Jonathan Tso
Homework 3
CS 401

1. Algorithm A $T(n) = T(n-1) + n$

$n + (n-1) + (n-2) + (n-3) + \ldots$ n times

$= \Theta(n^2)$

Algorithm B $T(n) = 6T(\frac{n}{4}) + n$

$\log_4 6 = 1.29 \quad K = 1$

$= \Theta(n^{\log_4 6})$

Algorithm C $T(n) = 4T(\frac{n}{4}) + n$

$\log_4 4 = 1 \quad K = 1 \quad P = 0$

$= \Theta(n \log n)$

Algorithm D $T(n) = 2T(\frac{n}{3}) + n^2$

$\log_3 2 = 0.63 \quad K = 2$

$2(n/3)^2 \leq cn^2 \quad c = \frac{2}{9}$ (Regularity Condition)

$= \Theta(n^2)$

I would choose to use algorithm C

2. Algorithm A = $T(n) = 7T(\frac{n}{2}) + n^2$

B = $T(n) = aT(\frac{n}{4}) + n^2$

A: $\log_2 7 > 2 \quad K = 2 \quad \log_2 7 \approx 2.8$

$\Theta(n^{\log_2 7})$

1. Algorithm A

$n + (n-1) + (n-2) + (n-3) + \ldots$ n times

$= \Theta(n^2)$

Algorithm B $T(n) = 6T(\frac{n}{4}) + n$

$\log_4 6 = 1.29$ $K = 1$

$= \Theta(n^{\log_4 6})$

Algorithm C $T(n) = 4T(\frac{n}{4}) + n$

$\log_4 4 = 1$ $K = 1$ $P = 0$

$= \Theta(n \log n)$

Algorithm D $T(n) = 2T(\frac{n}{3}) + n^2$

$\log_3 2 = 0.63$ $K = 2$

$2(n/3)^2 \leq cn^2$ $c = \frac{2}{9}$ (Regularity Condition)

$= \Theta(n^2)$

I would choose to use algorithm C

2. Algorithm A : $T(n) = 7T(\frac{n}{2}) + n^2$

B : $T(n) = aT(\frac{n}{4}) + n^2$

A: $\log_2 7 > 1$ $K = 2$ $\log_2 7 \approx 2.8$

$\Theta(n^{\log_2 7})$

B: $\log_4 a$ $K = 2$

if $a > 16$, then we have $\log_4 17 \approx 2.04$

$\Theta(n^{\log_4 17})$ This is smaller until a is about 50

Then, $\log_4 50 > 2.8$, so minimum $a \geq 50$

4. b) Arbitrarily choose 2 random items in the array and do a comparison (constant time). If they are

3. a) $T(n) = 8T(\frac{n}{2}) + n^3$ $\log_2 8 = 3$ $K=3$

case 2 $\rightarrow$ $\Theta(n^3 \log n)$

b) $T(n) = 5T(\frac{n}{4}) + n$ $\log_4 5 > 1$ $K=1$

case 1 $\rightarrow$ $\Theta(n^{\log_4 5})$

c) $T(n) = 7T(\frac{n}{7}) + n$ $\log_7 7 = 1$ $K=1$

case 2 $\rightarrow$ $\Theta(n \log n)$

d) $T(n) = 49T(\frac{n}{25}) + n^{3/2} \log n$ $\log_{25} 49 \approx 1.21$ $K=3/2$ $P=1$

$49(\frac{n}{25})^{3/2} \log(\frac{n}{25}) < cn^{3/2} \log n$

looking at the non-log

$49/125 < 1$

case 3 $\rightarrow$ $\Theta(n^{3/2} \log n)$

e) $T(n) = T(n-1) + n^c$ c = some constant, greater than 1

$n^c + (n-1)^c + (n-2)^c + \ldots$ n times

$n^{2c} \approx \Theta(n^c)$

4. a) 1. Do a total count of all items in the array (at the beginning, find

2. Check if the majority element of the prior is the majority element

(that is, when we run this recursively, we take the majority of

the smaller set).

3. a) $T(n) = 8T(\frac{n}{2}) + n^3$ $\log_2 8 = 3$ $k=3$

case 2 $\rightarrow$ $\Theta(n^3 \log n)$

b) $T(n) = 5T(\frac{n}{4}) + n$ $\log_4 5 > 1$ $k=1$

case 1 $\rightarrow$ $\Theta(n^{\log_4 5})$

c) $T(n) = 7T(\frac{n}{7}) + n$ $\log_7 7 = 1$ $k=1$

case 2 $\rightarrow$ $\Theta(n \log n)$

d) $T(n) = 49T(\frac{n}{25}) + n^{\frac{3}{2}} \log n$ $\log_{25} 49 \approx 1.21$ $k=3/2$ $p=1$

$49(\frac{n}{25})^{3/2} \log(\frac{n}{25}) < cn^{3/2} \log n$

looking at the non-log

$49/125 < 1$

case 3 $\rightarrow$ $\Theta(n^{3/2} \log n)$

e) $T(n) = T(n-1) + n^c$ c = some constant, greater than 1

$n^c + (n-1)^c + (n-2)^c + \ldots$ n times

$n^{2c} \approx \Theta(n^c)$

4. a) 1. Do a ~~total~~ count of all items in the array (at the beginning, find n)

2. Check if the majority element of the prior is the majority element here (that is, when we run this recursively, we take the majority element from the smaller set).

Here, if the majority is no longer majority, we must check the other half. If it is not majority in either, we know there is no majority. If there is a majority in either half, we must check either or both values when we do return.

The runtime here will be $T(n) = 2T(\frac{n}{2}) + n$, which resolves to $\Theta(n \log n)$. This is because we split the case into two each time and look for the majority. Because we pulled the element from the prior, we are only checking at most 2 elements to the n elements at each step.

4. b) Arbitrarily choose 2 random items in the array and do a comparison (constant time). If they are equal, retain one of them in a new array. If they are not, destroy both items. Keep count of the match for each item, +2. If this ever exceeds half of n, it is majority. How this works is that each time a match occurs, we keep the specific item to continue to check if not, we remove it because we are looking for matches on the majority of the items. In the end, we count the total match values and any exceeding $\frac{n}{2}$ is majority. This runs in $O(n)$ because we always at most have $\frac{n}{2}$ after each arbitrary match, since we either destroy $1/2$ of the match, or the whole unmatch. Thus, we have $T(n) = T(\frac{n}{2}) + n/2$ where $f(n) = \frac{n}{2}$ is when we have to do the equality checks.

5. We will still implement merge sort here. However, because we don't know if the value of right hand side < 2 left hand side when doing comparisons, we cannot just merge after identifying an inversion during the merge sort. Therefore, the algorithm will be the same, except that we do the entire merge at the end. This will still remain $O(n \log n)$ because we still only do single comparison scans on the left and right only once.