Problem Set 1 Solutions

Problem 1-1. Recurrence Relations

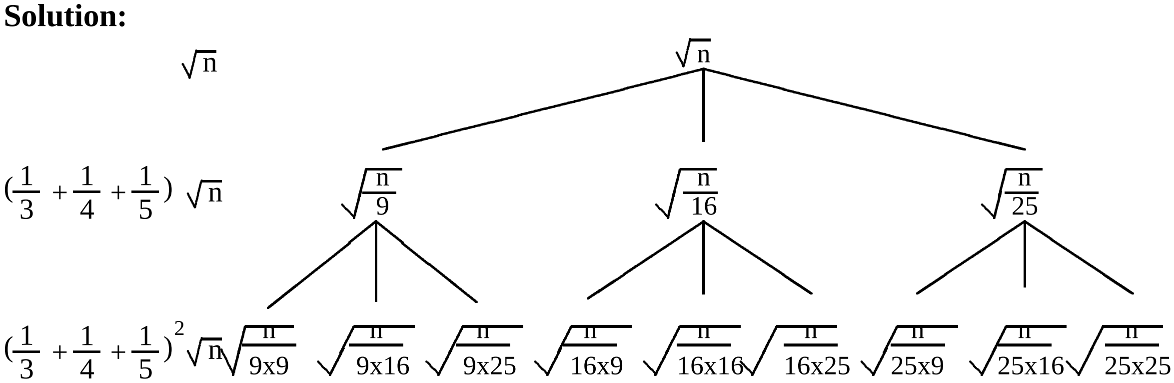
Solve the following recurrences. Give a Θ bound for each problem. If you are unable to find a Θ bound, provide as tight upper (O or o) and lower (Ω or ω) bounds as you can find. Justify your answers. You may assume that $T(1) = 1$.

(a) $T(n) = 7T(\frac{n}{2}) + n\sqrt{n}$
Solution: $\Theta(n^{\log_2 7})$ - By part 1 of the Master Method.

(b) $T(n) = 4T(\frac{n}{2}) + n^3 \log n$
Solution: $\Theta(n^3 \log n)$ - By part 3 of the Master Method.

(c) $T(2^n) = T(2^{n-1}) + 2^n$
Solution: $\Theta(2^n)$. Substitute m for 2^n. The resulting relation is $T(m) = T(\frac{m}{2}) + m$. Clearly, this is $\Theta(m)$ and thus $\Theta(2^n)$.

(d) $T(n) = T(\frac{n}{9}) + T(\frac{n}{16}) + T(\frac{n}{25}) + \sqrt{n}$
Solution:



The figure above shows the recursion tree that helps us in guessing a solution.
We have that

$$T(n) \leq \sum_{i=0}^{\lfloor \log_9 n \rfloor} \left(\frac{47}{60}\right)^i \sqrt{n} \leq \sum_{i=0}^{\infty} \left(\frac{47}{60}\right)^i \sqrt{n} = \frac{1}{1 - \frac{47}{60}} \sqrt{n} = O(\sqrt{n})$$

Thus we guess that $T(n) = O(\sqrt{n})$ and we prove it by substitution. Assume that $T(m) \leq c\sqrt{m}$ for an appropriate constant c and for all $m < n$. Then we have that

$$\begin{aligned} T(n) &\le c\sqrt{\frac{n}{9}} + c\sqrt{\frac{n}{16}} + c\sqrt{\frac{n}{25}} + \sqrt{n} \\ &= \left(\frac{47}{60}c + 1\right)\sqrt{n} \\ &= c\sqrt{n} - \left(\frac{13}{60}c\sqrt{n} - \sqrt{n}\right) \\ &\le c\sqrt{n}\,, \text{ for } c \ge 60/13 \end{aligned}$$

Hence $T(n) = O(\sqrt{n})$.
On the other hand by inspection we have $T(n) \ge \sqrt{n} = \Omega(\sqrt{n})$. Hence $T(n) = \Theta(\sqrt{n})$.

(e) $T(n) = T(n-1) + n^3$
Solution: $T(n) = \sum_{i=1}^{n-1} i^3 = \Theta(n^4)$

(f) $T(n) = T(\log n) + O(1)$
Solution: $T(n) = \Theta(\log^* n)$ - By inspection and substitution.

(g) $T(n) = 9T(\frac{n}{27}) + (\sqrt[3]{n})^2$
Solution: $\Theta(n^{\frac{2}{3}} \log n)$ - By part 2 of the Master Method.

Problem 1-2. Asymptotic Notation

Rank the following functions by order of growth; that is, find an arrangement $g_1, g_2, \ldots, g_{16}$ of the functions satisfying $g_1 = \Omega(g_2)$, $g_2 = \Omega(g_3)$, $\ldots$, $g_{19} = \Omega(g_{16})$. Partition your list into equivalence classes such that $f(n)$ and $g(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$. (The function $\log^* n$ is discussed on pages 55-56 of CLRS.)

$n^{2+\sin^2 n}$	$\sum_{k=1}^{n} \frac{1}{k}$	n	n^2
$n!$	3^{10000}	3^n	$\log n$
2^n	$n^{\log\log n}$	n^3	$(\log n)^{\log n}$
$n \log n$	$(n+1)!$	$\sum_{k=1}^{n} k$	$\log(n!)$

Solution:

The following are ordered asymptotically from smallest to largest, are as follows (two functions, f and g are on the same line if $f(n) = \Theta(g(n))$):

$$3^{10000}$$

$$\log n \qquad \sum_{k=1}^{n} \frac{1}{k}$$

$$n$$

$$n \log n \qquad \log(n!)$$

$$\sum_{k=1}^{n} k \qquad n^2$$

$$n^{2+\sin^2 n}$$

$$n^3$$

$$(\log n)^{\log n} \qquad n^{\log \log n}$$

$$2^n$$

$$3^n$$

$$n!$$

$$(n+1)!$$

Problem 1-3. Sieve of Eratosthenes

The Sieve of Eratosthenes, invented circa 200 B.C., is an algorithm to find all prime numbers between 2 and an input number N. The algorithm works as follows: we begin with a list of all integers from 2 to N. For each $m \leq N$, we cross out (i.e. mark as composite) each multiple of m ($n \cdot m$ for $n \geq 2$) that is less than or equal to N. When this process terminates, only the prime numbers between 2 and N are unmarked.

Below, we give pseudocode for this algorithm. At the beginning of the algorithm, every entry in the array P is initialized to *true*, i.e. $P[i]$ is *true* for all $i, 2 \leq i \leq N$. At the end of the algorithm, $P[i]$ is *true* iff i is prime.

```
ERATOSTHENES-SIEVE(N):
1  Let P[i] ← true for all i from 2 to N.
2  for m ← 2 to N:
3     n ← 2.
4     while n · m ≤ N:
5           P[n · m] ← false.
6           n ← n + 1.
7     endwhile
8  endfor
```

The table below shows the values of the array elements $P[1 \ldots N]$ at the end of each **while** loop (line 7) during an execution of the algorithm run on input $N = 13$.

i	$P[2]$	$P[3]$	$P[4]$	$P[5]$	$P[6]$	$P[7]$	$P[8]$	$P[9]$	$P[10]$	$P[11]$	$P[12]$	$P[13]$
1	T	T	T	T	T	T	T	T	T	T	T	T
2	T	T	**F**	T	**F**	T	**F**	T	**F**	T	**F**	T
3	T	T	F	T	**F**	T	F	**F**	F	T	**F**	T
4	T	T	F	T	F	T	**F**	F	F	T	**F**	T
...	T	T	F	T	F	T	F	F	F	T	F	T
13	T	T	F	T	F	T	F	F	F	T	F	T

Prove that the ERATOSTHENES-SIEVE algorithm is correct; that is, prove that upon termination, $P[i]$ is *true* iff i is prime.

Hint: You can use the following pre-condition and post-condition and you can prove the suggested loop invariant for the **for** loop (line 2). Let S_i represent the statement: $P[i]$ is *true* iff i is prime.

Pre-Condition: $N \geq 2$.

Post-Condition: $S_i, \forall i$ such that $2 \leq i \leq N$.

Loop Invariant: $S_i, \forall i$ such that $2 \leq i \leq m$.

Solution:

We will use the given loop-invariant for the **for** loop (line 2) and the relevant pre- and post-conditions to prove the correctness of the algorithm.

We will now prove the given loop invariant. Namely, we will prove by induction that when m is assigned the value k (line 2), S_i holds for all $i, 2 \leq i \leq k$.

When m is assigned the value 2, $P[2]$ is *true* since it was initialized to be *true*. Thus, we have proved that *before* we execute the **for** loop the first time, the base case of the loop invariant holds.

Now we will assume that when m is assigned the value k, S_i holds for all $i, 2 \leq i \leq k$. We will use this inductive hypothesis to prove that when m is assigned the value $k+1$, S_i holds for all $i, 2 \leq i \leq k+1$.

Let's consider an execution of the **while** loop with $m = k$. Since n goes from 2 to $\lfloor N/m \rfloor$, $n \cdot m$ is always at least $2k$. Since only elements $P[i]$ with $i = n \cdot m$ are (re)set to *false*, no element $P[i]$ with $i \leq k$ will change value. Thus, S_i for $i, 2 \leq i \leq k$, will hold at the end of the execution of the **while** loop.

It remains to show that S_{k+1} holds at the end of the execution of the **while** loop. If $P[k+1]$ is *true*, then it cannot have any divisor d such that $2 \leq d \leq k$. So $k+1$ must be prime. If $k+1$ is prime, then it does not have a divisor $d \leq k$. Thus, it must be *true*. Thus, we have proved that the loop invariant is correct.

When $m \geq N$, on the final execution of the **for** loop, we have that S_i holds for all i, $2 \leq i \leq N$, due to the correctness of the loop invariant.