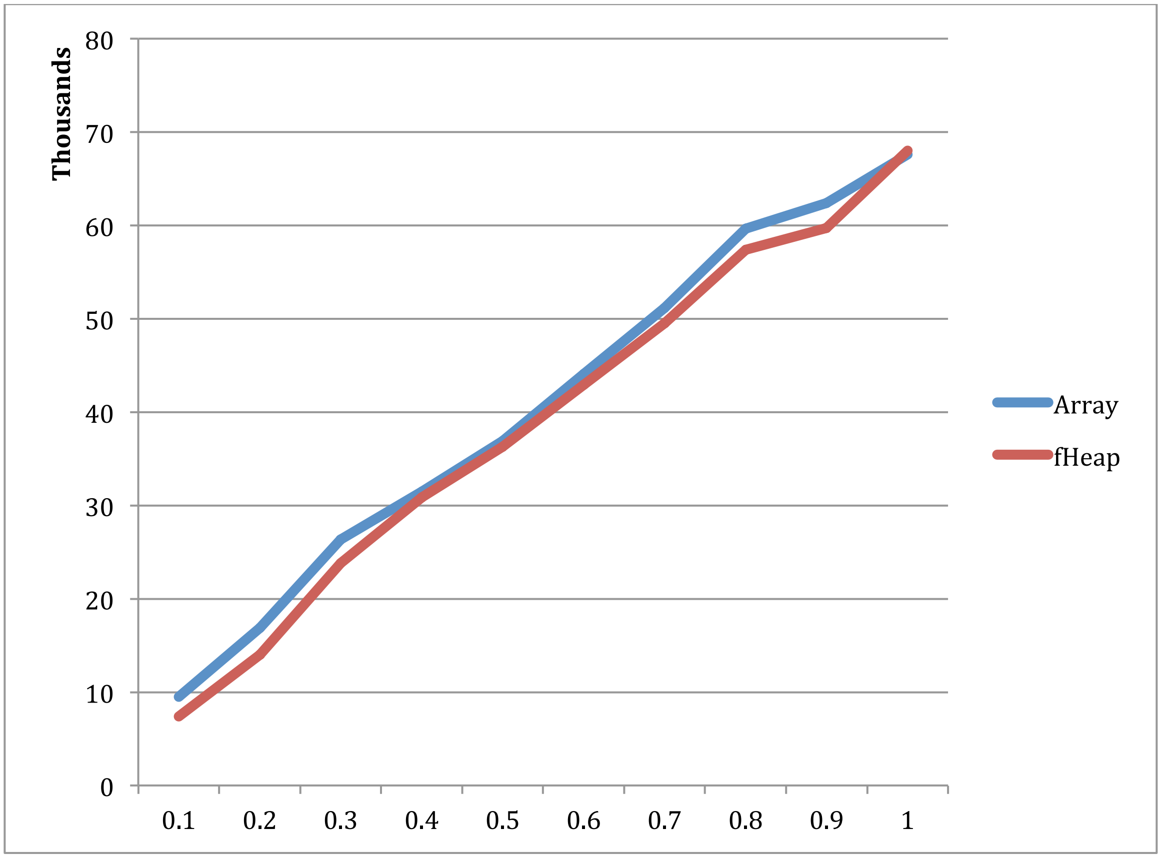


Figure 1: n = 1000

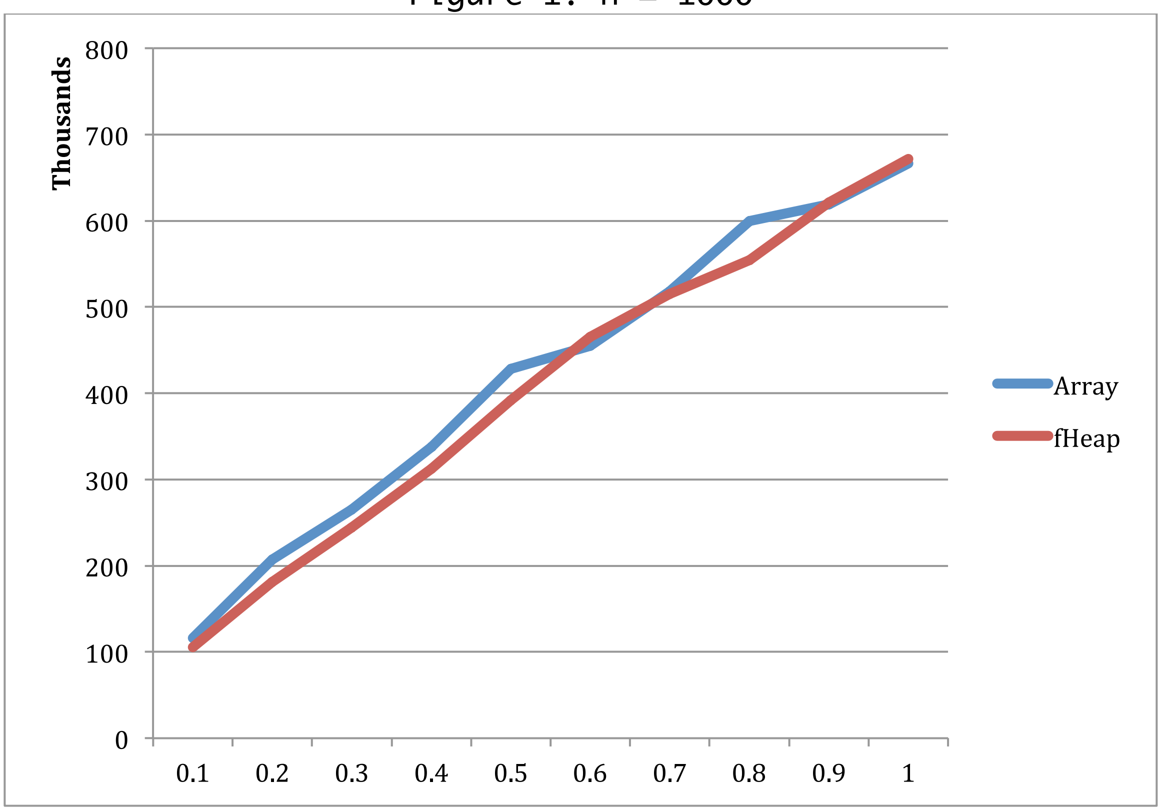


Figure 2: n = 3000

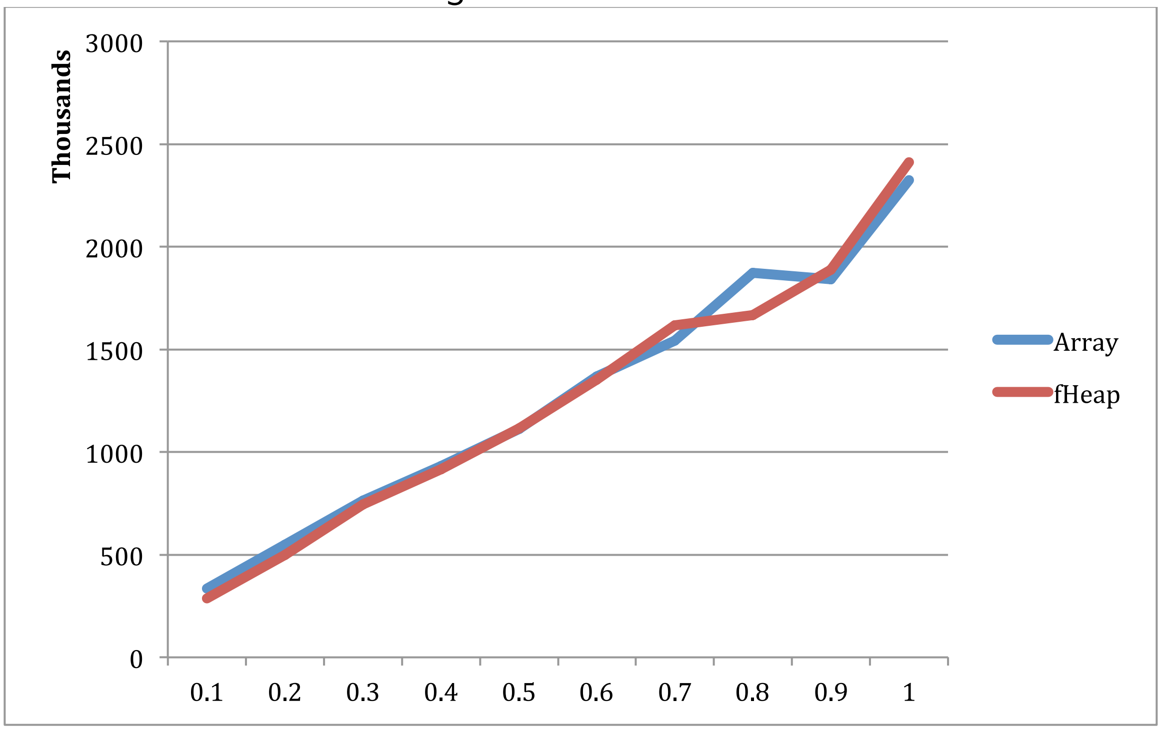


Figure 3: n = 5000

A complete experiment data can be found in result.xlsx.
The result of the experiment confirms our expectation.