Homework 1

Mengxiang Jiang
CSEN 5336 Analysis of Algorithms

September 1, 2022

Problem 1. Some functions are given below. Sort them in ascending order of asymptotic growth (big-O). (lg is log function with base 2)
1. $5\lg n$
2. $6n\lg n$
3. $n^{n/8}$
4. $7\lg\lg n$
5. $n^{0.6}$
6. $2n^{\lg n}$
7. $\lg^{12} n$ [or $(\lg n)^{12}$]
8. $(n/2)^n$
9. $n^{0.5}\lg n$
10. $3n$

$7\lg\lg n < 5\lg n < \lg^{12} n < n^{0.5}\lg n < n^{0.6} < 3n < 6n\lg n < 2n^{\lg n} < n^{n/8} < (n/2)^n$

Problem 2. Solve the recurrence relations using the master method
a) $T(n) = 2T(n/2) + 3n$
b) $T(n) = 2T(n/4) + 4n^{0.3}$
c) $T(n) = 2T(n/2) + 2n^2$
d) $T(n) = 3T(n/3) + 3n\lg n$
e) $T(n) = 2T(n/2) + \Theta(n)$

a) $a = 2,\ b = 2,\ f(n) = 3n,\ \log_b a = \log_2 2 = 1,\ n^{\log_b a} = n^1 = \Theta(n) = \Theta(f(n))$
since $f(n)$ is the same size as $n^{\log_b a}$, case 2 applies and $T(n) = \Theta(n^{\log_b a}\log n) = \Theta(n\log n)$

b) $a = 2,\ b = 4,\ f(n) = 4n^{0.3},\ \log_b a = \log_4 2 = 0.5$
$n^{\log_b a} = n^{0.5} = \Theta(n^{0.5}) > \Theta(f(n)) = \Theta(n^{0.3})$
since $f(n)$ is polynomially smaller than $n^{\log_b a}$, case 1 applies and $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{0.5})$

c) $a = 2,\ b = 2,\ f(n) = 2n^2,\ \log_b a = \log_2 2 = 1,\ n^{\log_b a} = n^1 = \Theta(n) < \Theta(f(n)) = \Theta(n^2)$
since $f(n)$ is polynomially larger than $n^{\log_b a}$, case 3 applies and $T(n) = \Theta(n^2)$

d) $a = 3,\ b = 3,\ f(n) = 3n\lg n,\ \log_b a = \log_3 3 = 1$

$n^{\log_b a} = n^1 = \Theta(n) < \Theta(f(n)) = \Theta(n \log n)$
since $f(n)$ is larger but not polynomially larger than $n^{\log_b a}$, $T(n)$ is not solvable by the master theorem

e) $a = 2$, $b = 2$, $f(n) = \Theta(n)$, $\log_b a = \log_2 2 = 1$, $n^{\log_b a} = n^1 = \Theta(n) = \Theta(f(n))$
since $f(n)$ is the same size as $n^{\log_b a}$, case 2 applies and $T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n \log n)$

Problem 3. Calculate the running time of the algorithms using big-O notation:
a)

```
for (i = 1; i*i<n; i++)
    printf(\%d\n", i)
```

b)

```
for (i = n; i > 1; i = ceil(i/8))
    printf(\%f\n", i);
```

a) Since n is compared with $i * i = i^2$, and as i grows linearly, i^2 grows quadratically, so the time it will take for i^2 to reach n is $\sqrt{n}$, so $T(n) = O(\sqrt{n}) = O(n^{0.5})$

b) Since i is divided by 8 each iteration, starting with $i = n$, the amount of time for i to reach 1 or lower is $\log_8 n$, so $T(n) = O(\log(n))$