CSE 8833
HW Assignment #2
09/16/2023
Group #1

Omid Abdolazimi
Adam Thomas
Charles Albright
Joseph Jabour
Martin Duclos
Robert Ward
Safae El Amrani

***Problem* 1: $T(n)$ is the running time of an algorithms with $T(n) = an^2 + bn + c$, where $a,b,c \succ 0$. Prove that $T(n) \in O(n^3)$.**

We need to show that, by definition of *O*, that there exist constants $c_1, n_0 \succ 0$ such that:
$0 \le an^2 + bn + c \le c_1 n^3$ for every $n \ge n_0$.
If we divide both sides by n^3 , we get:
$0 \le a / n + b / n^2 + c / n^3 \le c_1$ for every $n \ge n_0$.
Let $n_0 = 1$, and we know $a,b,c \succ 0$.
Then pick $c_1 = a + b + c$.
Then $0 \le an^2 + bn + c \le c_1 n^3$ for every $n \ge n_0$, and thus, $T(n) \in O(n^3)$.

***Problem* 2: $T(n)$ is the running time of an algorithms with $T(n) = 16n^2 + 8n + 1$. Prove that $T(n) \notin O(n)$.**

Suppose $T(n) \in O(n)$. Then, by definition, there exist constants $c, n_0 \succ 0$ such that:
$0 \le 16n^2 + 8n + 1 \le cn$ for every $n \ge n_0$.
If we divide both sides by *n*, we get:
$16n + 8 + \frac{1}{n} \le c$ for every $n \ge n_0$.
If we look at each piece, ($\frac{1}{n}$) goes to zero for sufficiently large *n*. 8 is a constant. However, $16n \le c$ is not true for sufficiently large *n*, thus we have reached a contradiction. Therefore, $T(n) \notin O(n)$.

***Problem* 3: $T(n)$ is the running time of an algorithms with $T(n) = 64n^2 + 16n + 1$. Prove that $T(n) \in$:**

a)* $\Theta(n^2)$ *or* *b)* $O(n^2)$ *or* *c) Both* $\Theta(n^2)$ *and* $O(n^2)$ *or* *d) None of the above

Let us first show if $T(n) \in O(n^2)$; then, we must show that, by definition of *O*, that there exist constants $c_1, n_0 \succ 0$ such that:

$0 \le 64n^2 + 16n + 1 \le c_1 n^2$ for every $n \ge n_0$.

Omid Abdolazimi
Adam Thomas
Charles Albright
Joseph Jabour
Martin Duclos
Robert Ward
Safae El Amrani

If we divide both sides by n^2, we get:

$64+\frac{16}{n}+\frac{1}{n^2}\leq c_1$ for every $n\geq n_0$.

Let $c_1=65$ and $n_0=17$. Therefore, $T(n)\in O(n^2)$.

Let us also show if $T(n)\in\Omega(n^2)$; then, we must show that, by definition of Ω, that there exist constants $c_2, n_0 \succ 0$ such that:

$64n^2+16n+1\geq c_2 n^2$ for every $n\geq n_0$.

If we divide both sides by n^2, we get:

$64+\frac{16}{n}+\frac{1}{n^2}\geq c_2$ for every $n\geq n_0$.

Let $c_2=1$ and $n_0=1$. Therefore, $T(n)\in\Omega(n^2)$.

Since $T(n)\in O(n^2)$ and $T(n)\in\Omega(n^2)$, then $T(n)\in\Theta(n^2)$. Therefore, ***(c)*** is the answer.

***Problem* 4: Consider the insertion sort algorithm on:**

(*i*) arbitrary inputs;
(*ii*) every input;
(*iii*) inputs are sorted in non-decreasing order.

Give the expression of the asymptotic tight bound for the running time of this algorithm for the these three cases. [Note: Justify your answer]

(*i*) The case of arbitrary inputs represents the worst case for the insertion sort algorithm. In this case, the input array is in reverse order, and the inner loop is executed $(p-1)$ times, for $p=2,3,\ldots,n$. The number of moves is $2(n-1)+(1+2+\ldots+n-1)=2(n-1)+n(n-1)/2$. So, the number of key comparisons is $(1+2+\ldots+n-1)=n(n-1)/2$. This means, in the worst case, the insertion sort is $O(n^2)$.

(*ii*) For the case of every input, this means the average case for the insertion sort. We look at all possible initial data organizations. For the average case, the insertion sort is $O(n^2)$.

(*iii*) When the inputs are sorted in non-decreasing order, this gives the best case for the insertion sort. The inner loop will not be executed. The number of moves is $2(n-1)$, and the number of key comparisons is $(n-1)$. This means that, for the best case, the insertion sort is $O(n)$.

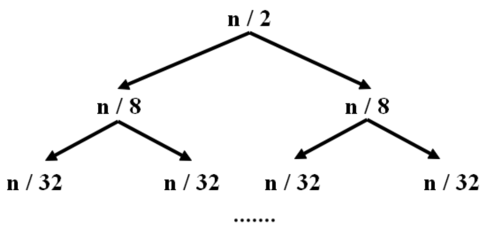
CSE 8833
HW Assignment #2
09/16/2023
Group #1

Omid Abdolazimi
Adam Thomas
Charles Albright
Joseph Jabour
Martin Duclos
Robert Ward
Safae El Amrani

***Problem* 5: Find the upper bound of the running times of an algorithms characterized by the following recurrence relation (Hint: use the substitution method):**

1) $T(n) = 2T\left\lfloor \frac{n}{4} \right\rfloor + \frac{n}{2}$

We can use a Binary tree to form a good guess for a solution in the substitution method:



Top level is $i=0$. The subproblem size at level i is $n/2^{i+1}$. The height of the tree is the level when the subproblem equals 1. When the subproblem = 1, we have $1 = n/2^{i+1}$. This gives $2^{i+1} = n$, which means $i = \log n - 1$. This makes the height of the tree $h = \log n - 1$.

Cost of level $i=0$ is $n/2$. Cost of level $i=1$ is $n/8 + n/8 = 2n/8 = n/4$. Cost of level $i=2$ is $4 \times (n/32) = n/8$.

Cost of a node is $f\ (subproblem) = (n/2^i)/4 = (2^i)/2^2 = n/2^{i+2}$.

Cost of a level is # nodes × cost of a node $= 2 \times (n/2^{i+2}) = n/2^{i+1}$.

The total cost at all levels is:

$$T(n) = \sum_{i=0}^{\log n - 1} n/2^{i+1}$$

$$\leq n \times \sum_{i=0}^{\infty} (1/2)^{i+1}$$

$$= n(1) = n$$

Based on the binary tree above, guess a solution of *n*.

Now, we need to prove that $T(n) \leq cn$ for an appropriate constant $c \succ 0$.

Assume correct for ($\left\lfloor \frac{n}{4} \right\rfloor$). Then $T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) \leq c \times \left\lfloor \frac{n}{4} \right\rfloor$.

Substitute in $T(n)$:

CSE 8833
HW Assignment #2
09/16/2023
Group #1

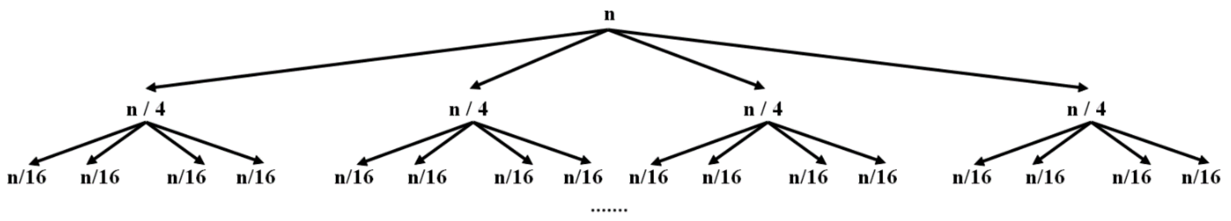
Omid Abdolazimi
Adam Thomas
Charles Albright
Joseph Jabour
Martin Duclos
Robert Ward
Safae El Amrani

$$T(n) = 2 \times T\left\lfloor \frac{n}{4} \right\rfloor + \frac{n}{2}$$
$$\leq 2[c \times \left\lfloor \frac{n}{4} \right\rfloor] + \frac{n}{2}$$
$$= 2c \times \left\lfloor \frac{n}{4} \right\rfloor + \frac{n}{2}$$
$$\leq 2c \times \frac{n}{4} + \frac{n}{2}$$
$$= \frac{nc}{2} + \frac{n}{2} = \frac{n}{2}(c+1) = \frac{(c+1)}{2} \times n$$
$$\leq cn, \; for \;\; c \succ 1$$

Therefore, the upper bound is $O(n)$.

2) $T(n) = 4\ T\left\lfloor \frac{n}{4} \right\rfloor + n$

We can use a binary tree to guess a solution for the substitution method:



Sum the cost at each level, then, the total cost at all levels.
The cost at each level sums to n. The height of the tree is $\log n$. So the total cost is $n\log n + n$ (to account for the head node). We will guess $n\log n$ because we do not have to do lower order terms for upper bounds.
Guess $O(n\log n)$. Prove there exist constants $c, n_0 \succ 0$ such that $T(n) \leq cn\log n$.
Basis step: $n = 1 : n\log n = 0 = T(n)$ Yes.
Hypothesis: $T(k) = ck\ \log k$ for some $c \succ 0$, all $k \prec n$.
Inductive step:

CSE 8833
HW Assignment #2
09/16/2023
Group #1

Omid Abdolazimi
Adam Thomas
Charles Albright
Joseph Jabour
Martin Duclos
Robert Ward
Safae El Amrani

$$T(n) = 4T\left\lfloor \frac{n}{4} \right\rfloor + n$$
$$\le 4T(\frac{n}{4}) + n$$
$$= 4(c(\frac{n}{4})\log(\frac{n}{4})) + n$$
$$= cn\log(\frac{n}{4}) + n$$
$$= cn\log n - cn\log 4 + n$$
$$= cn\log n - 2cn + n$$
$$= cn\log n - n(2c-1) \le cn\log n,\ for\ c \succ 1/2$$

Therefore, the upper bound is $O(n\log n)$. QED

***Problem* 6: Use the *master theorem* to find the asymptotic tight bounds for the running times of the algorithms characterized by the following recurrence relations. For each relation, indicate the case of the master theorem you used to find your solution and explain the reason for which you have used that case.**

a) $T(n) = 64\ T\left(\frac{n}{8}\right) + n$

$a = 64,\ b = 8,\ f(n) = n$. Compare $f(n)$ with $n^{\log_b^a}$.
$\log_8^{64} = (\log 64)/(\log 8) = 6/3 = 2$. Compare n with n^2.
Use ***Case (i)*** of the master theorem because $n \ll n^2$ for large *n*. Thus, $f(n) = O(n^{2-\varepsilon})$ for some $\varepsilon \succ 0$.
Therefore, $T(n) = \Theta(n^2)$.

b) $T(n) = T\left(\frac{5n}{8}\right) + 1$

$a = 1,\ b = 8/5,\ f(n) = 1$. Compare $f(n)$ with $n^{\log_b^a}$.
$\log_{8/5}^{1} = 0$. Compare 1 with $n^0 = 1$.
Use ***Case (ii)*** of the master theorem because we are comparing 1 with 1.
Therefore, $T(n) = \Theta(n^0\log n) = \Theta(\log n)$.

Omid Abdolazimi
Adam Thomas
Charles Albright
Joseph Jabour
Martin Duclos
Robert Ward
Safae El Amrani

c) $T(n) = 9\,T\left(\dfrac{n}{16}\right) + n\log n$

$a = 9,\ b = 16,\ f(n) = n\log n$. Compare $f(n)$ with $n^{\log_b^a}$.

$\log_{16}^{9} \approx 0.7925$. Compare $n\log n$ with $n^{0.7925}$. $n\log n = \Omega(n^{\log_{16}^{9+\varepsilon}})$ for some $\varepsilon \succ 0$, which is ***Case (iii)***.

Check the regularity condition of $a\,f(a/b) \le cn\log n$ for some $c \prec 1$ and all sufficiently large *n*.

Show that $9(n/16 \times \log(n/16)) \le cn\log n$.

LHS:
$(9/16)n\log(n/16) = (9/16)n(\log n - \log 16) = (9/16)n\log n - (9/16)n(4) = (9/16)n\log n - (9/4)n$

$\le (9/16)n\log n = cf(n)$.

So, choose $c = 9/16 (c \prec 1)$.

So, the regularity condition is satisfied. Therefore, by ***Case (iii)***, $T(n) = \Theta(n\log n)$.

d) $T(n) = 8\,T\left(\dfrac{n}{2}\right) + n^3\log^4(n)$

$a = 8,\ b = 2,\ f(n) = n^3\log^4 n$. Compare $f(n)$ with $n^{\log_b^a}$.
$\log_2^8 = 3$. Compare $n^3\log^4 n$ with n^3 .
$f(n)$ is asymptotically larger than n^3 but not polynomially larger than n^3 .This recurrence falls into the gap between ***Case (ii)*** and ***Case (iii)***.
Case (iv) can be applied here because:
$f(n) = \Theta(n^3\log^k n)$ for some $k \ge 0$. Here, $k = 4$.
Therefore, $T(n) = \Theta(n^3\log^{k+1} n)$ .Thus, $T(n) = \Theta(n^3\log^5 n)$.

CSE 8833
HW Assignment #2
09/16/2023
Group #1

Omid Abdolazimi
Adam Thomas
Charles Albright
Joseph Jabour
Martin Duclos
Robert Ward
Safae El Amrani

***Problem* 7: We consider the merge sort algorithm that uses a divide-and-conquer design strategy. The running time of a merge sort algorithm is given below:**

$$T(n) = \begin{cases} 2 & if \quad n = 2 \\ 2\,T(\frac{n}{2}) + n & if \quad n = 2^k,\ k \succ 1 \end{cases}$$

Prove that the expression giving the complexity of the merge sort algorithm with which you are already familiar is also a solution to the above recurrence. Hint: use mathematical induction.

Prove that $T(n) = \Theta(n \log n)$.

So, we prove that $T(n) = n \log n$ for the given recurrence.

Basis: $T(2) = 2\log 2 = 2 = T(n)$, Yes.

Inductive step: Assume for $n = 2^k$ and $k \succ 1$, assume $T(2^k) = 2^k \log 2^k$. This means $T(2^k) = k\,2^k$.

Assume it is true for $T(2^k) = 2\,T(2^k / 2) + 2^k = 2\,T(2^{k-1}) + 2^k$.

Prove true for $T(2^{k+1})$:

$T(2^{k+1}) = 2\,T(2^k) + 2^{k+1}$

$= 2\left[2^k \log 2^k\right] + 2^{k+1}$

$= 2^{k+1} \times k + 2^{k+1}$

$= (k+1)\ 2^{k+1}$

Therefore, $T(n) = \Theta(n \log n)$.

QED