Simon Chen
N10013388
sc4900
Homework 11

1) O(nlogn + klogn)
nlogn is to build the heap from the array
klogn is thhe time it takes to get the min and

2)
insert(x) : O(1) amortized
deleteMin(x) : O(ln)
findMin(x) : O(n)

3)

```
        a)               190
              /                     \
           380                       200
        /            \         /             \
  388               401     277              270
 /     \         /
1000   399     432
```

b) Insert 197

```
                       190
              /                     \
           380                       200
        /            \         /             \
  388               401     277              270
 /     \         /     \
1000   399     432     197
```

The 197 will get swapped with the parent until it is less than the parent.
The array will change as followed:

| Sentinel | 190 | 380 | 200 | 388 | 401 | 277 | 270 | 1000 | 399 | 432 | 197 |

197 swapped: with the parent, 401

| Sentinel | 190 | 380 | 200 | 388 | 197 | 277 | 270 | 1000 | 399 | 432 | 401 |

197 swapped: with the parent, 380

| Sentinel | 190 | 197 | 200 | 388 | 380 | 277 | 270 | 1000 | 399 | 432 | 401 |

Done with swapping. Tree is now:

```
                              190
                     /                              \
                197                              200
          /                  \            /                    \
    388                    380     277                    270
   /         \              /       \
1000    399          432     401
```

4)

a)

| Sentinel | 2 | 3 | 4 | 6 | 8 | 70 | 5 | 9 | 7 | 19 | 27 |

b)

Root is removed:

```
                              3
                     /                              \
                6                                 4
          /                  \            /                    \
    7                        8        70                        5
   /         \              /
9          27          19
```

5)

```
template <class Comparable>
void BinaryHeap<Comparable>::insert( const Comparable & x)
{
        if (theSize + 1 == array.size()){
                array.resize( array.size() * 2 + 1);
        }

        int hole = ++theSize;

        for ( ; x < array[hole/2] && hole > 0; hole /= 2)
                array[hole] = move(array[hole/2])

        array[hole] = x;
}
```

6)

The results of using the linear-time algorithm is to first insert the tree then make the tree follow the order of a binary tree heap.

The array will be | Sentinel | 10 | 12 | 1 | 14 | 6 | 5 | 8 | 15 | 3 | 7 | 4 | 11 | 10 | 0 |

The tree will be

```
                      10
             /                   \
         12                       1
      /           \        /            \
  14               6     5               8
 /     \        /    \  /    \         /
15      3      7      4  11   10       0
```

Now percolate up to make it have the heap order while swapping. The end result will be.

```
                      0
             /                   \
         3                        1
      /           \        /            \
  12               4     5               8
 /     \        /    \  /    \         /
15      14     7      6  11   10       10
```

The array will be | Sentinel | 0 | 3 | 1 | 12 | 4 | 5 | 8 | 15 | 14 | 7 | 6 | 11 | 10 | 10 |

7)

phase	distance/pre							Visit	Queue
	G	A	B	C	D	E	F		
init	0/-								G
1	0/-	1/G					1/G	G	F A
2	0/-	1/G				2/F	1/G	F	A E
3	0/-	1/G				2/F	1/G	A	E
4	0/-	1/G			3/E	2/F	1/G	E	D
5	0/-	1/G	4/E	4/E	3/E	2/F	1/G	D	
5	0/-	1/G	4/E	4/E	3/E	2/F	1/G	B	
5	0/-	1/G	4/D	4/D	3/E	2/F	1/G	C	

G → F → E → D → C

8)

Adj. List

A → (D,3) → (B,6) → (E,1)
B → (A,6) → (D,0) → (C,5)
C → (D,12) → (B,5) → (E,18) → (G,5)
D → (A,3) → (B,0) → (C,12) → (F,3)
E → (A,1) → (C,18) → (G,19)
F → (D,3) → (G,14)
G → (E,19) → (C,5) → (F,14)

Adj. Matrix

A	B	C	D	E	F	G
	6		3	1		
6		5	0			
	5		12	18		5
3	0	12			3	
1		18				19
			3			14
		5		19	14	

9)

phase	distance/pre A	B	C	D	E	F	G	Visit	Discovered
init	0								A
1	0	6/A		3/A	1/A			A	B D E
2	0	6/A	19/E	3/A	1/A		20/E	E	B D C G
3	0	3/A	15/E	3/A	1/A	6/D	20/E	D	B C G F
4	0	3/D	8/B	3/A	1/A	6/D	20/E	B	C G F
5	0	3/D	8/B	3/A	1/A	6/D	20/E	F	C G
5	0	3/D	8/B	3/A	1/A	6/D	13/E	C	G
5	0	3/D	8/B	3/A	1/A	6/D	13/E	G	

10)

```
          A
   8  /          \ 1
   B _-10_     C
```

Dijkstra's algorithm will close off the path to C and ignore that the shortest weighted path to C is A B C

11) You can modify Dijstra's algorithm to keep track of all the paths so that when the path is the same value as the closed path it will count that as an alternative path.

12)

DFS Starting from the Top to Bottom
A → D → G → C → B → A
A → B (already discovered)
A → E
(everything is discovered now)

BFS Starting from Top to Bottom
A → D
A → B
A → E
D → F
B → C
E → C (Already discovered)
F → G
C → G(Already discovered)
G → C(Already discovered)
G → E(Already discovered)

13)