CS 3343 Analysis of Algorithms – Fall 11

9/6/11

2. Homework

Due **9/13/11** before class

1. **Big-Oh ranking (6 points)**
   Rank the following functions by order of growth, i.e., find an arrangement $f_1, f_2, ...$ of the functions satisfying $f_1 \in O(f_2)$, $f_2 \in O(f_3)$,... . Partition your list into equivalence classes such that f and g are in the same class if and only if $f \in \Theta(g)$. For every two functions f_i, f_j that are adjacent in your ordering, prove shortly why $f_i \in O(f_j)$ holds. And if f and g are in the same class, prove that $f \in \Theta(g)$.

   $$n^2\ ,\ \log n\ ,\ 2^n\ ,\ \sqrt{n}\ ,\ \sqrt[3]{n}\ ,\ \log\log n\ ,$$

   Bear in mind that in some cases it might be useful to show $f(n) \in o(g(n))$, since $o(g(n)) \subset O(g(n))$. If you try to show that $f(n) \in o(g(n))$, then it might be useful to apply the rule of l'Hôpital which states that

   $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

   if the limits exist; where $f'(n)$ and $g'(n)$ are the derivatives of f and g, respectively.

2. **Code snippet (3 points)**
   Give the Θ-runtime depending on n for the code snippet below. Justify your answer. Note that `log` is $\log_2$ and is assumed to run in constant time.

```
for(i=1; log(i)<=n; i=i+1){
  for(j=1; j<=n*n; j=j+3){
    for(k=1; k<=j; k++){
        print(" ");
```

3. **Heaps with links (7 points)**
   (a) (2 points) Consider storing a heap as a linked binary tree with pointers. Please give pseudo-code on how you would store a heap node, and which modifications you need to make to the heap routines that we discussed in class. What are the runtimes of the heap routines?
   (b) (2 points) Now consider storing a heap as a linked list with pointers. Please give pseudo-code on how you would store a heap node, and which modifications you need to make to the heap routines that we discussed in class. What are the runtimes of the heap routines?
   (c) (1 points) Which of the three heap implementations (array, linked tree, linked list) is preferrable? Justify your answer.
   (d) (2 points) Assume you are given two heaps of height h each, that are given as linked binary trees. And assume we do not require that the last level of the heap is "flushed left", i.e., keys can be in any place in the last level. Give an **efficient** algorithm that merges those two heaps into one heap (without the "flushed left" condition). Analyze the runtime of your algorithm.