Analysis of Algorithms
Homework 4

Arthur Nunes-Harwitt

1. (project) Write a function max2 to compute the maximum of two numbers. It should use *no* comparisons. Assume that other mathematical operations such as $\sqrt{x}$, x^2, $|x|$, etc. do not use any comparisons.

2. (project) Implement the functional select algorithm called fSelect. It should take two parameters: a list and an index. It should return the specified element of the list.

3. (project) Implement an imperative select algorithm called iSelect. It should take two parameters: an array and an index. It should return the specified element of the array.

The implementation should avoid allocating memory by calling iSelectHelper that takes an array and an index as well as low and high slice parameters that indicate the region under consideration. Further, the in-place partition algorithm from quick sort should be used.

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) CLRS 22.1-1

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

(b) CLRS 22.1-8

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) CLRS 22.2-2

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) Show that using a single bit to store each vertex color suffices by arguing that the BFS procedure would produce the same result if line 18 were removed.

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) CLRS 22.2-5

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. CLRS 22.3-7

```
DFS-iterative (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
```