CS161 Handout 03
Summer 2001 July 5th, 2001

Problem Set 2: Algorithms and Asymptotics

This week's problem set is more CS161-specific that last week's, and will require you to read and study Chapters 1, 2, 3, and 7 of the course textbook. As always, you are free to ask us any questions you have about the material and the homework exercises.

Due: Wednesday, July 11th at 5 p.m.

Problem 1: Solving Recurrences (20 points)

Give asymptotically tight upper (big O) bounds for $T(n)$ in each of the following recurrences. Justify your solution by naming the particular case of the master theorem, by iterating the recurrence, or by making an intelligent guess and substituting. Assume that $T(n) = \Theta(1)$ for small enough n.

- $T(n) = T(n-2) + 1$
- $T(n) = 2T(n/2) + n\lg^2 n$
- $T(n) = 9T(n/4) + n^2$
- $T(n) = 3T(n/2) + n$
- $T(n) = T\left(n/2 + \sqrt{n}\right) + n$

Problem 2: More Heapifying (10 points)

Consider the code for **`Build-Heap`** from Chapter 7, which operates on a heap stored in an array $A[1\ldots n]$.

```
Build-Heap(A)
   heap-size[A] ← length[A]
   for i ← ⌈length[A]/2⌉ downto 1
      do Heapify(A,i)
```

- Show how this procedure can be implemented as a recursive divide-and-conquer procedure **`Build-Heap(A,i)`** where **`A[i]`** is the root of the subheap to be built. To build the entire heap, one would **call `Build-Heap(A,1).`**
- Give a recurrence that describes the worst-case running time of your procedure.
- Solve the recurrence using the master method.

Problem 3: Quick and Dirty Data Structures (20 points)

For each of the following data structures, assume that an infinite amount of memory is available, and that you needn't free memory even when it is no longer needed.

- Show how to implement a queue that efficiently supports **`Enqueue`**, **`Dequeue`**, **`Extract-Min`** as well as **`Minimum`**. **`Enqueue`**, **`Dequeue`** and **`Extract-Min`** should run in logarithmic time—that is, $O(\lg n)$ time—whereas **`Minimum`** should run in constant time.
- Show how to efficiently implement a priority queue that supports **`Insert`**, **`Extract-Min`**, and **`Extract-Max`** to all run in $O(\lg n)$ time.