ICCS 313: Assignment 1

Tawan Chaeyklinthes u5980963

Date : 14 Sep 2018

Problem 1

(a) $3n^3 + 75n^2 + 8\log_2 n \in \mathcal{O}(n^3)$

Proof:

$$\begin{aligned} 3n^3 + 75n^2 + 8\log_2 n &\leqslant 3n^3 + 75n^3 + 8^3 && ;\text{ for } n \geqslant 1 \\ &\leqslant 90n^3 \end{aligned}$$

Therefor this is true when $c = 5$ and $n_0 = 1$.

(b) $1 + 3 + 5 + ... + (2n - 1) \in \mathcal{O}(n^2)$

Proof: We know that $1 + 2 + 3 + ... + (2n - 1) = \sum_{i=1}^{n}(2i - 1)$. Which can be written as:

$$\begin{aligned} \sum_{i=1}^{n} 2i - \sum_{i=1}^{n} 1 &= 2(\frac{n(n+1)}{2}) - n \\ &= n^2 + n - 2n \end{aligned}$$

Thus,

$$n^2 \leqslant n^2 \qquad ;\text{ for } n \geqslant 1$$

So,this is true when $c = 1$ and $n_0 = 1$.

(c) $1 + 2 + 4 + 8 + ... + 2n^2 \in \mathcal{O}(2^n)$

Proof: Let $S(n) = 1 + 2 + 4 + 8 + ... + 2n^2$,

$$\begin{aligned} S(n) &= 2^0 + 2^1 + 2^2 + ... + 2^n \\ 2S(n) &= 2^1 + 2^2 + ... + 2^n + 2^{n+1} \\ &= S(n) - 1 + 2^{n+1} \\ S(n) &= 2^{n+1} - 1 \\ &= 2(2^n) - 1 \end{aligned}$$

Thus,

$$2(2^n) - 1 \leqslant 2(2^n) \qquad ;\text{ for } n \geqslant 1$$

So, this is true when $c = 2$ and $n_0 = 1$.

(d) $1 + \frac{1}{2} + \frac{1}{4} + ... + \frac{1}{2^n} \in \mathcal{O}(1)$

Proof: Let $S(n) = 1 + \frac{1}{2} + \frac{1}{4} + ... + \frac{1}{2^n}$

$$\begin{aligned}
S(n) &= \frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + ... + \frac{1}{2^n} \\
\frac{S(n)}{2} &= \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + ... + \frac{1}{2^n} + \frac{1}{2^{n+1}} \\
&= S(n) - 1 + \frac{1}{2^{n+1}} \\
-S(n)(\frac{1}{2}) &= \frac{1}{2^{n+1}} - 1 \\
S(n) &= 2 - \frac{1}{2^n}
\end{aligned}$$

Therefor, we can see that:

$$2 - \frac{1}{2^n} \leqslant 2(1) \qquad\qquad ; \text{for } n \geqslant 1$$

So,this is true when $c = 2$ and $n_0 = 1$.

Problem 2

(a) We need to show that all the elements in A remains in A after the loop.

(b) Loop-invariant: At the start of each iteration, in the sub-array $A[j...n], A[j]$ is the smallest.

Initialized: At the start, $j = n$ so the sub-array is $A[n]$. Since $A[n]$ is the only element, it is the smallest.

Maintenance: Let $k =$ iteration number. Suppose that $A[j_k] < A[j - 1_k]$, the value of $A[j_k]$ and $A[j - 1_k]$ will then be swapped. The value of $A[j - 1_k]$ is now the smallest since $A[j_k]$ was originally the smallest in $A[j_k...n]$.Therefore, in the next iteration, value of $A[j_{k+1}]$ will be the smallest in $A[j_{k+1}...n]$. If $A[j_k] > A[j - 1_k]$, it means that $A[j - 1_k]$ is now the smallest in $A[j - 1_k...]$. There will be no swapping so in the next iteration, $A[j_{k+1}]$ will be the smallest in $A[j_{k+1}...n]$. Thus, the invariant is maintained.

Termination: The for-loop will terminate at $j = i$. From the loop-invariant, this will give us a sub-array $A[i...n]$ where $A[i]$ is the smallest.

(c) Loop-invariant: At the start of each iteration, the sub-array $A[1...i]$ is sorted

Initialized: At the start i = 1, the sub-array A[1] is already sorted.

Maintenance: Let $i = k$ where k is an iteration of the for-loop. Using invariant from part (b), we know that $A[k]$ is the smallest in $A[k...n]$. And with invariant from part (c), we know that $A[1...k - 1]$ is already sorted. We also know that as the value of i increases, the length of final sub-array in inner-loop decreases. Therefore, we are sure that $A[k - 1]$ must be the smallest value in $A[k - 1, k, ...n]$ the iteration before. Thus, at the end of the iteration, we would have $A[1...k]$ which is sorted.

Termination: The loop will terminate when $i = n$. With our invariant, this will give us a sub-array $A[1...n]$ that is sorted. By observation, the sub-array is our array. Thus, the algorithm gives us a sorted array upon termination.

(d) The worst-case is when the array is sorted in descending order. The running-time would be :

$$\begin{aligned} T(n) &= \sum_{i=2}^{n-1} i \\ &\approx \frac{n(n+1)}{2} \\ &= \mathcal{O}(n^2) \end{aligned}$$

Which is the same as the worst-case running time for insertion sort.

Problem 3

$S(n) = 1^c + 2^c + 3^c + ... + n^c$

(a) $S(n)$ is $\mathcal{O}(n^c + 1)$

Proof:

$$1^c + 2^c + 3^c + ... + n^c \leqslant n(n^c) \qquad ; \text{for } n \geqslant 1$$

So, this is true when constant $= n$ and $n_0 = 1$.

(b)

Proof:

$$\begin{aligned} 1^c + 2^c + 3^c + ... + n^c &\geqslant (\frac{n}{2})^c + ... + n^c \qquad ; \text{for } n \geqslant 1 \\ &\geqslant (\frac{n}{2})(\frac{n}{2})^c \\ &\geqslant (\frac{n}{2})^{c+1} \\ &\geqslant (\frac{1}{2})^{c+1} n^{c+1} \end{aligned}$$

So, this is true when constant $= (\frac{1}{2})^{c+1}$ and $n \geqslant 1$.

Problem 4

(a) Observations: $2 = 2$ bits, $4 = 3$ bits, $8 = 4$ bits, $2^n = n + 1$ bits.

So the big-O is: $\mathcal{O}(\log_2 n + 1) = \mathcal{O}(\log_2 n)$.

(b) The number n is being divided by 2 for each loop. So the numbers of times that n has to be divided by 2 before reaching 1 is $\log_2 n$ and the total iteration is $\log_2 n + 1$. Thus the big-O is $\mathcal{O}(\log_2 n)$.

(c) Each iteration that the code makes, the input size is reduced by half. So we can write the running time as follow:

$$\begin{aligned}
T(n) &= n^3 + (\frac{n}{2})^3 + (\frac{n}{2^2})^3 + (\frac{n}{2^3})^3 + ... + 1 \\
&= n^3 + (\frac{n^3}{8}) + (\frac{n^3}{8^2}) + (\frac{n^3}{8^3}) + ... + 1 \\
\frac{T(n)}{8} &= (\frac{n^3}{8}) + (\frac{n^3}{8^2}) + (\frac{n^3}{8^3}) + (\frac{n^3}{8^4}) + ... + 1 + \frac{1}{8} \\
&= T(n) - n^3 + \frac{1}{8} \\
\frac{7}{8}T(n) &= n^3 - \frac{1}{8} \\
T(n) &= \frac{8n^3}{7} - 7 \\
&= \mathcal{O}(n^3)
\end{aligned}$$

So the running time is $\mathcal{O}(n^3)$.

Problem 5

(a) Function foo() takes in a sorted list, integers m and val. The function first search for an interval of size $m+1$ where the value val could be inside. If such interval is not found, function returns -1. Else, it would then search the interval one by one for the value val. If the value is found, function will return the index of the value, otherwise ,it returns -1.

(b) Lemma 1 : The first while-loop will terminate with the right interval (if exist) for $m \geqslant 1$.

Proof : When $m \geqslant 1$, first while-loop will terminate on 2 conditions. Firstly, the loop will break if an interval where $alist[left] \leqslant val$ and $alist[right] \geqslant val$ since there is a statement checking for this condition. Hence, this break will give us the correct interval and will terminate. If such interval doesn't exist, the value of $left$ will keep increasing until $left \geqslant length$ or $alist[left] > val$. This will then return -1 on line 10 which indicates that val doesn't exist in the array.

Lemma 2 : The second while-loop will return the index of val or -1.

Proof : In second while-loop, it will iterate over the range that we got from the first while-loop one element at a time. If the element is equal to val it will return that index, thus, returning the index of val. However, if val is not in the array, i will keep increasing till $i > r$ or $alist[i] > val$ which will terminate the while-loop and returns -1.

With this 2 lemmas, it can be seen that this code will return the right answer.

(c) The worst-case is when $m = 1$ and the value to find is more than last index. This will give us a running time of:

$$T(n) = n = \mathcal{O}(n)$$

(d) The maximum number of comparison made in first-loop is $\frac{n}{m}$ and the second loop is m. So to find the minimum we take the derivative of function: $\frac{n}{m}+m$.

$$\frac{d}{dm}(\frac{n}{m}+m) = \frac{-n}{m^2}+1$$

To find the minimum, we let : $\frac{d}{dm}(\frac{n}{m}+m) = 0$:

$$\begin{aligned}\frac{-n}{m^2}+1 &= 0\\ \frac{n}{m^2} &= 1\\ n &= m^2\\ m &= \sqrt{n}\end{aligned}$$

Therefore, the $m = \sqrt{n}$ for the minimum comparison.

Problem 6

(a) $\mathcal{O}(n^3)$

(b) $\mathcal{O}(nlogn)$

(c) $\mathcal{O}(3^n)$

(d) $\mathcal{O}(n^3)$

(e) $\mathcal{O}(n^4)$