Dan Ports
6.046 PS 1-1

R8
No collabs

In order of ascending order of growth (functions on the same line are Θ of each other)

$(2/3)^n$

$1, 100^{100}, (1/n)^{1/\lg n}$

$\lg(\lg^* n)$

$\lg^*(\lg n)$

$2^{\lg^*(n)}$

$\ln \ln n$

$\sqrt{\lg n}$

$\ln n$

$n, \lfloor n \rfloor, 2^{\lg n}$

$n \lg n$

n^2

$(3/2)^n$ $\leftarrow n^2 + 2^{100} n, \sum_{k=1}^{n} k$

2^n

$n 2^n$

e^n

2^{2n}

$(\lg n)!$

$(\lg n)^{(\lg n)}$ X

$n^{2 \lg \lg n}$

$n^{\log_8 n}$

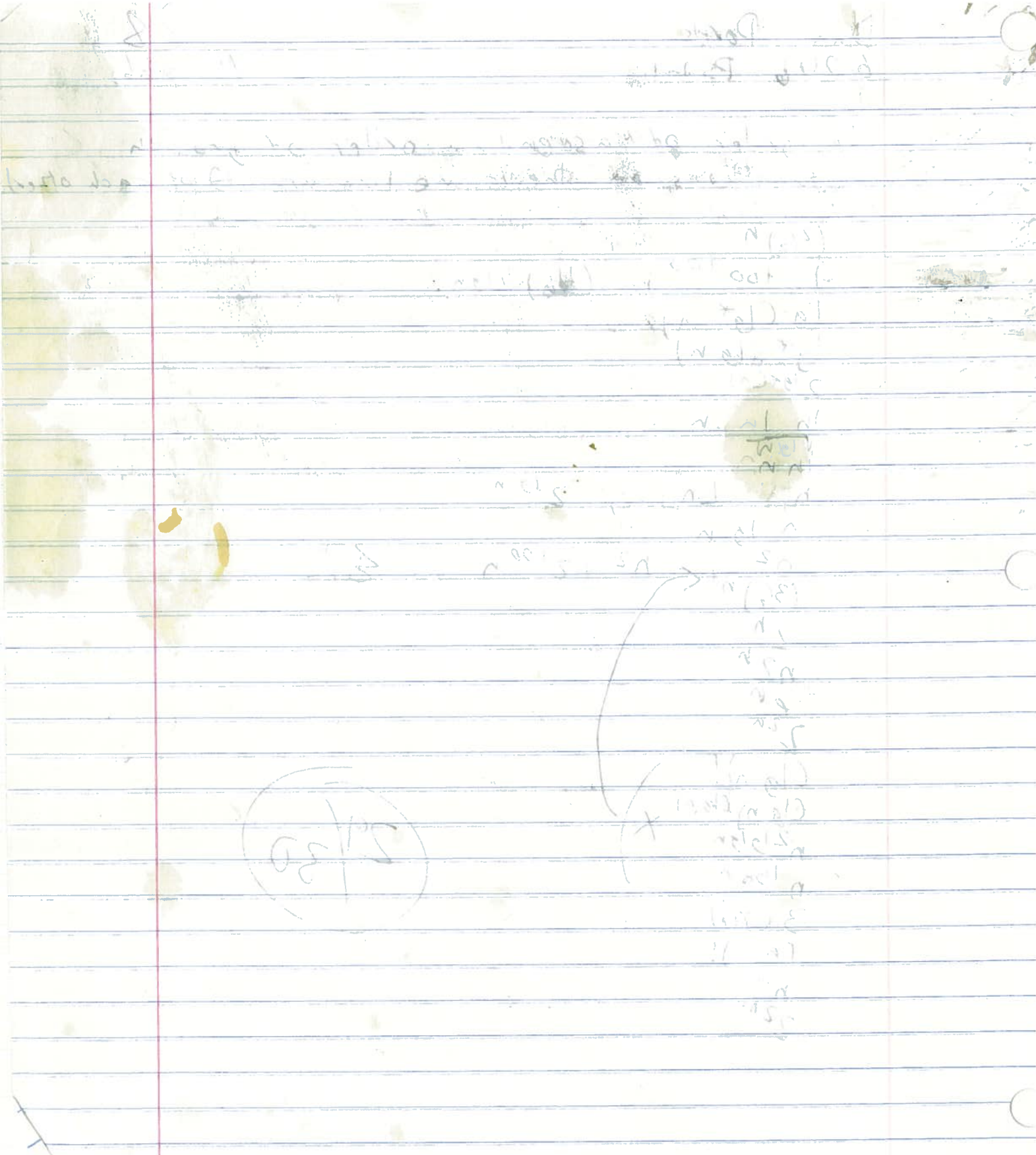
$3(n!)$

$(n+1)!$

n^n

$2^{2^{n+1}}$

29/30



Dan Ports
6.046 PS1-2

a) $T(n) = \Theta(n^3)$ from the master thm, case 3

b) $T(n) = \Theta(n^{\log_3 6})$ from master, case 1

c) $T(n) = \Theta(n^2 \lg n)$ from master, case 2

d) $T(n) = \Theta(n^3 \lg 3n)$ from the extended master thm

e) $T(n) = \Theta(n^{\sqrt{n}})$ from master thm, case 3

f) $T(n) = \Theta(n)$. Proof by induction:

Let the inductive hypothesis be that $T(n) \leq cn$

(choose appropriate value of c such that the base case satisfies when $T(n)$ is constant for $n \leq n_0$) then the inductive step is

$$T(n) = T(n/3) + 2T(n/4) + n$$
$$\leq cn/3 + 2cn/4 + n$$
$$\leq (c/3 + c/2 + 1)n$$
$$\leq (5/6\, c + 1)n$$
$$\leq cn \quad \text{if we choose a } c > 6$$

Thus $T(n)$ is bounded above by cn for some sufficiently large c by induction, so $T(n) = O(n)$. It is also $\Omega(n)$ by inspection, since there is a n term in the recurrence, so $T(n) = \Theta(n)$

g) Let $m = \lg n$. $T(2^m) = T(2^{m/3}) + m$

Let $S(m) = T(2^m)$

$$S(m) = S(m/3) + m$$

By case 3 of master thm, $S(m) = \Theta(m)$

Thus, $T(n) = \Theta(\lg n)$

h) $T(n) = \Theta(3^n)$ By induction:

Let the induction hypothesis be that

$\leq 3(c3^{n-1} - dn^2) - 3n^2 + 3n + 1) + n^2$

$\leq c3^n - (3c-1)n^2 + 9n^2 - 9n + 3)$

$\leq c3^n - dn^3$ (given d sufficiently large)

Choose c and d large enough for the induction hypothesis to be true for the boundary case of small n. Then, by induction $T(n) = O(3^n)$. By inspection of the recursion tree, $T(n) = \Omega(3^n)$. So $T(n) = \boxed{\Theta(3^n)}$

✓ 1) $T(n) = \Theta(\lg^* n)$

The recursion tree has depth at least $\lg^*(n)$, so $T(n) = \Omega(\lg^* n)$. By induction, with hypothesis $T(n) \leq c\lg^*(n)$,

$T(n) = T(\lg n) + 1 \leq c\lg^*(\lg n) + 1$

$\leq c\lg^* n - c + 1$

Or by def $\leq c\lg^*(n)$ for sufficiently large c

Choose c to meet this condition and the base case. Then $T(n) = O(\lg^* n)$, and it is thus $\boxed{\Theta(\lg^* n)}$

✓ J) $T(n) = \Theta(\sqrt{n})$ by Master Thm case 3.

✗ $T(n) = \Theta(\lg n)$

Dan Ports 8/10 R8
6.046 PS 1-3 No collab

Algorithm:
We can find all unordered pairs (P_1, P_2) such that $A[P_1] \cdot A[P_2] = X$. Begin by using the merge-sort algorithm to place A in sorted order. Then:

Initialize i to 0 and j to length(A) - 1
While (i <= j) {
  If A[i] · A[j] < X
    i = i + 1
  else if A[i] · A[j] > X
    j = j - 1
  else if A[i] · A[j] = X
    { print (i, j)
      i = i + 1; j = j - 1; }
}

original indices?
negative (numbers?
(short) description in plain English!

Proof of correctness:
Invariant: If a pair (P_1, P_2) contains an element whose index is less than i or greater than j, either the product is not X or it has been printed already.
The invariant is vacuously true at the beginning.
Assume it is true at the start of one iteration.
A[i], A[j] is either equal, less than, or greater than X. If less than, then we know that all pairs A[i] · A[a] ≠ X for any a, because every value of a < j will lead to an even smaller product, and every a > j has already been excluded by the invariant. Thus we can increment

have found a matching pair, which we print. The elements of A are distinct, so there are no other pairs containing i and j, so the invariant still holds when we increment i and decrement j. When the loop terminates, $i = j+1$, and by the invariant we will have printed all matching pairs ... ok.

The runtime of this procedure is $\Theta(n \log n)$. We know that mergesort has $\Theta(n \log n)$ running time. The loop is executed at most n times, so that part of the algorithm is $\Theta(n)$. The algorithm as a whole is therefore $\Theta(n \log n)$

This assumes it's ok to sort the array A, changing the order of the elements. If not, it's possible to create an array B of indexes into A and sort it appropriately; this makes the algorithm a bit more complicated, but functionally the same.

Oh, ok then. put this in the algorithm.

Dan Ports R8
6.046 PS1 #4 No collab.

Algorithm: Given n coins, one of which is known to be lighter than the others:
If n=1, the solution is trivial. If n=2, weigh the two coins and identify the lighter one. Otherwise, divide the coins into three equal-size sets (specifically, sets A and B of size $\lfloor n/3 \rfloor$ and set C of the remaining coins, since n may not be an even multiple of 3). Weigh A against B. If they are not equal, the lighter one contains the lighter coin; otherwise C contains the lighter coin. Recursively apply this procedure on the set containing the lighter coin.

+1 fast

Proof of ~~correctness~~: Each iteration of the algorithm takes a set of coins and produces a smaller set that is guaranteed to contain the fake coin. Each set of n coins is divided into three groups, so the fake coin must be in one of them. A and B are the same size, and all other coins are the same weight, so if the fake coin is in set A or B, it will weigh less and the coins of set will be selected. If it is in C, A and B will weigh the same, so the algorithm will select set C. There are no other possibilities. The resulting group has at most size $n - 2\lfloor n/3 \rfloor$, so the number of coins to be searched is a strictly decreasing quantity that will eventually

ok, but see solution

8

Each iteration of the algorithm divides the set to be searched by 3 and discards the rest, so it is $\Theta(\log_3 n)$