Assignement 1, COSC 3P03, Algorithms, Winter, 2016

Due: 5:00 pm, Jan. 28, Thursday.

1. (10) Modify the mergesort as follows: instead of dividing the input list into two lists, the list is divided into three sublists of (roughly) the same size. (a) Write down the recursive algorithm; (b) Let $t(n)$ be the running time of your algorithm, write a recurrence for $t(n)$; (c) What is the running time and how does it compare to the traditional mergesort? Note that you need to solve for $t(n)$ first. Note: if you want to merge two lists L_1 and L_2, you can just say in your algorithm "Merge L_1 and L_2."

   ```
   if the list has 3 or more elements
        sort the first 1/3 of the list into L1;
        sort the second 1/3 of the list into L2;
        sort the third 1/3 of the list into L3;
        merge L1 with L2 into L12;
        merge L12 with L3 into L123
   else sort the list
   ```

   Let $t(n)$ be the running time for the algorithm, we have

   $$\begin{aligned} t(n) &= 3t(n/3) + cn \\ t(2) &= c' \end{aligned}$$

   with a solution of $t(n) = O(n \log)$ (masters method), asymptotically the same as that for the traditional mergesort.

2. (5) Let $A[1..n]$ be a sorted array of distinct integers, some of which may be negative. Give a recursive $O(\log n)$ algorithm that can find an index i such that $1 \leq i \leq n$ and $A[i] = i$, provided that such an index exists. This means that you need to give the algorithm first and then analyze it.

   We use a binary-search like algorithm (this algorithm checks a segment of the array $A[a..b]$). We check whether for index $middle$ $A[middle] = middle$. If yes, it returns yes, else, depending on whether $A[middle] < middle$, we go to the right half or left half. For example, if $A[10] < 10$, then we know $A[9] < 9$, $A[8] < 8$, ... so we have to go to the right half if such an i exists. The algorithm is given below:

   ```
   check(A, a, b)
          if a = b
                 if A[a] = a return yes else no
          else
                 middle = ceiling((a+b)/2)
                 if A[middle] = middle return yes
                 else if A[middle] < middle then return (check(middle+1, b))
                      else return (check(a, middle-1))
   end
   ```

   This algorithm is similar to the binary search with a running time $t(n) = t(n/2) + c$ and therefore has a running time of $O(\log n)$, which is $o(n)$.

3. (15) Consider the following three algorithms for the problem of computing x^n, $x \neq 0$ and $n \geq 0$. For each algorithm, find its asymptotic running time. Show your work, which means if the algorithm is recursive, you need to give its recurrence (don't forget the initial condition) and solve it in whatever way.

```
A1(x, n)
 r = 1
 for i=1 to n                                   t(n) = O(n)
        r = r × x
 return r
```

```
A2(x, n)
if n = 0
      return 1                                  t(n) = t(n − 1) + c = O(n)
else
      return (x × A2(x, n − 1))
```

```
A3(x, n)
if n = 0
      return 1                                  t(n) = t(n/2) + c = O(log n)
else
      t = A3(x, ⌊n/2⌋)
      if n is even
          return t × t
      else
          return t × t × x
```

4. (10) List the functions below from lowest order to highest order, where $k = 3/2$ and $c = \log 3$ are constants. If any two or more are of the same asymptotic order, group them together.

$$\log^k n,\ \log n^k,\ n^k,\ k^n,\ 2^n,\ 2^{cn},\ 2^{n+1},\ n^{\log k},\ k^{\log n},\ \log_3^5 n,\ \log_{100}^{500} n,\ \log(n!),\ \log(n^n),$$
$$100n + \log n,\ n + (\log n)^2,\ \log n,\ \log(n^2),\ n^2/\log n,\ n\log^2 n,\ (\log n)^{\log n},\ n/\log n,\ n^{1/2},\ (\log n)^5,\ n2^n,\ 3^n.$$

$\log n, \log(n^2), log n^k$
$\log^{3/2} n$
$\log_3^5 n, (\log n)^5$
$\log_{100}^{500} n$
$n^{1/2}$
$n^{\log k}, k^{\log n}$
$n/\log n$
$100n + \log n, n + (\log n)^2$
$\log(n!), \log(n^n)$
$n\log^2 n$
$n^{3/2}$
$n^2/\log n$
$(\log n)^{\log n}$
k^n
$2^n, 2^{n+1}$
$n2^n$
$3^n, 2^{cn}$

5. (15) Use as many methods as possible to solve the following recurrence:

$$t(n) = 2t(n/2) + 1.$$

Full credits if you solved it with 3 different methods. Assume that $n = 2^k$ for some k.

Iteration

$$\begin{aligned} t(n) &= 2t(n/2) + 1 \\ &= 2(2t(n/2^2) + 1) + 1 \\ &= 2^2 t(n/2^2) + 2 + 1 \\ &= 2^3 t(n/2^3) + 2^2 + 2 + 1 \\ \vdots &= 2^k t(1) + 2^{k-1} + 2^{k-2} + \cdots + 2 + 1 \\ &= 2^k + (2^k - 1) \\ &= 2^{k+1} - 1 \\ &= O(2^k) \\ &= O(n). \end{aligned}$$

The master method: $a = b = 2$, $f(n) = 1$. Case 1, $t(n) = O(n^{\log_b a}) = O(n)$.

Recursion tree:

The tree is a complete binary tree with total work $(1 + 2 + 2^2 + \cdots 2^k) = (2^{k+1} - 1)/(2 - 1) = O(2^k)$.

Substitution:

Guess $t(n) = O(n)$. Assume $t(n/2) \leq C(n/2)$, need to show $t(n) \leq Cn$.

$$\begin{aligned} t(n) &= 2t(n/2) + 1 \\ &\leq 2C(n/2) + 1 \\ &= Cn + 1. \end{aligned}$$

We now get stuck since $Cn + 1 > Cn$. This situation is discussed in the Cormen et al's book and can be resolved by guessing that $t(n) \leq Cn - b$ for some constant b:

$$\begin{aligned} t(n) &= 2t(n/2) + 1 \\ &\leq 2(C(n/2) - b) + 1 \\ &= Cn - 2b + 1 \\ &\leq Cn - b. \end{aligned}$$

for $b \geq 1$.

6. (20) Solve the following using whatever methods.

a. $t(n) = 2t(n/2) + n^3$; $O(n^3)$

b. $t(n) = t(n/10) + n$; $O(n)$

c. $t(n) = 16t(n/4) + n^2$; $O(n^2 \log n)$

d. $t(n) = t(n-1) + \log n$. $O(n \log n)$

7. (15) Solve the following recurrences by the masters method, if possible. In case(s) where the method does not apply, state why and solve it/them by other means.

$$\begin{aligned} t(n) &= 9t(n/3) + n \\ t(n) &= t(2n/3) + 1 \\ t(n) &= 3t(n/4) + n \log n \end{aligned}$$

(a) $n^{\log_b a} = n^{\log_3 9} = n^2$ and $f(n) = n = O(n^{2-\epsilon})$ for some $\epsilon > 0$, Case 1, $t(n) = O(n^2)$.

(b) $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$ and $1 = \theta(1)$, Case 2, $t(n) = O(\log n)$.

(c) $n^{\log_b a} = n^{\log_4 3} = n^{0.793}$ (0.793 not important as long as we know it's less than 1). $f(n) = n \log n = \Omega(n^{0.793+\epsilon})$ for some $\epsilon > 0$ so this could be Case 3. Need to verify one more condition. whether $af(n/b) \leq cf(n)$ for some constant $c < 1$:

$$\begin{aligned} af(n/b) &= 3f(n/4) \\ &= 3(n/4)\log(n/4) \\ &= (3/4)n\log(n/4) \\ &= (3/4)n(\log n - 2) \\ &< (3/4)n\log n. \end{aligned}$$

where $c = 3/4$. So it is Case 3 and $t(n) = \theta(n \log n)$.

8. (10) What is the Big Oh function for each of the following functions of n (it should be as tight and as simple as possible). There is no need to justify your answers.

a. $t(n) = (n \log n)/2 + f(n)$, where $f(n) = o(n \log(n^{100}))$: $O(n \log n)$

b. $t(n) = 1 + 3 + 5 + ... + (n-1)$, where n is even: $O(n^2)$

c. $t(n) = \log^2 n + \log(n^3)$: $O(\log^2 n)$

d. $t(n) = 4^{\log n}$: $O(n^2)$

e. $t(n) = 1 + 1/2 + 1/2^2 + ... + 1/2^n$: $O(1)$

f. $t(n) = n^1 + n^2 + n^3 + ... + n^k + 2^n$, where $k > 0$ is an integer: $O(2^n)$

g. $t(n) = 4n^{3/4} + 5n \log n + 2n \log \log n$: $O(n \log n)$

h. $t(n) = 2016 + \sin(n)$: $O(1)$

i. $t(n) = 2t(n-1) + 1$, $t(1) = 1$ (this is the number of moves for the Tower of Hanoi): $= 2^n - 1 = O(2^n)$

j. $t(n) = (n^2 - 1)/(n + 1)$: $O(n)$