ICCS313: Assignment 1

Natthakan Euaumpon
natthakan.eua@student.mahidol.edu
September 2019

1: Problem 1

(a)

$$\lim_{n\to\infty} \frac{f(n)}{g(n)}$$

$$= \lim_{n\to\infty} \frac{2n^3 + 75n^2 + 8\log_2 n}{n^3}$$

$$= 3$$

$$3 > 0, f(n) \in \Theta(g(n))$$

$$Then, f(n) \in O(g(n))$$

(b)

$$1 + 3 + 5 + ... + (2n - 1)$$

$$= n^2$$

$$\lim_{n\to\infty} \frac{n^2}{n^2}$$

$$= 1$$

$$1 > 0, f(n) \in \Theta(g(n))$$

$$Then, f(n) \in O(g(n))$$

(c)

$$1 + 2 + 4 + ... + 2^n = p$$

Natthakan Euaumpon
natthakan.eua@student.mahidol.edu

$$2+4+6+...+2^n+2^{n+1}=2p$$

$$p+2^{n+1}=2p+1$$

$$=2^{n+1}-1=2p-p$$

$$=2^{n+1}-1=p$$

$$\lim_{n\to\infty}\frac{2^{n+1}-1}{2^n}$$

$$=2$$

$$2>0, f(n)\in\Theta(g(n))$$

$$Then, f(n)\in O(g(n))$$

(d)

$$\sum_{i=0}^{n}(\frac{1}{2})$$

$$=1$$

$$1+1$$

$$=2$$

$$\lim_{n\to\infty}2$$

$$=2$$

$$2 > 0, f(n) \in \Theta(g(n))$$

$$Then, f(n) \in O(g(n))$$

2: Problem 2

(a)
Loop Invariant, containing initialization and maintenance.

(b)
At the start of each iteration of the for loop in line 2, the sub array $A[1...k-1]$ consists of the elements originally in $A[1...k-1]$ but in sorted order.
Then:
Initialization – This is true, when we start j= 2,A[1..1]is just one single number, which we consider as sorted.
Maintenance – If $A[1..j-2]$ is sorted before, the current iteration will cause $A[1..j-1]$ to be sorted.
Termination – When $j = A.length$ after the termination, We will have $A[1..A.length]$ in a sorted order.
So, the invariant is true. Hence, after the algorithm terminates, this will leave the array A sorted.

(c)
At the start of each iteration, sub array $A[1...k]$ is sorted ans elements in $A[k+1...A.length]$ has the greater or equal value to any element in $A[1...k]$.// Initialization - For i = 1, the invariant is true as there is one element. Which means it is sorted.
Maintenance - The sub array $A[1...j-1]$ is sorted, this mean when we insert the value j which is the smallest number compare with other remaining unsorted element $A[j....A.length]$. As previously computed $A[1...j-1]$ contain only the sorted element which is smaller than the value in $A[j...A.length]$. This means A[1...j] is sorted which means the invariant is preserved.
Termination - A[1...A.length-1] is sorted and both $A[A.length-1, A.length]$ are both larger than elements in $A[1...A.length-1]$. This means array A[1...A.length] is sorted.

(d)
$O(n^2)$

3: Problem 3

(a)
$S(n) = 1^c + 2^c + 3^c + ... + n^c \leq n^c + n^c + n^c + ... + n^c; \forall n \leq 1$
$S(n) \leq n^{c+1}$
$S(n) \leq tn^{c+1}$
$t = 1, \forall n \geq n_0 = 1$

(b)

$S(n) = 1^c + 2^c + 3^c + ... + n^c \geq (\frac{n}{2})^c + (\frac{n}{2})^c + (\frac{n}{2})^c + ... + (\frac{n}{2})^c, \forall n \geq 1$
$S(n) \geq (\frac{n}{2})^{c+1}, \forall n \geq 1$
$S(n) \geq tn^{c+1}$
$t = (\frac{1}{2})^{c+1}, \forall n \geq n_0 = 1$

4: Problem4

(a)
$O(logn) + 1$
$= O(logn)$

(b)
$O(logn)$

(c)
$O(n^3 \cdot logn)$

5: problem 5

(a)
Search the index of the val which is inserted into the function. The variable at first are "val", "m", "alist", "left", "right" and "length". Where "left" and "right" is for the index, the "length" is to make sure that the range is not greater than length-1. The "val", "m", and "alist" is the inserted value. The first while loop (from line 4 - 8) is use to find the range of index that containing "val". For line 4, the loop will run until "left" is out of the array range or when the value is greater than the "val" we are searching for. Both condition means that the function will surely terminate and return -1 as the "val" is not in an array. This condition work because we assume that the array is sorted in an increasing order. For line 5 we wanted to find the value for the variable "right". For line number 6 we check if the value on "alist[left]" is less than or equal to the "val" and "alist[right]" is greater than or equal to the "val". This is because the elem in the array is sorted from small to large, which means in most of the cases the the "left" index need to be smaller than the "right". The only cases where it is possible for "left" to be equal to "right" is when the first elem has the same value as "val" or that it is greater than "val". For line number 7, if we got the range the program will break and skip to line number 9. Else, the program will increase the value of "left" which in turn increase the value of right. This means it shifted the range to find the correct one as the value "alist[right]" in the previous range is smaller the the "val". Then the loop continues. After the loop is finish, the program goes to line number 9. This check if "left" is greater or equal to length or "alist[left]" greater than "val" it will return -1. This is because if "left" is greater or equal to the length, the value is smaller and does not exist in an array. As the index goes from 0 to length-1. For the other condition it means if the "alist[left]" is greater than the "val" the "val does not exist in the array". This is because the "alist[left]" is the smallest possible value. Some time the "right" value can be greater than lenght-1. The line number 11 is to make sure that the largest possible value for "right" is length-1. Then we start searching for the value. So we start the searching at index "i" which is equal to "left". The if condition is there to return the correct index instantly if "i" is equal to "val". However if the "val" is within the range but there is no "val" in the array it will return -1 when the loop is finish.

(b)

There are 2 possible cases.
First case:
This case will happen when "alist" conatain "val". We need to prove 2 loop invariant for this case, one for the first loop and another for the second loop. First loop invariant:
During the first iteration, alist[1...alist.length] is sorted in an increasing order containing the "val".
Initialization: When we start, alist[left] will be smaller than or equal to the "val" and alist[right] will be greater than or equal to "val". As we promise that the "alist" will contain only ascending order numbers, we just need to make sure that the index is not out of range. There will be a case where left and right has equal value and it is both lesser than the "val" it will get into the condition where left will be increase by the value of m. This will result in the increase in the right value.
Maintenance:
If $left < alist.length$ and $alist[left] is \leq val$, the iteration will continue checking until its get the range for the val containing j numbers, where $j \leq m$.
Termination:
When we got the range, where $alist[left] \leq val$ and $alist[right] \geq val$. We know that we might be able to find the index i which has the same value as "val".
So the invariant is true. Hence, after the loop terminate, this loop will left us with the range of numbers containing j integers. Which might contain the "val".
At the start iteration for the second loop, we know that the starting index i is equal to the value left from the previous invariant and that $a[1...alist.length]$ is sorted in ascending order. This means we will search from $alist[i]$ to alist[right].
Then we show:
Initialization:
This is true, when we start the iteration, the loop start from $alist[i]$ and it will check whether $alist[i] == val$ or not. If $alist[i] == val$ the program will immediately return the value i. Else the loop will increase the value of i by 1.
Maintenance:
If $i \leq right$ and $alist[i] \leq val$ the loop will continue searching for the element in the range j containing val.
Termination:
When the val is found which led to the termination of a program, we will have the index i.
So the invariant is true. Hence after the algorithms terminates, this will leave us with the index of the element of alist where "val" is located.
Second case:
This case will happen when the alist is not containing "val".
The loop invariant for the first loop will make sure that if we cannot find val 2 possible case can happen. The first case is when we cannot find the range j, as all of the numbers are less than or greater than the val. This means line 8 will take care for the case where all of the element has less value than the right one. As the if indicates that $left < alist.length$. Another main cases will will led to the second loop is that the first loop contain the possible range j, but when we loop it does not contain "val". this indicate that $left < right$ and $alist[left] \leq val$. The program will terminates with -1 due to the second loop, where it runs from i to $alist.length$. This prove that the program works correctly.

(c)
$O(\frac{n}{m} + m)$

(d)
$\sqrt{n}$

Natthakan Euaumpon
natthakan.eua@student.mahidol.edu

6: Problem 6

(a)
$O(n^4)$

(b)
$O(nlogn)$

(c)
$O(3^n)$

(d)
$O(n^3)$

(e)
$O(n^3)$