Sean
Blanchard
COMP354

HOMEWORK 2.

SORT IN INCREASING ORDER

2-1) a) GROUP 1:

$f_1(n) = n^{0.9999999} \log n$

$f_2(n) = 10000000n$

$f_3(n) = 1.000001^n$

$f_4(n) = n^2$

ORDER:
1) $f_1(n)$
2) $f_2(n)$
3) $f_4(n)$
4) $f_3(n)$

REASON: Apply log to all 4

$f_1(n) = \log(n^{0.9999999} \log(n)) = 0.9999999 \log n + \log \cdot \log(n)$

$f_2(n) = \log(1000000n) = \log(1000000) + \log(n)$

$f_3(n) = \log(1.00000001^n) = n \log(1.0000001)$

$f_4(n) = \log(n^2) = 2 \log n$

Here it shows $f2 > f1$, and $f_4 > f_2$ as time compexity is greater than linear time compler.

b) GROUP 2:

$f_1(n) = 2^{2^{1000000}}$

$f_2(n) = 2^{100000n}$

$f_3(n) = \binom{n}{2}$ OR $n(n-1)/2$

$f_4(n) = n\sqrt{n}$ OR $n^{1.5}$

ORDER:
$f_1(n)$
$f_4(n)$
$f_3(n)$
$f_2(n)$

Reason:
$f_1(n)$ is a constant, Thus it is clearly smaller than $f_4(n)$. $f_4(n) = O(n^2) \cdot O(f_3(n))$.
$f_2(n)$ is eponential, and $f_3(n)$ is quadratic.

c) $f_1(n) = n^{\sqrt{n}}$

$f_2(n) = 2^n$

$f_3(n) = n^{10} \cdot 2^{n/2}$

$f_4(n) = \sum_{i=1}^{n} (i+1)$

ORDER:	Reasoni
1) $f_4(n)$	$f_1(n) = \sqrt{n} \log n$
2) $f_1(n)$	$f_2(n) =$ linear function of n $>$
3) $f_3(n)$	$f_3(n) = 2^{\log(n^{10})} \cdot 2^{n/2} = 2^{n/2 + 10\log(n)}$
4) $f_2(n)$	$f_4(n) = \sum_{i=1}^{n} (i+1) = \frac{n([n+1]+2)}{2} = \frac{n(n+3)}{2} = \Theta(n^2)$

2-2) a) Recurrence Relation Resolution

$T(x,c) = \Theta(x)$ for $c \leq 2$

$T(c,y) = \Theta(y)$ for $(c \leq 2)$

$T(x,y) = \Theta(x+y) + T(x/2, y/2)$

$T(x,y) = \Theta(x+y) + T(x/2, y/2)$

Form a geometric series

$T(x,y) = \Theta(x+y) + \Theta\left(\frac{x+y}{2}\right) + \Theta\left(\left(\frac{x+y}{4}\right)\right) + \Theta\left(\frac{x+y}{8}\right)$

$T(x,y) = \Theta((x+y) + (x+y/2) + \left(\frac{x+y}{4}\right) + \left(\frac{x+y}{8}\right) \ldots$

$T(x,y)$ is upper bounded by $\boxed{\Theta(x+y)}$ as sum of series $2(x+y)$. Lower bound $(x+y)$

The answer is $\boxed{\Theta(n)}$.

b) $T(x, c) = \Theta(x)$ for $c \leq 2$
$T(c, y) = \Theta(y)$ for $c \leq 2$
$T(x, y) = \Theta(x) + T(x, y/2)$

$T(x, y) = \Theta(x) + T(x, y/2)$
we can replace $T(x, y/2)$ w/ recursive formula:
$$T(x,y) = \underbrace{\Theta(x) + \Theta(x) + \Theta(x) + \cdots + \Theta(x)}_{\Theta(\log y)}$$

As shown above, $T(x,y)$ is $\Theta(x \log y)$. substitute n for x, y: $T(n,n) = \Theta(n \log n)$.

c) $T(x, c) = \Theta(x)$ for $c \leq 2$
$T(x, y) = \Theta(x) + S(x, y/2)$
$S(c, y) = \Theta(y)$ for $c \leq 2$
$S(x, y) = \Theta(y) + T(x/2, y)$.

Thus...
$T(x, y) = \Theta(x) + \Theta(y/2) + T(x/2, y/2)$
$T(x, y) = \Theta(x+y) + T(x/2, y/2)$

Thus is like part a) proven above,
The correct answer is $\Theta(n)$.

2-3 Peak finding correctness

a) Is algorithm 1 correct?

Yes, I ran 10+ test runs and determined that it found the peak correctly every time. This reminds me of cartesian product of two lists.

It is $O(n)$ because it takes less than or equal ~~to max_dimensions~~ iterations

This gives $\boxed{n + n/2 + n/4 + \ldots}$

b) Is algorithm 2 correct?

Yes, This algorithm will always return a location, because its value increases each call, and the values we are giving will always end in the matrix. Tested ☑

c) Is algorithm 3 correct?

NO! I was able to find a bug quickly by plugging in random numbers into the matrix!!! fails.

d) Is algorithm 4 correct?

Yes! Ran multiple matrixes. It works because Splits array into 3 parts and compares the central with the two neighbors.

2-4 Peak Finding Efficiency

a) What is worst case runtime of Algorithm1 (nxn)

$\Theta(n \log n)$

b) What is worst case runtime of Algorithm2 (nxn)

$\Theta(n^2)$

c) worst case runtime of algorithm3 (nxn)

$\Theta(n)$ - (Recurrence Relation) $\rightarrow T(m,n) = \Theta(m+n) + T(m/2, n/2)$

d) worst case runtime of algorithm 4

$\Theta(n)$ - alternates between splitting

2-5 Peak - Finding Proof

Given: proof of correctness for algorithm 1.
we are going to show that algorithm 4 is the most efficient correct algorithms out of the three.
We know that for cases where the maximum element is in the center column, if it is not, we step from max to increase elements that don't sit in the center row, this shows a peak will exist in the corresponding half. The algorithm will always recurse into non-empty subsets, thus it must return a location.

2b Peak finding Counterexamples

algorithm3:

[0, 0, 2, 6, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[0, 5, 0, 0, 0, 0, 0],
[0, 2, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0], ERROR!

2b Peak finding Counterexamples

algorithm3:

[0, 0, 2, 6, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[0, 5, 0, 0, 0, 0, 0],
[0, 2, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0],   ERROR!