Mark Abraham

CS 3510 2d

(2.5) substitution $T(n) = aT(n-b) + O(n^d)$

(i) $T(n) = T(n-1) + c^n$, $c > 1$

substitute $n \longrightarrow n-1$

$k=0$ $T(n) = T(n-1) + c^n$

$T(n-1) = T(n-1-1) + c^n$
$= T(n-2) + c^n$

$k=1$ $T(n) = [T(n-2) + c^n] + c^{n-1}$

$T(n-2) = T(n-3) + c^n$

$k=2$ $T(n) = [T(n-3) + c^n] + c^{n-1} + c^{n-2}$

$n - k_{max} - 1 = 0$

$k_{max} = n-1$

k_{max} $T(n) = \cancel{T(0)} + \sum_{i=1}^{n} c^i$ $\theta(c^n)$

(j)

$T(n) = 2T(n-1) + 1$

$n \longrightarrow n-1$

$k=0$ $T(n) = 2\boxed{T(n-1)} + 1$

$T(n-1) = 2T(n-2) + 1$

$k=1$ $T(n) = 2[2T(n-2) + 1] + 1$
$= 2^2 T(n-2) + 2 + 1$

$T(n-2) = 2T(n-3) + 1$

$k=2$ $T(n) = 2^2 [2T(n-3) + 1] + 2 + 1$
$2^3 T(n-3) + 2^2 + 2 + 1$

$n - k_{max} - 1 = 0$

$k_{max} = n-1$

$k = n-1$ $T(n) = 2^{k+1} T(n-k-1) + \sum_{i=0}^{k} 2^i$

k_{max} $T(n) = \cancel{2^n T(0)} + \sum_{i=0}^{n-1} 2^i = \theta(2^{n-1})$

2.19

(a) Analyze the complexity

```
function mergesort(a[1...n])
input: An array of numbers a[1...n]
output: A sorted version of this array

if n > 1:
   return merge(mergesort(a[1...⌈N/2⌉]), mergesort(a[⌊N/2⌋+1...n]))
else:
   return a

function merge(x[1...k], y[1...L])
if k = 0: return y[1...L]
if L = 0: return x[1...k]
if x[1] ≤ y[1]:
      return x[1] ∘ merge(x[2...k], y[1...L])
else:
   return y[1] ∘ merge(x[1...k], y[2...L])
```

running time $O(k+1)$

recurrence relation: $T(n) = 2T(n/2) + O(n)$

or

$O(n \log n)$

for i in range k

k[0] = merge(k[0] + k[i+1])

$O(k^2 n)$

(b) similar to merge sort, we could recurse on the number arrays in k until there is a pair and then combine these two with merge and then combine that merged pair with the pair next to it that has just been merged. The time complexity would be similar to merge sort thus $O(k \log k n)$