Amber Harding

Assignment 4

CSCI 261

4.

The expected case time complexity for the select algorithm is characterized by the following recurrence.

$$
\begin{aligned}
T(1) &= 2 \\
T(n) &= (n+1) + \frac{1}{n}\sum_{q=1}^{n-1} T(q)
\end{aligned}
$$

Use techniques, including iteration, to solve the recurrence exactly.

T(1) = 2

T(n) = (n + 1) + $\frac{1}{n}\sum_{q=1}^{n-1} T(q)$

T(2) = (2 + 1) + ½ * $\sum_{q=1}^{1}$ T(q)

= 3 + ½(T(1))

= 3 + ½ * 2

= 3 + 1

= 4

T(2) = 4

T(3) = (3 + 1) + ⅓ (T(1) + T(2))

= 4 + ⅓ (2+4)

= 4 + 6/3

= 6

T(3) = 6

T(4) = (4 + 1) + ¼ (T(1) + T(2) + T(3))

= 4 + 1 + ¼(2 + 4 + 6)

= 8

T(4) = 8

T(n) = 2n

5.

a. $\sum_{v \in V} O(out - degree(v)) = O(|E| + |V|)$

B. The expected time to look up an edge is O(1), worst case could be O(Vertices).

An alternative is to first sort the vertices then binary search could be used which would make the worst case look up time O(lg|V|) but the disadvantage is that the expected lookup time is now slower than O(1).

6

a.

vertex	r	s	t	u	v	w	x	y
d	4	3	1	0	5	2	1	1
π	s	w	u	Nil	r	t	u	u

b. The textbook uses Black to distinguish nodes that have been dequeued and Gray to distinguish nodes that have been enqueued. This can be represented using 0 or 1 as individual bits.
c. The value d assigned to a vertex is independent of the order of the adjacency lists. To prove this the theorem that proves the correctness of BFS states that v, d = S(s,v) at the termination of BFS. S(s,v) is a property of the underlying graph, representation of the graph will not change because the d values are equal to S(s,v) and S(s,v) is invariant

for any ordering of the adjacency lists, S(s,v) will not change.

The given worked out procedure, states that in the adjacency list for w, t precedes x. Also in this procedure we have that $u.\pi = t$.

Suppose instead of x preceding t in the adjacency list of w. Then it would get added to the queue before t. Which means that it would u as it's child before we have a chance to process the children of t. This means that $u.\pi = x$ in this different ordering of the adjacency list for w.

7.

```
DFS(G, s):
//Where G is graph and s is source vertex
   let S be stack
   S.push( s )        //Inserting s in stack
   mark s as visited.
   while ( S is not empty):
     //Pop a vertex from stack to visit next
     v  =  S.top( )
     S.pop( )
     //Push all the neighbours of v in stack that are not visited
    for all neighbours w of v in Graph G:
       if w is not visited :
            S.push( w )
            mark w as visited
```