COMP 3711 HW1

Hu, Wenbin 20760747 , email : whuak@connect.ust.hk.

Problem 1:

(a) from the question, we have : $T(1) = T(2) = 1$, $\forall n > 2$, $T(n) \le T(\lfloor \frac{n}{3} \rfloor) + C$ ($C > 0$ is constant)

(i) let $n = 3^k$, $k = \log_3 n$, we can simplify the equation as $T(n) \le T(\frac{n}{3}) + C$

$$\begin{aligned} \Rightarrow T(n) &\le T(\tfrac{n}{3}) + C \\ &\le T(\tfrac{n}{3^2}) + C + C \\ &\le \cdots \\ &\le T(\tfrac{n}{3^h}) + h \cdot C \qquad (h = \log_3 n) \\ &= T(1) + C \cdot \log_3 n \\ &= 1 + C \cdot \log_3 n \end{aligned}$$

$\therefore T(3^k) \le 1 + C \cdot \log_3 3^k = 1 + C \cdot k \le C_0 k$

$\therefore \exists C_0 \ge \frac{1}{k_0} + C$, $k_0 \in \mathbb{N}$, $\forall k \ge k_0$, $T(3^k) \le C_0 k$, which means $T(3^k) = O(k)$

(ii) we can find a $k \in \mathbb{N}$ s.t. $3^{k-1} \le n \le 3^k$ for every $n \in \mathbb{N}$, so $3^k \le 3n$ and $k \le \log_3 3n$

from $S(3^k) = O(k)$,

we have : $\exists C_0 > 0$, $n_0 > 0$. for $\forall n \ge n_0$. s.t. $S(3^k) \le C_0 \cdot k$

$\because S(n)$ is a non-decreasing function of n.

$\therefore S(n) \le S(3^k) \le C_0 \cdot k \le C_0 \cdot \log_3 3n \le C_0 (\log_3 3 + \log_3 n) \le 2C_0 \cdot \log_3 n$

$\therefore \exists C_1 \ge 2C_0$, $n \ge n_0$, $\forall n \ge n_0$, s.t. $S(n) \le C_1 \log_3 n$, which is $S(n) = O(\log_3 n)$

$\therefore$ proved.

(iii) by definition, we have : $\forall n \ge 1$, $R(n) = \max_{1 \le i \le n} T(i)$

so, $R(x)$ has these 2 property : I. $R(n)$ is a non-decreasing function

II. for $\forall i \in \{1, \dots n\}$, $T(i) \le R(i)$

from Question, we have: $\forall n > 2$, $T(n) \le T(\lfloor \frac{n}{3} \rfloor) + C$, $T(1) = T(2) = 1$

$$T(n) \le T(\lfloor \tfrac{n}{3} \rfloor) + C \le R(\lfloor \tfrac{n}{3} \rfloor) + C$$

$$\Rightarrow \max_{2 < i \le n} T(i) \le \max_{2 < i \le n} \left(R(\lfloor \tfrac{i}{3} \rfloor) + C \right) = R(\lfloor \tfrac{n}{3} \rfloor) + C$$

$$\Rightarrow R(n) \le R(\lfloor \tfrac{n}{3} \rfloor) + C$$

$\therefore$ proved.

(iv) from (i) : we know : if $S(3^k)$, then $T(3^k) = O(k)$

from (iii) : we know : by definition of $\forall n \ge 1$, $R(n) = \max_{1 \le i \le n} T(i)$, we have $\forall n > 2$, $R(n) \le R(\lfloor \frac{n}{3} \rfloor) + C$

$\Rightarrow R(3^k) = O(k)$ (same proof as (i))

from (ii) : if $S(n)$ is non-decreasing function: $S(3^k) = O(k) \Rightarrow S(n) = O(\log_3 n)$

$\therefore R(n) = O(\log_3 n)$

$\therefore \exists C_0 > 0$, $n_0 > 0$, $\forall n \ge n_0$, $R(n) \le C_0 \log_3 n$

$\therefore T(n) \le c_0 \log_3 n = \frac{c_0}{\log 3} \cdot \log n \le c_1 \log n = O(\log n)$ $\quad (c_1 \ge \frac{c_0}{\log 3})$

$\Rightarrow \exists c_1 \ge \frac{c_0}{\log 3}$, $\forall n \ge n_0$, $T(n) \le c_1 \log n$, which is $T(n) = O(\log n)$

(b)

(v) for every $n \in \mathbb{N}$, we can find exactly $k \in \mathbb{N}$, s.t. $2^{k-1} \le n \le 2^k$. so $2^k \le 2n$ and $k \le \log_2 2n$

by $S(2^k) = O(k \cdot 2^k)$, we have: $\exists n_0 > 0, c_0 > 0, \forall n \ge n_0$, $S(2^k) \le c_0 \cdot k \cdot 2^k$

since $S(n)$ is a nondecreasing function of n.

$\therefore S(n) \le S(2^k) \le c_0 \cdot k 2^k \le c_0 \cdot (\log_2 2n) \cdot 2n \le 2c_0 n (\log_2 2 + \log_2 n) \le 4c_0 (n \log_2 n) \le c_1 (n \log_2 n)$ $\quad (c_1 \ge 4c_0)$

$\therefore \exists c_1 \ge 4c_0 > 0$, $\forall n \in n_0$, $S(n) \le c_1 (n \log_2 n)$

which is $S(n) = O(n \log_2 n)$

(vi) $\forall n > 1$, $T(n) \le T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n$,

by definition: $\forall n \ge 1$, $R(n) = \max_{1 \le i \le n} T(i)$

$\therefore$ $R(n)$ is non-decreasing function, and $\forall i \in [1, n]$, $T(i) \le R(i)$

$\Rightarrow$ $T(\lfloor \frac{i}{2} \rfloor) \le R(\lfloor \frac{i}{2} \rfloor)$, $T(\lceil \frac{i}{2} \rceil) \le R(\lceil \frac{i}{2} \rceil)$

$\Rightarrow$ $T(i) \le R(\lfloor \frac{i}{2} \rfloor) + R(\lceil \frac{i}{2} \rceil) + i$

$\Rightarrow$ $\max_{1 < i \le n} T(i) \le \max_{1 < i \le n} \{ R(\lfloor \frac{i}{2} \rfloor) + R(\lceil \frac{i}{2} \rceil) + i \} \le R(\lfloor \frac{n}{2} \rfloor) + R(\lceil \frac{n}{2} \rceil) + n$

$\Rightarrow$ $R(n) \le R(\lfloor \frac{n}{2} \rfloor) + R(\lceil \frac{n}{2} \rceil) + n$

$\therefore$ proved.

(vii) by definition: $T(n) \le R(n)$

when $n = 2^k$, then $\forall n > 1$, $R(2^k) \le 2R(2^{k-1}) + 2^k$

$\le 2^2 R(2^{k-2}) + 2 \cdot 2^k + 2^k$

$\cdots$

$\le 2^k R(1) + 2^k \sum_{i=0}^{k-1} 2^i$. $\quad (h = \log_2 n, R(1) = 1)$

$= 2^k + 2^k \frac{1 - 2^{\log_2 k}}{1 - 2} = k \cdot 2^k$

$\therefore R(2^k) = O(k \cdot 2^k)$

$\therefore R(n)$ satisfies $T(2^k) = O(k \cdot 2^k)$

from (v): we know $R(n) = O(n \log_2 n)$

$\exists c_0 > 0, n_0 > 0$. $\forall n \ge n_0$. s.t. $R(n) \le c_0 (n \log_2 n)$.

$\because T(n) \le R(n)$

$\therefore T(n) \le c_0 n \log_2 n$

$\therefore \exists c_0 > 0, n_0 > 0$, $\forall n \ge n_0$. s.t. $T(n) \le c_0 n \log_2 n$.

which is $T(n) = O(n \log n)$

Problem 2:

(a) $A = O(B)$

(b) $A = \Theta(B)$

(c) $A = \Omega(B)$

(d) $A = \Theta(B)$

(e) $A = \Omega(B)$

(f) $A = O(B)$

(g) none of the relations is satisfied.

Problem 3. (a) let $h = \log_3 n$

$$T(n) = 10\,T(n/3) + n^2$$

$$= 10^2\,T(n/3^2) + \frac{10}{3^2}\cdot n^2 + n^2$$

$$= 10^3\,T(n/3^3) + \left(\frac{10}{3^2}\right)^2 n^2 + \frac{10}{3^2} n^2 + n^2$$

$$= \cdots$$

$$= 10^i\,T(n/3^i) + \left(1 + \frac{10}{3^2} + \left(\frac{10}{3^2}\right)^2 + \cdots \left(\frac{10}{3^2}\right)^{i-1}\right) n^2$$

$$= \cdots$$

$$= 10^h\,T(n/3^h) + n^2 \cdot \sum_{i=0}^{h-1} \left(\frac{10}{9}\right)^i$$

$$= 10^{\log_3 n} \cdot T(1) + n^2 \cdot \frac{1 - \left(\frac{10}{9}\right)^{\log_3 n}}{1 - \frac{10}{9}}$$

$$= 10 \cdot n^{\log_3 10} - 9n^2 \leq 10 \cdot n^{\log_3 10} = O(n^{\log_3 10})$$

(b) let $h = \log_4 n$

$$T(n) = 16\,T(n/4) + n^2$$

$$= 16^2 \cdot T(n/4^2) + \frac{16}{4^2} n^2 + n^2$$

$$= 16^3 \cdot T(n/4^3) + \left(\frac{16}{4^2}\right)^2 n^2 + \frac{16}{4^2} n^2 + n^2$$

$$= \cdots$$

$$= 16^i \cdot T(n/4^i) + i \cdot n^2$$

$$= \cdots$$

$$= 16^h \cdot T(n/4^h) + h \cdot n^2$$

$$= 16^{\log_4 n} \cdot T(1) + \log_4 n \cdot n^2$$

$$= n^2 + \log_4 n \cdot n^2$$

$$= O(n^2 \log n)$$

Problem 4:

(a)

linear-time Algorithm:

$V_{min} \leftarrow \infty$, $X_{max} \leftarrow 0$ // V_{min} is used to keep the current min term in $P[1..k]$, X_{max} is to keep the current max in $P[1..k]$

$i \leftarrow 0$, $j \leftarrow 0$, $i_v \leftarrow 0$ // i_v is used to keep the index of V_{min}

for $k \leftarrow 1$ to n:

  if $P[k] < V_{min}$: $V_{min} \leftarrow P[k]$, $i_v \leftarrow k$ // if we find a $P[k]$ is smaller than V_{min}, we renew V_{min}, keep the related index i

  if $P[k] - V_{min} > X_{max}$: $X_{max} \leftarrow P[k] - V_{min}$, $i \leftarrow i_v$, $j \leftarrow k$ // if we find $P[k] - V_{min}$ is bigger than the $\max_{1 \le i \le j \le n}\{P[j] - P[i]\}$ we found, we renew X_{max}, and keep the related index i, j

if $X_{max} > 0$: return X_{max}

else : return 'nil'

explanation: we do the for loop, keep the current minimum, and do the substraction iteratively,

if we find a element smaller than the current minimum V_{min}, then renew the current minimum

if we find the result of the substraction is greater than the $\max_{1 \le i \le j \le k-1}\{P[j] - P[i]\}$ we found,

that means we find $\max_{1 \le i \le j \le k}\{P[j] - P[i]\} = P[k] - V_{min}$, so we renew X_{max}

(b) we do Induction to prove it:

denote $J(k)$: when we do the k^{th} step in for loop iteratively, we can find $\max_{1 \le i \le j \le k}\{P[j] - P[i]\}$

Base Case: $k = 1$: there is no $\max_{1 \le i \le j \le 1}\{P[j] - P[i]\}$, let $V_{min} \leftarrow P[1]$

$k = 2$: if $P[2] - P[1] > 0$, then we keep $X_{max} = P[2] - P[1]$

since $\max_{1 \le i \le j \le 2}\{P[j] - P[i]\} = \begin{cases} P[2] - P[1], & P[2] > P[1] \\ \text{'nil'}, & \text{else} \end{cases}$ $\therefore J(2)$ is correct

General Case: $k > 2$

Assume $J(k-1)$ is correct by Induction hypothesis.

so in $P[1 \ldots k-1]$, V_{min} keep the minimum term, X_{max} keep the value of $\max_{1 \le i \le j \le k-1}\{P[j] - P[i]\}$

when we do the k^{th} step:

if we find : $A[k] \le V_{min}$, which means V_{min} is not the minimum in $P[1 \ldots k]$

  then we need to renew it as $V_{min} \leftarrow P[k]$ (since $\max_{1 \le i \le j \le k}\{P[j] - P[i]\}$ need a $\min_{1 \le i \le j} P[i]$)

else : we still have the minimum in $P[1 \ldots k]$

if we find : $A[k] - V_{min} > X_{max}$, which means X_{max} is not the $\max_{1 \le i \le j \le k}\{P[j] - P[i]\}$

  then we renew it as $X_{max} \leftarrow A[k] - V_{min}$

else : we still keep the $X_{max} = \max_{1 \le i \le j \le k}\{P[j] - P[i]\}$

So, through above explanation, we can always find the $X_{max} = \max_{1 \le i \le j \le k}\{P[j] - P[i]\}$

$\therefore J(k)$ is correct

$\therefore$ from 1 to n, in $P[1 \ldots n]$, we can always find the $\max_{1 \le i \le j \le n}\{P[j] - P[i]\}$

(C) before the for loop we have $O(1)$ operations, after the for loop we have $O(1)$ operations

inside the for loop: in the first line we have $O(1)$ operations
in the second line we have $O(1)$ operations

∴ after we finish the loop, we have $O(n)+O(1)$ operations, which is $O(n)$ operations

∴ my algorithm is $O(n)$ time

Problem 5:

(a) O(logn) time algorithm:

Search(A, p, r):

$q \leftarrow \lfloor \frac{p+r}{2} \rfloor$ // get the middle index

if (q=0 or $A[q] \le A[q-1]$) and (q=n-1 or $A[q] \le A[q+1]$) // find the local minimal

return q

else

if q>0 and $A[q-1] \le A[q]$: // if the left item's value is less than A[q] then the local minimal is on the left of A[q] and abandon $A[q+1 \cdots r]$

return Search(A, p, q-1)

else if q<n-1 and $A[q+1] \le A[q]$: // if the right item's value is less than A[q] then the local minimal is on the right of A[q] and abandon $A[p \cdots q-1]$.

return Search(A, q+1, r)

FirstCall: Search(A, 0, n-1)

(b) Proof: let m = r-p+1, $q = \lceil \frac{m}{2} \rceil$

I. first we find the middle item A[q] in A[p....r], , check whether it is the local minimal.

if q=0, we check whether $A[0] \le A[1]$, if so, A[0] is the local minimal we find

if q=n-1, we check whether $A[n-1] \le A[n-2]$, if so, A[n-1] is the local minimal we find

if q is "inside" of A[1...n-1], i.e. q≠0, q≠n. check whether it's local minimal.

II. if A[q] is not local minimal. then we use this method:

if $A[q-1] \le A[q]$: then the local minimum must in $A[p \cdots q-1]$, go to find it

if $A[q+1] \ge A[q]$: then the local minimum must in $A[q+1 \cdots r]$, go to find it

(explanation of the correctness of this method: if $A[q-1] \le A[q]$, there is 2 situation:

if $\forall i \in \{p, p+1, \cdots q-1\}$, $A[i] \le A[i+1]$, then A[p] must be the local minima.

else $\exists i \in \{p, \cdots, q-1\}$, $A[i+1] \ge A[i]$, $A[i-1] \ge A[i]$)

So. after the recursion, we must find local minimum in A[p...r]

So. for A[0....n-1], we also can find the local minimum in it.

(c) let T(n) be the function of running time of the algorithm

let m = r-p+1, $q = \lfloor \frac{r+p}{2} \rfloor$

before the "else block", we have O(1) time operations.

then we analyse the "else block":

if $A[q-1] \le A[q]$, then we abandon $A[q+1 \cdots r]$, do the recursion with $T(\lfloor \frac{n}{2} \rfloor)$ time

if $A[q-1] \ge A[q]$, then we abandon $A[p \cdots q-1]$, do the recursion with $T(\lfloor \frac{m-1}{2} \rfloor)$ time

So. the whole "else block" will operate $T(m) = \max\{T(\lfloor \frac{m}{2} \rfloor), T(\lfloor \frac{m-1}{2} \rfloor)\} + O(1) \le T(\lfloor \frac{n}{2} \rfloor) + O(1)$

∴ for the whole algorithm. $T(n) \le T(\lfloor \frac{n}{2} \rfloor) + O(1)$, which implies $T(n) = O(\log n)$