CSCI 570 - Fall 2019 - HW 2

Junzhe Liu, 2270250947

September 8, 2019

1 Graded Problems

1.1 Kleinberg and Tardos, Chapter 2, Exercise 3.

$$\sqrt{2n} \; < \; n+10 \; < \; n^2 \log n \; < \; n^{2.5} \; < \; 10^n \; < \; 100^n \tag{1}$$

1.2 Kleinberg and Tardos, Chapter 2, Exercise 4.

$$n^{\frac{4}{3}} \; < \; n(\log n)^3 \; < \; n^{\log n} \; < \; 2^{\sqrt{\log n}} \; < \; 2^n \; < \; 2^{n^2} \; < \; 2^{2^n} \tag{2}$$

1.3 Kleinberg and Tardos, Chapter 2, Exercise 5.

1.3.1 $\log_2 f(n)$ is $O(\log_2 g(n))$

This is false. If $g(n) = 1$ and $f(n) = 2$ for all n, then by choosing $c \geq 2$ we will have $f(n) = O(g(n))$. However, $O(\log_2 g(n))$ will be 0.

1.3.2 $2^{f(n)}$ is $O(2^{g(n)})$

False. Let $f(n) = 2n$ and $g(n) = n$, then $2^{f(n)} = 4^n > 2^n = 2^{g(n)}$ for all $n > 0$.

1.3.3 $f(n)^2$ is $O(g(n)^2)$

True. We have $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. Then:

$$f(n)^2 \leq c^2 \cdot g(n)^2 = C \cdot g(n)^2 = O(g(n)^2). \tag{3}$$

By choosing new $c' = C = c^2$, we will have $f(n)^2 = O(g(n)^2)$.

1.4 Which of the following statements are true?

1.4.1 If f, g, and h are positive increasing functions with f in $O(h)$ and g in $\Omega(h)$, then the function $f + g$ must be in $\Theta(h)$.

False. We let $f(n) = \log n, g(n) = n^2, h(n) = n$, then $f + g = n^2 + \log n = \Omega(h(n))$, but not $\Theta(h(n))$.

1.4.2 Given a problem with input of size n, a solution with $O(n)$ time complexity always costs less in computing time than a solution with $O(n^2)$ time complexity.

False. $O(n), O(n^2)$ are just the upper bound of the worst situation complexities. The real operation numbers and running time are uncertain, and there is possibility that $O(n^2)$ program runs quicker than $O(n)$. We can only say that $O(n)$ costs less when n is really big.

1.4.3 $F(n) = 4n + \sqrt{3n}$ is both $O(n)$ and $\Theta(n)$.

True. Since $n^2 > 3n$ for all $n > 3$, then $4n + \sqrt{3n} \leq 4n + n = 5n$. Therefore $F(n) = O(n)$. Apparently $F(n) > 4n$, then $F(n) = Theta(n)$.

1.4.4 For a search starting at node s in graph G, the DFS Tree is never as the same as the BFS tree.

False. For a graph G takes the shape of a straight line, and by choosing starting point s as one of the end point, DFS tree is as the same as BFS tree.

1.4.5 BFS can be used to find the shortest path between any two nodes in a non-weighted graph.

False, if the graph is not connected, and two points are located separately, BFS can not find the shortest path.

1.5 Kleinberg and Tardos, Chapter 3, Exercise 2.

Algorithm 1 Undirected Graph Cycle Detection Algorithm

Input: The start vertex, the visited set, and the parent node of the vertex.
Output: True a cycle is found.

```
function isCyclicUntil(start point $v$, array $visited$, parent node $parent$)
    visited[v]=true;
    for all adjacent point $u$ of $v$ do
        if $u$ is not visited and isCyclicUntil(u, visited, v) then
            return $true$
        else
            if $u$ is not the parent node of current node then
                return $true$
            end if
        end if
    end for
end function

function isCyclic(Graph $G$)
    Initialize array $visited$ as false
    for point $v$ in the Graph do
        if $v$ is not visited and isCyclicUntil(v, visited, -1) then
            return $true$
        end if
    end for
    return $false$
end function
```

We use DFS algorithm to detect Cycle in a graph.

Detect Cycle in a undirected Graph: For every visited vertex u, when we have found any adjacent vertex v, such that v is already visited, and v is not the parent of vertex u. Then one cycle is detected.

2 Practice Problems

2.1 Kleinberg and Tardos, Chapter 2, Exercise 6.

2.1.1

The outer loop takes n iterations, and the inner loop takes at most $n-1$ loops. In each iteration, the sum takes $j-i+1$ operations. Therefore the complexity of this function is:

$$O(f(n)) = O(n^3) \tag{4}$$

2.1.2

For $i < n/4$ and $j > 3n/4$, the sum step takes $j-i+1 > n/2$ operations. Since there are $n/4$ such pairs of (i, j), the complexity is at least:

$$\frac{n}{2} \cdot (\frac{n}{4})^2 = \frac{n^3}{32} \tag{5}$$

Therefore it's also $\Omega(n^3)$.

2.1.3

Algorithm 2 Quick-sum Algorithm

```
for i=1,2,3,...,n-1 do
    B[i,i+1]=A[i]+A[i+1];
end for
for i=1,2,3,...,n-2 do
    for k=2,3,...,n-i do
        j=i+k;
        B[i,j]= B[i, j-1]+A[j];
    end for
end for
```

The complexity is $O(n^2)$.

2.2 Kleinberg and Tardos, Chapter 3, Exercise 6.

Suppose that G contains a edge e that does not belong in BFS tree (We denote as T_1). Since all nodes on each levels of BFS tree have the same number of edges going from the root u, therefore the edge e must be connected to two nodes on the same level, otherwise, it will be included in the BFS tree. However, if we operate DFS on the same graph G, it will include e in its tree because it always goes deeper into the graph. Eventually making the DFS tree differs from the BFS tree. According to the topic, BFS tree and DFS tree are identical, therefore such edge e does not exists. $G = T$.