CS-521-900, Final

Yiyun Zhang

August 30, 2020

1. **a)** Let the proportional be a, each node has b children, then the running time is $T(n) = a * T(n/b) + cn$. Therefore if $a < b$, the time complexity is $O(n)$, if $a = b$, the time complexity is $O(n \log n)$, if $a > b$, the time complexity is $O(n^{\log_b a})$,

   b) A topological sort is useful for *make*. Each vertice is a file, the vertices are connected to each other. For every directed edge [u,v], u comes before v in the ordering.

   c)

```
Sort(A) {
  d = the largest number's digits
  for i = 0, i<d, i++:
    digitSort(A,i)
}
digitSort(A,d) {
  max = the largest element in current place
  count = 0, countArray = []
  for i = 0, i < A.size, j++:
    for each unique digit in the elements in curent place, count++
    countArray[i] = count
  for j = 1, j < max, j++:
    sum = cumulative sum
    countArrcy = sum
  for i = size, i>1, i--:
    A[i] = countArray[i]
    count--
 }
 Find(A) {
   For i = 1, i < A.size, i++:
     if(A[i]==A[i-1]):
       return A[i]
 }
```

2. Base case: When n = 0, there is no node, therefore it is true.
   Assume a red-black tree T with n key-bearing nodes has at least n/3 black

nodes:
For a red-black tree Q with n+1 key-bearing nodes: if the extra node is black, then it has at least n/3+1 black nodes, which means at least n/3 black nodes. If the extra node is red, it must have two black children, therefore the tree Q at least has n/3+2 black nodes, which is at least n/3 black nodes. Therefore the statement is true.

3.
```
count = 1, result = []
for i = 0, i < A.size, i++:
  if A[i+1] < A[i]:
    find the next minimum number
  else
    find the next maxmum number
  if(minimum number found):
    counter++
    result.add(min)
  else if(maxmum number found)
    counter++
    list.add(max)
  return result.size
```

Loop invariant: The first number in the result list is always larger than the next element.
Initialization: Before the loop starts, the list is empty, so it is true.
Maintenance: For each loop, $result[0]$ is always larger then result[1] because A[i] and A[i+1] is added to the end of the list, where $result[0]$ and $result[1]$ is the first two elements of the current longest osciallating subsequence, which by definitation $X[i] > X[i+1]$ for all odd i, it is also true for A[i+1], therefore it is true.
Termination: When the loop ends, the result list has the longest oscial-lating sub-sqeuence, which by definitation $X[i] > X[i+1]$ for all odd i, therefore it is true.
The running time is $O(n^2)$

4.
```
w = weight of [u,v]
findMax(T,w) {
  for u, v in V
    max[u,v] = null
  x = null
  recFunction(x,u)
  while x != null
    y = recFunction(x)
    for v in T's adjacent element y
      if max[u,v] == null and u != v
        if y == u or weight of [y,v] > max[u,y]
          max[u,v] = (y, v)
        else
```

```
            max[u,v] = max[u,y]
          recFunction(x,v)
return max[u,v]
```

If u and v are adjacent, the max weight edge is [u,v], so the running time is constant. Otherwise, the algorithm is $O(|V|^2)$ since