Hollis Holmes

Introduction to Algorithms: 6.006
Massachusetts Institute of Technology
Professors Erik Demaine and Srini Devadas

September 8, 2011
Problem Set 1

Problem Set 1

Both theory and programming questions are due **Thursday, September 15** at **11:59PM.** Please download the .zip archive for this problem set, and refer to the `README.txt` file for instructions on preparing your solutions. Remember, your goal is to communicate. Full credit will be given only to a correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

We will provide the solutions to the problem set 10 hours after the problem set is due, which you will use to find any errors in the proof that you submitted. You will need to submit a critique of your solutions by **Tuesday, September 20th, 11:59PM.** Your grade will be based on both your solutions and your critique of the solutions.

Problem 1-1. [15 points] **Asymptotic Practice**

For each group of functions, sort the functions in increasing order of asymptotic (big-O) complexity:

✓ **(a)** [5 points] **Group 1:**

$$\begin{aligned} f_1(n) &= n^{0.999999}\log n \\ f_2(n) &= 10000000n \\ f_3(n) &= 1.000001^n \\ f_4(n) &= n^2 \end{aligned}$$

$n^{0.9}\log n$
n
x^n
n^2

f_1, f_2, f_4, f_3.

✓ **(b)** [5 points] **Group 2:**

$$\begin{aligned} f_1(n) &= 2^{2^{1000000}} \\ f_2(n) &= 2^{100000n} \\ f_3(n) &= \binom{n}{2} \\ f_4(n) &= n\sqrt{n} \end{aligned}$$

c
c^n
$\frac{n!}{2!\,(n-2)!} = n\,n-1 = n^2$
$n^{1.5}$

f_1, f_4, f_3, f_2

X **(c)** [5 points] **Group 3:**

$$\begin{aligned} f_1(n) &= n^{\sqrt{n}} \\ f_2(n) &= 2^n \\ f_3(n) &= n^{10}\cdot 2^{n/2} \\ f_4(n) &= \sum_{i=1}^{n}(i+1) \end{aligned}$$

$n^{n^{0.5}}$
2^n
$n2^n$
n^2

$f_4, f_2, f_3 \quad f_1$

→ $f_1 = n^{\sqrt{n}} = (2^{\lg n})^{\sqrt{n}} = 2^{\sqrt{n}\lg n}$

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

Problem 1-2. [15 points] **Recurrence Relation Resolution**

For each of the following recurrence relations, pick the correct asymptotic runtime:

√ **(a)** [5 points] Select the correct asymptotic complexity of an algorithm with runtime $T(n, n)$ where

$$\begin{aligned} T(x, c) &= \Theta(x) && \text{for } c \leq 2, \\ T(c, y) &= \Theta(y) && \text{for } c \leq 2, \text{ and} \\ T(x, y) &= \Theta(x + y) + T(x/2, y/2). \end{aligned}$$

1. $\Theta(\log n)$.
2. $\Theta(n)$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.
5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

$T(n,n) = \Theta(n+n) + T(n/2, n/2)$

$= \Theta(n+n) + \Theta(n/2 + n/2) + T(n/4, n/4)$

$= \Theta(2n + n + n/2 + \cdots)$ $\sum_{i=0} \frac{n}{2^i}$ $<4n \rightarrow \Theta(n)$

$O(n)$

√ **(b)** [5 points] Select the correct asymptotic complexity of an algorithm with runtime $T(n, n)$ where

$$\begin{aligned} T(x, c) &= \Theta(x) && \text{for } c \leq 2, \\ T(c, y) &= \Theta(y) && \text{for } c \leq 2, \text{ and} \\ T(x, y) &= \Theta(x) + T(x, y/2). \end{aligned}$$

1. $\Theta(\log n)$.
2. $\Theta(n)$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.
5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

$T(n,n) = O(n) + T(x, y/2)$

$= O(n) + O(n) + T(x, y/4)$

$\vdots$

$\underbrace{O(n) + O(n) + \cdots}_{\log(n)}$ $n \log n$

(c) [5 points] Select the correct asymptotic complexity of an algorithm with runtime $T(n, n)$ where

$$\begin{aligned} T(x, c) &= \Theta(x) && \text{for } c \leq 2, \\ T(x, y) &= \Theta(x) + S(x, y/2), \\ S(c, y) &= \Theta(y) && \text{for } c \leq 2, \text{ and} \\ S(x, y) &= \Theta(y) + T(x/2, y). \end{aligned}$$

1. $\Theta(\log n)$.
2. √ $\Theta(n)$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.
5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

$T(x,y) = O(x) + S(x, y/2) = O(x) + O(y) + T(x/2, y/2)$

$= O(x) + O(y) + O(\frac{x}{2}) + S(x, y/2) = O(x) + O(y) + O(x/2) + O(y$

$\vdots$

$O(x) + O(y) + O(\frac{x}{2}) + O(y/2) \cdots$

$x + \frac{x}{2} + \frac{x}{4} + \cdots + y + \frac{y}{2} + \frac{y}{4} \cdots$

sub n for x, y $\quad x[1 + \frac{1}{2} + \frac{1}{4}] \cdots + y[1 + \frac{1}{2} + \frac{1}{4} \cdots]$

$\leq n[2]$ $\quad \Theta(n)$

Peak-Finding

In Lecture 1, you saw the peak-finding problem. As a reminder, a *peak* in a matrix is a location with the property that its four neighbors (north, south, east, and west) have value less than or equal to the value of the peak. We have posted Python code for solving this problem to the website in a file called `ps1.zip`. In the file `algorithms.py`, there are four different algorithms which have been written to solve the peak-finding problem, only some of which are correct. Your goal is to figure out which of these algorithms are correct and which are efficient.

Problem 1-3. [16 points] **Peak-Finding Correctness**

(a) [4 points] Is `algorithm1` correct?

1. Yes.
2. No.

(b) [4 points] Is `algorithm2` correct?

1. Yes.
2. No.

(c) [4 points] Is `algorithm3` correct?

1. Yes.
2. No.

(d) [4 points] Is `algorithm4` correct?

1. Yes.
2. No.

Problem 1-4. [16 points] **Peak-Finding Efficiency**

(a) [4 points] What is the worst-case runtime of `algorithm1` on a problem of size $n \times n$?

1. $\Theta(\log n)$.
2. $\Theta(n)$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.
5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

(b) [4 points] What is the worst-case runtime of `algorithm2` on a problem of size $n \times n$?

1. $\Theta(\log n)$.
2. $\Theta(n)$.

3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.
5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

(c) [4 points] What is the worst-case runtime of `algorithm3` on a problem of size $n \times n$?

1. $\Theta(\log n)$.
2. $\Theta(n)$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.
5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

(d) [4 points] What is the worst-case runtime of `algorithm4` on a problem of size $n \times n$?

1. $\Theta(\log n)$.
2. $\Theta(n)$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.
5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

Problem 1-5. [19 points] **Peak-Finding Proof**

Please modify the proof below to construct a proof of correctness for the *most efficient correct algorithm* among `algorithm2`, `algorithm3`, and `algorithm4`.

The following is the proof of correctness for `algorithm1`, which was sketched in Lecture 1.

> We wish to show that `algorithm1` will always return a peak, as long as the problem is not empty. To that end, we wish to prove the following two statements:
>
> **1. If the peak problem is not empty, then `algorithm1` will always return a location.** Say that we start with a problem of size $m \times n$. The recursive subproblem examined by `algorithm1` will have dimensions $m \times \lfloor n/2 \rfloor$ or $m \times (n - \lfloor n/2 \rfloor - 1)$. Therefore, the number of columns in the problem strictly decreases with each recursive call as long as $n > 0$. So `algorithm1` either returns a location at some point, or eventually examines a subproblem with a non-positive number of columns. The only way for the number of columns to become strictly negative, according to the formulas that determine the size of the subproblem, is to have $n = 0$ at some point. So if `algorithm1` doesn't return a location, it must eventually examine an empty subproblem.
>
> We wish to show that there is no way that this can occur. Assume, to the contrary, that `algorithm1` does examine an empty subproblem. Just prior to this, it must examine

Understanding The Algorithms

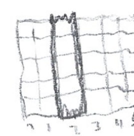
Problem object method/attribute notes.

Algorithm 1

1) given problem object (array augmented w/ more functionality)
2) split into 2 subproblems along the [middle] column.
3) get locations of all items in the dividing column
4) find max of these items
5)  get the highest of this best location and its neighbours.
6) if the highest is the current square: we found a peak!
   else: we haven't found a peak.
   - we previously split the current problem in 2.
   - work on the subproblem that contains the highest neighbour value found above, there must be a peak there.

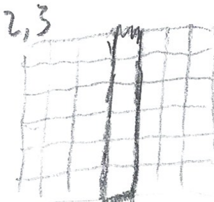
crossProduct(A,B):
list of all pairs (a,b) w/ one item from A and 1 from B.
len(cP(A,B)) = |A|·|B|

divide= cP (range rows, [mid])

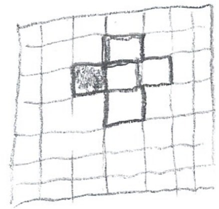


returns loc pairs of all squares in division

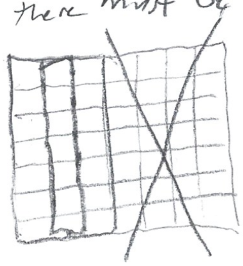
1,2,3



4,5



6,1

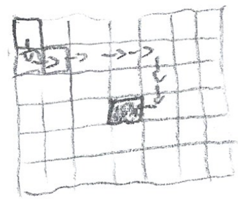


find max in col $O(n)$
recursively dividing in 2 -> $\log(n)+1$ -> $O(\log n)$

$\Big\}\ O(n\log n)$

Algorithm 2

1) Start at location (0,0)
2) find max neighbour
3) if cur location is max; found peak!,
   else; move to max neighbour, repeat step 2-3



$O(1)$ to find max neighbour.
$O(n^2)$ recursions, could visit every square

$\Big\}\ O(n^2)$

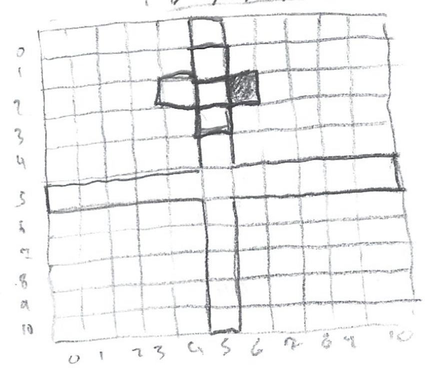
Algorithm 3

divided by 1 row
and by 1 col

2	1
3	4

1) find middle row and middle column.
2) Split current problem into 4 subproblems: NE, NW, SW, SE
3) get all squares on dividing cross (1 row & 1 col)
4) get highest value on the cross
5) find highest neighbour of best cross value.
6) if highest neighbour on cross: found a peak!
   else: recurse on portion of array that has this highest value.

[1, 2, 3, 4, 5]



could still have a peak here?

This is wrong. In the image to the right we recurse on top right subproblem. We have now discarded all other items. per Fig 2, bottom item (val=3) will be seen as a peak because it can no longer see item below w/ val = 4.

$T(a,b) = O(a) + O(b) + T(\frac{a}{2}, \frac{b}{2})$

$a+b + \frac{a+b}{2} + \cdots$

$\boxed{O(n)}$

Algorithm 4

The key difference here is when identifying a peak on the col/row. The ideal location, it's neighbour in case the top/bottom could have been eliminated in a previous round.

So, you have to check if it is a local peak and whether it is greater than the highest seen so far.

Can't eliminate locations based of items that are at least as big as the best seen so far.

worst case 2x case from part C

splits on m, then on n

$2\,O(n) \simeq \boxed{O(n)}$