CPSC 320 Midterm 2
Thursday March 13th, 2014

[12] 1. Answer each question with True or False, and then justify your answer briefly.

[2] (a) The Master theorem can be applied to the recurrence relation

$$T(n) = \begin{cases} T(\lfloor n^{2/3} \rfloor) + n & \text{if } n \geq 2 \\ \Theta(1) & \text{if } n = 1 \end{cases}$$

to prove that $T(n) \in \Theta(n)$.

Solution : This is false: $T(\lfloor n^{2/3} \rfloor)$ can not be written as $T(n/b)$ for any positive real number b, and so this recurrence does not have the form needed to apply the Master theorem (its solution is in $\Theta(n)$, however).

[2] (b) A divide and conquer algorithm always splits a problem into approximately equal-sized subproblems.

Solution : This is false. Most of them do, but it is not required (Quicksort is an example of a divide and conquer algorithm that does not usually split the problem into approximately equal-sized subproblems).

[2] (c) The worst-case performance for the Randomized Caching algorithm discussed in class is at least as good as the worst-case performance for the Least-Recently-Used Caching algorithm.

Solution : This is true: both algorithms are marking algorithms, and hence their worst-case performance is at most k times the performance of the optimal algorithm.

[2] (d) Let S be a skip list with 25 keys. It is possible that every node of S, except for the head node and the tail node (if the implementation uses one), will have only a single level.

Solution : This is true; it's very unlikely, but it's possible.

[2] (e) Amortized analysis is used to obtain a tight bound on the running time of one operation on a data structure.

Solution : This is false: it is used to obtain a tight bound on the running time of *a sequence of operations* on a data structure.

[2] (f) When we use the potential method for amortized analysis, the potential of a data structure can only become negative after an operation has been performed if it becomes ≥ 0 during the execution of the next operation.

Solution : This is false: a valid potential function can never be negative.

[7] 2. In class, we proved that given the recurrence

$$T(n) = \begin{cases} aT(n/b) + f(n) & \text{if } n \geq n_0 \\ \Theta(1) & \text{if } n < n_0 \end{cases}$$

and a value $n = b^t$, the function $T(n)$ is equal to

$$\Theta(n^{\log_b a}) + \sum_{j=0}^{t-1} a^j f(n/b^j).$$

[2] (a) Where did the term $\Theta(n^{\log_b a})$ come from?

Solution : This is the work being done by the leaves of the recursion tree.

[2] (b) Where did the term $\sum_{j=0}^{t-1} a^j f(n/b^j)$ come from?

Solution : This is the work being done at all levels of the recursion tree other than the leaves.

[3] (c) How is this formula related to the three cases of the Master Theorem?

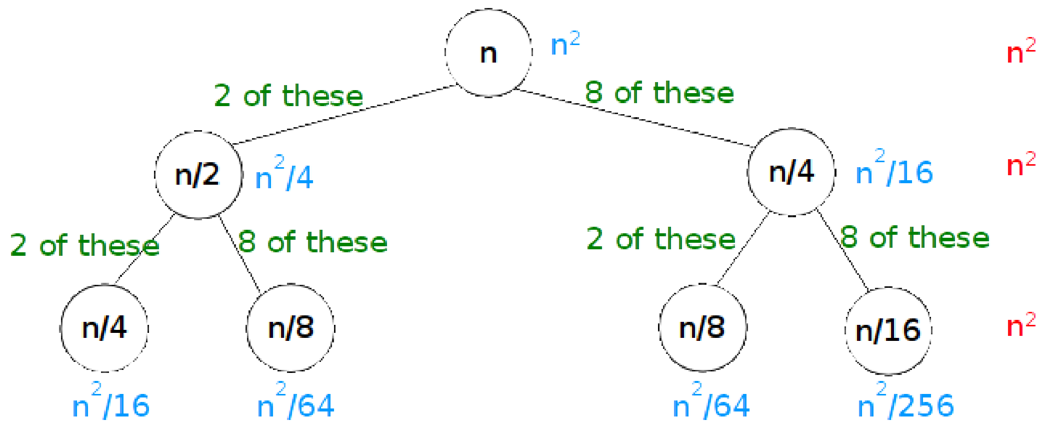
Solution : The three cases are simply cases where we can evaluate the summation $\sum_{j=0}^{t-1} a^j f(n/b^j)$, and determine which of it or the term $\Theta(n^{\log_b a})$ is the dominating term.

[9] 3. Prove upper and lower bounds on the function $T(n)$ defined by

$$T(n) = \begin{cases} 2T(n/2) + 8T(n/4) + n^2 & \text{if } n \geq 4 \\ 1 & \text{if } n \leq 3 \end{cases}$$

Your grade will depend on the quality of the bound you provide (so, proving that $T(n) \in \Omega(1)$ and $T(n) \in O(100^n)$, while true, will not give you many marks).

Solution : Here are the first three levels of the recursion tree (some of the repeated nodes have been removed, and replaced by an indication that there are "2 of these" since the tree would not have fit on the page otherwise).



The children of the root do $2 \times n^2/4 + 8 \times n^2/16 = n^2/2 + n^2/2 = n^2$ work. The grand-children of the root do $4 \times n^2/16 + 16 \times n^2/64 + 16 \times n^2/64 + 64 \times n^2/256 = n^2/4 + n^2/4 + n^2/4 + n^2/4 = n^2$ work as well.

As we can see, the children of a node do the same amount of work, together, as their parent. Hence the work done on every level of the tree is exactly n^2, at least up to the level containing the leaf closest to the root (after which the amount of work done starts decreasing). The tree contains $\log_2 n$ levels, since the path along which the size of the subproblems decreases slowest is that where the size is divided by 2 when we go from one level to the next. Hence the total amount of work done is in $O(n^2 \log n)$ (recall that the base of the log does not matter when we use asymptotic notations, because $\log_x n$ is within a constant factor of $\log_y n$).

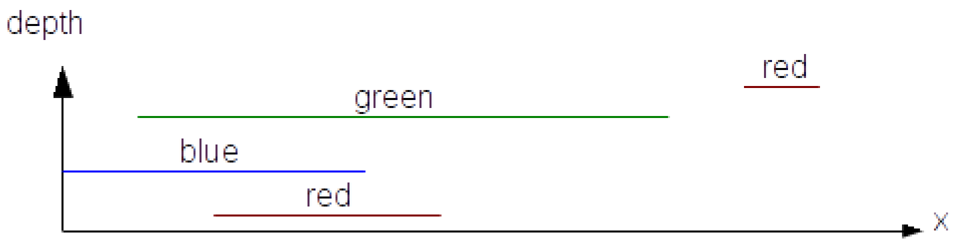
For the lower bound, notice that the first leaf occurs at level $\log_4 n$, and that every level up to that one does exactly n^2 work, and so the total amount of work done is in $\Omega(n^2 \log n)$.

[13] 4. The *Hidden Surface Removal* problem occurs in Computer Graphics and can be described as follows: you are given a collection of coloured objects, which are projected onto a screen to produce an image. The objects are at different distances (depth) from the screen; objects that are closer cover those that are further back. The problem is to decide how to color each point of the screen.

In this question we consider the one-dimensional version of the problem: each object is an interval on the x-axis, with a depth and a color, and can be represented by a 4-tuple (x_l, x_r, d, c). For instance, $(0, 4, 3, \texttt{blue})$ is an interval that goes from $x = 0$ to $x = 4$, has depth 3, and color `blue`.

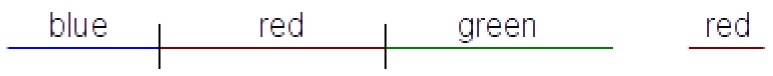
[10] a. Describe an efficient divide-and-conquer algorithm that takes as input a collection of n shapes, and returns a list of 4-tuples (x_l, x_r, d, c) where such a 4-tuple indicates that the points from $x = x_l$ to $x = x_r$ should be colored c (and the line segment with that color is at depth d). For instance, given the shapes

$$(2, 5, 1, \texttt{red}), (9, 10, 6, \texttt{red}), (0, 4, 3, \texttt{blue}), (1, 8, 5, \texttt{green})$$



your algorithm should return

$$(0, 2, 3, \texttt{blue}), (2, 5, 1, \texttt{red}), (5, 8, 5, \texttt{green}), (9, 10, 6, \texttt{red}).$$



(the vertical bars were added as separators between colors because the exam is printed in black and white). More space is available for your answer on the next page.

Solution : Note: the solution is presented here in a lot more details than I expected you to provide on the exam. The first part alone would have been worth 5/10, even if you provided no details about how the merge is carried out.

We mimic the `Mergesort` algorithm. The algorithm will return the 4-tuples sorted from left to right (the ordering simplifies the merge operation considerably).

```
Algorithm OneDSurfaceRemoval(S, first, last)

if first = last then
  return S[first ... first]
endif

mid ← ⌊ (first + last)/2 ⌋
scene1 ← OneDSurfaceRemoval(S, first, mid)
scene2 ← OneDSurfaceRemoval(S, mid + 1, last)
return OneDSurfaceMerge(scene1, scene2)
```

The merge stage takes in two arrays of intervals (with depths and colors) that are already sorted from left to right. No two intervals in one array overlap, although of course intervals in the first array may overlap with intervals in the second array. The algorithm proceeds from left to right. It maintains a variable `xmin` that contains the smallest value of x that has not yet been assigned a color. At each step, we consider one interval from each subproblem, and decide how to deal with them up to the end of the first one to terminate. There are 3 cases:

i. There is no interval immediately to the right of $x =$ `xmin`. In this case we simply advance `xmin` to the beginning of the next interval. No output is produced.
ii. There is only one interval immediately to the right of $x =$ `xmin`. We add the portion of this interval that comes before the start of the first interval in the other array to our output, and update `xmin`.
iii. There are two intervals immediately to the right of $x =$ `xmin`. Only one of them will be visible until one of the two terminates. We add that one to the output, and then update `xmin`.

In every case we move `xmin` from one endpoint of an interval to another endpoint of an interval. Since there are at most $O(n)$ interval endpoints when we have n intervals, this merge process thus runs in $O(n)$ time. Pseudo-code follows (**it was not required on the exam!** I will only be looking for the right ideas):

```
Algorithm OneDSurfaceMerge(scene1, scene2)

i ← 0
j ← 0
```

```
S ← { }
xmin ← −∞

while (i < length[scene1] and j < length[scene2]) do
  //
  // Case 1
  //
  if xmin < scene1[i].xl and xmin < scene2[j].xl then
    xmin ← min (scene1[i].xl, scene2[j].xl)

  //
  // Case 2 (with one interval).
  //
  else if xmin ≥ scene1[i].xl and xmin < scene2[j].xl then
    if scene1[i].xr ≤ scene2[j].xl then
      add (xmin, scene1[i].xr, scene1[i].depth, scene1[i].col) to S
      xmin ← scene1[i].xr
      i ← i + 1
    else
      add (xmin, scene2[j].xl, scene1[i].depth, scene1[i].col) to S
      xmin ← scene2[j].xl
    endif

  //
  // Case 2 again (with the other interval).
  //
  else if xmin ≥ scene2[j].xl and xmin < scene1[i].xl then
    if scene2[j].xr ≤ scene1[i].xl then
      add (xmin, scene2[j].xr, scene2[j].depth, scene2[j].col) to S
      xmin ← scene2[j].xr
      j ← j + 1
    else
      add (xmin, scene1[i].xr, scene2[j].depth, scene2[j].col) to S
      xmin ← scene1[i].xl
    endif

  //
  // Case 3 (the interval in scene 1 finishes first, or maybe both).
  //
  else if scene1[i].xr ≤ scene2[j].xr
    if scene1[i].depth < scene2[j].depth then
      add (xmin, scene1[i].xr, scene1[i].depth, scene1[i].col) to S
    else
      add (xmin, scene1[i].xr, scene2[j].depth, scene2[j].col) to S
    endif
    xmin ← scene1[i].xr

    i ← i + 1
    if (xmin = scene2[j].xr) then
      j ← j + 1
    endif

 //
 // Case 3 (the interval in scene 2 finishes first).
 //
  else if scene2[j].xr ≤ scene2[i].xr
    if scene1[i].depth < scene2[j].depth then
      add (xmin, scene2[j].xr, scene1[i].depth, scene1[i].col) to S
    else
      add (xmin, scene2[j].xr, scene2[j].depth, scene2[j].col) to S
    endif
    xmin ← scene2[j].xr
    j ← j + 1
  endif
endwhile

//
// Now we need to collect the remaining intervals in each scene. Only
// one of these loops will actually execute.
//
while i < length[scene1] do
    add (xmin, scene1[i].xr, scene1[i].depth, scene1[i].col) to S
    i ← i + 1
    if i < length[scene1] then
      xmin ← scene1[i].xl
    endif
endwhile

while j < length[scene2] do
    add (xmin, scene2[j].xr, scene2[j].depth, scene2[j].col) to S
    j ← j + 1
    if j < length[scene2] then
      xmin ← scene2[j].xl
    endif
endwhile

return S
```

After this, there may be multiple consecutive intervals with the same depth and color. Consolidating these is left as an exercise.

[3] b. Analyze the running time of your algorithm.

Solution : The running time of the algorithm can be described by the same recurrence relation as the running time of `Mergesort`, and so it is in $\Theta(n \log n)$.

[4] 5. Mr. Isulo, an alien computer scientist, decided he does not like the definition of *good pivot* used in the expected time analysis of the `RandomizedQuickSelect` algorithm. Instead, he decides to call a pivot *good* if $\lfloor n/10 \rfloor \leq \texttt{leftsize} \leq n - 1 - \lfloor n/10 \rfloor$. That is, good pivots are any pivot that ends up in the middle 80% of the array after the call to `RandomizedPartition`.

What recurrence relation would Mr. Isulo get for the expected time $E(T(n))$ of algorithm `RandomizedQuickSelect` using his definition of *good* pivot?

Solution : The recurrence relation for Mr. Isulo's algorithm is:

$$E(T(n)) = \begin{cases} E(T(9n/10)) + \Theta(n) & \text{if } n \geq 10 \\ \Theta(1) & \text{if } n \leq 9. \end{cases}$$

The expected value of $g(n)$ in this case is 1.25, which is a constant.