Algorithms
Computer Science 140 & Mathematics 168
Instructor: B. Thom
Spring 2004
Homework 3a
Due on 2/05/04 (beginning of class)

1. [**15 points**] **Amortizing your Heap!**

   Recall that, for any node i in a heap, `heapify`(i) runs in at most the height, $h(i)$, of the subtree rooted at i. This is because heapify percolates i downwards in the tree, swapping it with one of its children until either i becomes a leaf or it maintains a desired ordering with respect to its children. Thus, heap can be built as follows:

```
buildheap(A[1,...,i,...,n]) {
  for i = parent(n) downto 1
    heapify(i)
}
```

   Since the height of the overall tree is $O(\log n)$, a naive analysis of this function's runtime provides $O(n \log n)$. You're now going to use amortization to argue the tighter bound mentioned in class: $\Theta(n)$.

   First, you'll make an upper bound argument using an amortized analysis. Here's how you should proceed: At "birth," give every node some number of rubles in which to pay for future work. Your goal is to argue that, with your proposed payment scheme, all of heapify's work at node i can be paid for by rubles in the bank. To do this:

   (a) Prove that, when `buildheap` is at node i, there are $h(i)$ rubles left in the bank after its heapify operation (you'll want to do this with respect to whatever accounting scheme you propose; I suggest some constant number of rubles per node).

   (b) Use this observation to link back into the desired claim: that `buildheap` runs in $O(n)$ time.

   (c) Why can you conclude that this bound is tight, i.e. its $\Theta(n)$?

2. [**10 points**] **Diminishing Returns!** Asymptotically solve the following recurrence (be sure to show all of your work):

$$\begin{aligned} \mathrm{T}(1) &= c \\ \mathrm{T}(n) &= \mathrm{T}\left(\frac{n}{3}\right) + \mathrm{T}\left(\frac{n}{3}\right) + \mathrm{f}(n) \\ \mathrm{f}(n) &= \Theta(n) \end{aligned}$$

   and answer the following questions:

(a) What assumption did you make about n and why will your answer apply to any positive integer n?

(b) What's happening to the number of items per row as you descend down the work tree?

(c) Does the final asymptotic run-time depend on the tree's height? Explain your answer.

(d) What answer do you get when $\mathrm{f}(n) = c$?

(e) What answer do you get when $\mathrm{f}(n) = n^2$?

Again the $(\frac{2}{9})^k$ part of the first term can be ignored when making an upper bound. Moreover, since each row again reduces by $\frac{2}{3}n$ each time—size is only squared when calculating cost per node—we can safely drop the second term, because it is upper bounded by $O(n)$. Again, the largest order is $O(n^2)$, which has the same order as $\mathrm{f}(n)$.