CS3933 Fall 2007 – Group Assignment 2
Due Thursday October 4, 2007, at 5pm.
Sample Solution

1. A warehouse contains a set of boxes, all aligned with respect to both x and y axes of the warehouse. The position of box i is given by its x_i and y_i coordinates, and each box i has a width (with respect to the x axis) specified by w_i. The door to the warehouse is along its x-axis, so the y coordinate indicates how far the box is from the door, and the x coordinate indicates how far to the right the centre of the box is. You can assume that no boxes are on top of each other.

   A box is visible from the door if there is no box closer to the door (with a smaller y coordinate) that overlaps all of its width (thus blocking all view of the box). Mathematically, a box i is visible if and only if there is some value k such that $x_i - \frac{w_i}{2} \leq k \leq x_i + \frac{w_i}{2}$ and such that for all j with $y_j < y_i$, either $k < x_j - \frac{w_j}{2}$ or $k > x_j + \frac{w_j}{2}$.

   Design a divide and conquer algorithm to determine which boxes are visible, and find the sequence of visible boxes from left to right. Note that a box may be visible in multiple places, if it is partially blocked by narrower boxes.

   Your algorithm should run in time $\in \Theta(n \log n)$, where the running time is not simply due to sorting.

   For this algorithm, we need to produce a sequence of boxes that are visible, ordered from left to right, and (to enable the merge) we also give the edges of each visible box part. Note that when we split, ties with respect to y-coordinate are not important, since boxes with the same y-coordinate cannot block each other.

   Algorithm:(input: set of boxes $B = \{(x_i, y_i, w_i)\}$, number of boxes n
   Sort the set of boxes B by increasing y-coordinate.
   $L \leftarrow Visible(B, 1, n)$

   Algorithm Visible(set of boxes $B = \{(x_i, y_i, w_i)\}$, indices a and b)
   if $b - a = 1$
   $\quad L \leftarrow \langle (x_i - \frac{w_i}{2}, x_i + \frac{w_i}{2}) \rangle$
   $\quad$ return L
   else
   $\quad N \leftarrow Visible(B, a, \lfloor \frac{a+b}{2} \rfloor)$
   $\quad F \leftarrow Visible(B, \lfloor \frac{a+b}{2} \rfloor + 1, b)$
   $\quad L \leftarrow merge(N, F)$

```
  return L

Algorithm merge(N = ⟨(s_i, f_i)⟩, F = ⟨(b_j, e_j)⟩)
i ← 1
j ← 1
let L be an empty list
while (i is not past end of N) or (j is not past end of F)
  if b_j < s_i or i is past end of N (so far box is leftmost)
    if e_j ≤ s_i or i is past end of N (so far box is all visible)
      append (b_j, e_j) to L
      j ← j + 1
    else (far box is only partly visible)
      append (b_j, s_i) to L
      b_j ← s_i
  else (near box is leftmost)
    while f_i > e_j and j is not past end of F (far boxes are covered by near box)
      j ← j + 1
    if f_i > b_j and j is not past end of F (so left part of far box is covered)
      b_j ← f_i
    append (s_i, f_i) to L
    i ← i + 1
return L
```

2. You are analysing a divide & conquer algorithm, and have determined that it has the following characteristics:

   - The algorithm does not recurse if $n = 1$
   - The algorithm finds three parts of the data, each of size $\frac{n}{2}$, in constant time ($\in \Theta(1)$)
   - The algorithm recurses three times (each time on a part of size $\frac{n}{2}$)
   - the algorithm merges the results from the recursion in linear time ($\in \Theta(n)$)

   (a) Complete the asymptotic analysis of this algorithm, by setting up a recurrence for $T(n)$ and solving it. Use the recursion tree analysis method (in which you would sum up all levels of the recursion tree). You can assume that n is a power of 2.

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 3 \cdot T(\frac{n}{2}) + c \cdot n & \text{otherwise} \end{cases}$$

The recursion tree for this recurrence will have deepest level $\log_2 n$, since it divides the size by 2 until it is reduced to 1.

At level k, where level 0 is the root, there will be 3^k nodes, each of size $\frac{n}{2^k}$. Since the amount of work done for each node is linear, then the amount of time needed at level k is

$$W(k) \;=\; 3^k \cdot c \cdot \frac{n}{2^k} \;=\; cn \cdot \left(\frac{3}{2}\right)^k$$

Summing over all levels, the total time taken for the algorithm is

$$\begin{aligned}
T(n) &= \sum_{k=0}^{\log_2 n} W(k) \\
&= \sum_{k=0}^{\log_2 n} cn \cdot \left(\frac{3}{2}\right)^k \\
&= cn \cdot \left(\sum_{k=0}^{\log_2 n} \left(\frac{3}{2}\right)^k\right) \\
&= cn \cdot \left(\frac{\left(\frac{3}{2}\right)^{\log_2 n+1} - 1}{\frac{3}{2} - 1}\right) \\
&= 2 \cdot \frac{3}{2} cn \left(\frac{3}{2}\right)^{\log_2 n} - 2cn \\
&= 3cn \cdot n^{\log_2\left(\frac{3}{2}\right)} - 2cn \\
&= 3cn \cdot n^{\log_2 3 - 1} - 2cn \\
&= 3cn^{\log_2 3} - 2cn
\end{aligned}$$

Since $\log_2 3 > 1$, $T(n) \in \Theta(n^{\log_2 3})$, which is approximately $\Theta(n^{1.58})$.

(b) Consider a change in the algorithm so that the merge of the results takes $\Theta(n^2)$ time. State the new recurrence for $T(n)$ and complete the recursion tree analysis to find the time complexity.

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 3 \cdot T(\frac{n}{2}) + c \cdot n^2 & \text{otherwise} \end{cases}$$

The recursion tree for this recurrence will have deepest level $\log_2 n$, since it divides the size by 2 until it is reduced to 1. At level k, where level 0 is the root, there

will be 3^k nodes, each of size $\frac{n}{2^k}$. Since the amount of work done for each node is cn^2, where n is the size of the node, then the amount of time needed at level k is

$$W(k) \;=\; 3^k \cdot c \cdot \left(\frac{n}{2^k}\right)^2 \;=\; cn^2 \cdot \left(\frac{3}{4}\right)^k$$

Summing over all levels, the total time taken for the algorithm is

$$\begin{aligned}
T(n) \;&=\; \sum_{k=0}^{\log_2 n} W(k) \\
&=\; \sum_{k=0}^{\log_2 n} cn^2 \cdot \left(\frac{3}{4}\right)^k \\
&=\; cn^2 \cdot \left(\sum_{k=0}^{\log_2 n} \left(\frac{3}{4}\right)^k\right) \\
&=\; cn^2 \cdot \left(\frac{1-\left(\frac{3}{4}\right)^{\log_2 n+1}}{1-\frac{3}{4}}\right) \\
&=\; 4cn^2 \cdot \left(1-\left(\frac{3}{4}\right)^{\log_2 n+1}\right) \\
&=\; 4cn^2 - 4cn^2 \cdot \left(\frac{3}{4}\right)^{\log_2 n+1} \\
&=\; 4cn^2 - 4cn^2 \cdot \frac{3}{4} \cdot \left(\frac{3}{4}\right)^{\log_2 n} \\
&=\; 4cn^2 - 3cn^2 \cdot \left(\frac{3}{4}\right)^{\log_2 n} \\
&=\; 4cn^2 - 3cn^2 \cdot n^{\log_2\left(\frac{3}{4}\right)} \\
&=\; 4cn^2 - 3cn^2 \cdot n^{\log_2 3-2} \\
&=\; 4cn^2 - 3cn^{\log_2 3}
\end{aligned}$$

Since $3cn^{\log_2 3} < 2$, we can conclude that $T(n) \in \Theta(n^2)$.