Problem 1. Reorder this sequence so each function is big-O of the following function:

$$n^2 \quad 6^{\log_2 n} \quad n! \quad 11^n \quad \log_2(n!) \quad 2^{3n} \quad 10n^{3/2} \quad 100n \quad 3^{\log_4 n} \quad \log_2^4 n$$

Problem 1 answer

$\log_2^4 n$

$3^{\log_4 n}$

$100\, n$

$\log_2(n!)$

$10\, n^{3/2}$

$6^{\log_2 n}$

n^2

2^{3n}

11^n

$n!$

Problem 2. Use the so-called naive upper and lower bounds, and the rule for geometric sums, to determine the big-Θ value of each of the following sums.

(a) $\displaystyle\sum_{i=1}^{\log_2 n} i(\log_3 i)^2$

(b) $\displaystyle\sum_{i=1}^{n} i^2/3^i$

(c) $\displaystyle\sum_{i=1}^{2^n} (\log_2 i)^2$

(d) $\displaystyle\sum_{i=1}^{n} 1.2^i/i$

Problem 2 answer

(a) $\displaystyle\sum_{i=1}^{\log_2 n} i(\log_3 i)^2$ is $\Theta(log^2 n \cdot log^2(logn))$

(b) $\displaystyle\sum_{i=1}^{n} i^2/3^i$ is $\Theta(1)$

(c) $\displaystyle\sum_{i=1}^{2^n} (\log_2 i)^2$ is $\Theta(n^2 2^n)$

(d) $\displaystyle\sum_{i=1}^{n} 1.2^i/i$ is $\Theta(\frac{1.2^n}{n})$

Problem 3. For each of the algorithms below, use the steps to find big-Θ. To make it easier, ignore floors in your calculations. That is, for (a) and (c) assume n is a power of 3. For (b) assume n is a power of 4.

(a)
```
def PrintAs($n$):                                              Assume $n$ is a power of 3.
  1. for $i = 1$ to $3n$: print("A")                           ...print $3n$ A's
  2. if $n > 1$:
  3.     for $j = 1$ to 4: PrintAs($\lfloor n/3 \rfloor$)      ...make 4 recursive calls, each with input $\lfloor n/3 \rfloor$
```

(b)
```
def PrintBs($n$):                                              Assume $n$ is a power of 4.
  1. for $i = 1$ to $n^2$: print("B")                          ...print $n^2$ B's
  2. if $n > 1$:
  3.     for $j = 1$ to 16: PrintBs($\lfloor n/4 \rfloor$)     ...make 16 recursive calls, each with input $\lfloor n/4 \rfloor$
```

(c)
```
def PrintCs($n$):                                              Assume $n$ is a power of 3.
  1. for $i = 1$ to $5n$: print("C")                           ...print $5n$ C's
  2. if $n > 1$:
  3.     for $j = 1$ to 2: PrintCs($\lfloor n/3 \rfloor$)      ...make 2 recursive calls, each with input $\lfloor n/3 \rfloor$
```

(continued)

Problem 3(a) answer

(i) Define $A(n)$ to be the number of letters printed by PrintAs given input $n \geq 1$.

Then $A(1) = 3$.

For $n > 1$, $A(n) = 3n + 4A(\frac{n}{3})$.

(ii) Within level i:

The number of nodes is 4^i.

The problem size is $n/3^i$.

The number of letters printed by each call is $n \cdot \frac{n}{3^i} = \frac{n}{3^{i-1}}$.

(iii) The number of levels is $log_3 n$.

(iv) The total number of letters printed is A(n) $= \sum_{i=0}^{log_3 n} 3n(\frac{4}{3})^i$.

(v) Using Rule for Geometric Sums , the big-Θ-value of $A(n)$ is $\Theta(n^{log_3 4})$.

(continued)

Problem 3(b) answer

(i) Define $B(n)$ to be the number of letters printed by PRINTBS given input $n \geq 1$.

Then $B(1) = 1$.

For $n > 1$, $B(n) = n^2 + 16B(\frac{n}{4})$.

(ii) Within level i:

The number of nodes is 16^i.

The problem size is $\frac{n}{4^i}$.

The number of letters printed by each call is $\frac{n^2}{16^i}$.

(iii) The number of levels is $log_4 n$.

(iv) The total number of letters printed is B(n) $=$ $\sum_{i=0}^{log_4 n} n^2 = n^2 log_4 n$.

(v) Using Naive Upper and Lower Bounds , the big-Θ value of $B(n)$ is $\Theta(n^2 logn)$.

(continued)

Problem 3(c) answer

(i) Define $C(n)$ to be the number of letters printed by PrintBs given input $n \geq 1$.

Then $C(1) = 5$.

For $n > 1$, $C(n) = 5n + 2C(\frac{n}{3})$.

(ii) Within level i:

The number of nodes is 2^i.

The problem size is $\frac{n}{3^i}$.

The number of letters printed by each call is $\frac{5n}{3^i}$.

(iii) The number of levels is $log_3 n$.

(iv) The total number of letters printed is $C(n) \;=\; \sum_{i=0}^{log_3 n} 5n(\frac{2}{3})^i$.

(v) Using Rule for Geometric Sums , the big-Θ value of $C(n)$ is $\Theta(n^{log_3 2})$.

Problem 4. Consider a variant of mergesort that, given an array of size N, recurses as usual unless the subproblem to be solved is of size $\sqrt{N}$ or less, in which case it solves the subproblem without recursing, instead using insertion sort (a quadratic-time algorithm) on the subproblem. Use a recursion-tree analysis to determine the asymptotic worst-case running time of this algorithm on arrays of size N.

You may assume N is an even power of two. That is $N = 2^{2k}$ for some integer k, so $\sqrt{N} = 2^k$.

Problem 4 answer

Fix the input size N.

(i) For $n \leq N$, define $T(n)$ to be run-time on sub-arrays of size n.

Then, for $n \leq \sqrt{N}$, we have $T(n) = n^2$

For $n > \sqrt{N}$, we have $T(n) = n + 2T(\frac{n}{2})$.

(ii) Within each level i:

The number of nodes is 2^i.

The problem size is $\frac{N}{2^i}$.

If i is not the bottom level, the work done per subproblem is $\frac{N}{2^i}$.

If i is the bottom level, the work done per subproblem is $\frac{N^2}{4^i}$.

(iii) The number of levels is $k = \frac{log_2 N}{2}$, because for number of levels, we need to find the number of times we divide N by 2 to get to $N^{1/2}$ (after that, there are no new levels, as insertion sort takes place) - let the number of times we divide be m. Since We know $N = 2^{2k}$ and $N^{1/2} = 2^k$, we are trying to find m such that $\frac{2^{2k}}{2^m} = 2^k$. We get $2^m = \frac{2^{2k}}{2^k} = 2^k$. Thus, $m = k$, so the number of levels $= m = k = \frac{log_2 N}{2}$.

(iv) Summing over the levels, the total work is

$$T(N) \;=\; N^{3/2} + \frac{N log_2 N}{2} - N.$$

(v) The big-Θ value of this sum is

$$\Theta(N^{3/2}).$$

Problem 5. Prove or disprove the following theorem:

Theorem 1. *In every execution of the hospitals-propose Stable Matching algorithm, there is at most one hospital that makes offers to every doctor.*

If the theorem is false, describe a counter-example: a particular instance, and an execution of the algorithm on that instance, in which at least two hospitals make offers to every hospital. If the theorem is true, give a long-form proof. In the proof, you may reuse without proof any observations, lemmas, or theorems from the lecture notes, but cite the observation, lemma, or theorem by number when you do, to let the reader know.

Problem 6. Prove or disprove the following theorem:

Theorem 2. *In any execution of the hospitals-propose Stable Matching algorithm there is at least one doctor that receives no offer before the final iteration.*

If the theorem is false, describe a counter-example: a particular instance, and an execution of the algorithm on that instance, in which every doctor receives an offer before the final iteration. If the theorem is true, give a long-form proof. In the proof, you may reuse without proof any observations, lemmas, or theorems from the lecture notes, but cite the observation, lemma, or theorem by number when you do, to let the reader know.

Problem 6 answer: the theorem is true

Proof (long form).

1. Consider any execution of the algorithm.

2.1. Proof by Contradiction: Assume every doctor received at least one offer before the final iteration.

2.2. Any unmatched doctor d always becomes matched after it receives an offer, so no doctor is unmatched.

2.3. All hospitals are matched because the algorithm produces a perfect matching, as a hospital must be unmatched to make an offer to a doctor.

2.4. This is a contradiction, as we assumed we aren't on the final iteration, but all hospitals and doctors are matched.

3. By block 2, every doctor cannot receive an offer before the final iteration, so there is at least one doctor that receives no offer before the final iteration. □