

# CS344: Design and Analysis of Computer Algorithms

## Homework 3

Group Members: Stephen Kuo, Derek Mui

2.4) Suppose you are choosing between the following three algorithms:

- Algorithm  $A$  solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm  $B$  solves problems of size  $n$  by recursively solving two subproblems of size  $n - 1$  and then combining the solutions in constant time.
- Algorithm  $C$  solves problems of size  $n$  by dividing them into nine subproblems of size  $n/3$ , recursively solving each subproblem, and then combining the solutions in  $O(n^2)$  time.

**Answer:**

- $T(n) = 5 * T(n/2) + cn$ .  
Applying master theorem  $a = 2, b = 5, f(n) = cn$ ,  $\text{degree}(f(n)) = 1$   
Since  $\log_2 5 > 1, T(n) = O(n^{\log_a b}) = O(n^{\log_2 5})$
- $T(n) = 2T(n - 1) + c. \therefore T(n) = O(2^n)$
- $T(n) = 9T(\frac{n}{3}) + cn^2$ .  
Applying master theorem  $a = 3, b = 9, f(n) = cn^2$ ,  $\text{degree}(f(n)) = 2$   
Since  $\log_3 9 = 2, T(n) = O(n^2 \log n)$

Time complexity of the third algorithm is the best.  $\therefore$  choose algorithm C

2.5) Solve the following recurrence relations and give a  $\Theta$  bound for each of them.

**Answer:**

- (a)  $T(n) = 2T(n/3) + 1$   
 $(n_i A = a_i \dots a_n, B - B_i \dots b_n)$   
 $a = 2, b = 3, d = f(n) = O(1) = 0$   
 $d > < \log_b a = 0 < \log_3 2$   
 $\therefore O(n^{\log_3 2})$

$$\begin{aligned}
\text{(b)} \quad & T(n) = 5T(n/4) + n \\
& a = 5, b = 4, d = f(n) = O(n) = 1 \\
& d > \log_b a = 1 < \log_4 5 \\
& \therefore O(n^{\log_4 5})
\end{aligned}$$

$$\begin{aligned}
\text{(c)} \quad & T(n) = 7T(n/7) + n \\
& a = 7, b = 7, d = 1 \\
& d > \log_b a = 1 < \log_7 7 = 1 < 1 \\
& \therefore O(n \log n)
\end{aligned}$$

$$\begin{aligned}
\text{(d)} \quad & T(n) = 9T(n/3) + n^2 \\
& a = 9, b = 3, d = f(n) = O(n^2) = 2 \\
& d > \log_b a = 2 \log_3 9 = \frac{\log 9}{\log 3} \\
& \therefore O(n^2 \log n)
\end{aligned}$$

2.14) You are given an array of  $n$  elements, and you notice that some of the elements are duplicates; that is, they appear more than once in the array. Show how to remove all duplicates from the array in time  $O(n \log n)$

**Answer:** Simply sort the elements of the array using merge sort in  $O(n \log n)$  time and then remove the duplicate elements by traversing the sorted array.

2.25)

**Answer:**

2.28) The *Hadamard matrices*  $H_0, H_1, H_2, \dots$  are defined as follows:

- $H_0$  is the  $1 \times 1$  matrix  $[1]$
- for  $k > 0$ ,  $H_k$  is the  $2^k \times 2^k$  matrix

$$H_k = \begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{bmatrix}$$

Show that if  $v$  is a column vector of length  $n = 2^k$ , then the matrix-vector product  $H_k v$  can be calculated using  $O(n \log n)$  operations. Assume that all the numbers involved are small enough that basic arithmetic operations like addition and multiplication take unit time.

**Answer:** For any column vector  $u$  of length  $n$ , let  $u_1$  denote the column vector of length  $n/2$  consisting of the first  $n/2$  coordinates of  $u$ . Similarly, let  $u_2$  be

the vector of the remaining coordinates. Note that:

$$(H_k v)_1 = H_{k-1} v_1 + H_{k-1} v_2 = H_{k-1}(v_1 + v_2)$$

and

$$(H_k v)_2 = H_{k-1} v_1 - H_{k-1} v_2 = H_{k-1}(v_1 - v_2)$$

Recursion 1: Shows that we can find  $H_k v$  by calculating  $v_1 + v_2$  and  $v_1 - v_2$  and recursively computing  $H_{k-1}(v_1 + v_2)$  and  $H_{k-1}(v_1 - v_2)$

Recursion 2: Only need to compute two subproblems,  $H_{k-1} v_1$  and  $H_{k-1} v_2$

Lastly, combining the solutions of the two subproblems using addition and subtraction, both taking  $O(n)$  time.

3.5) The *reverse* of a directed graph  $G = (V, E)$  is another directed graph  $G^R = (V, E^R)$  on the same vertex set, but with all edges reversed; that is,  $E^R = \{(v, u) : (u, v) \in E\}$

Give a linear-time algorithm for computing the reverse of a graph in adjacency list format.

**Answer:**

```

1 for each edge  $(v, u) \in E$ 
2   do reverse  $(v, u)$  to get  $(u, v)$ 
```

The algorithm runs in time  $O(n)$  because it does a constant amount of work for each of the  $O(n)$  edges. We simply look at each element of the edge list in our adjacency list representation and swap the vertices.