Algorithm Assignment2

Xinyu Jiang

September 2018

1

a).$7(n+10)^2 = \Theta(n^2)$
Proof:
$T(n) = 7(n^2+20n+100) = 7n^2+140n+700$
when $n \geq 1$,then T(n)=$7n^2$+$140n$+$700 \leq 7n^2$+$140n^2$+$700n^2 = 847n^2, therefore, T(n) is O(n^2)$
$T(n) = 7n^2+140n+700 \geq 7n^2$, therefore, Ω is $\Omega(n^2)$
Since $O(n^2)$ and $\Omega(n^2)$ are both n^2,so $7(n+10)^2 = \Theta(n^2)$

b).$1000log(n) = O(\sqrt{n})$
Proof:For upper bound, after the insertion point,
$log(n) \leq \sqrt{n}$, thus $1000log(n) \leq 1000\sqrt{n}$,
therefore, the big O for $1000log(n)$ is $O(\sqrt{n})$

c).$10^{20} = \Theta(1)$
Proof:T(n)=10^{20}, then $10^{20} \leq 10^{21}(10^{20} * 10)$,thus for big O it is O(1)
T(n)=$10^{20} \geq 1$,therefore the big Ω is $\Omega(1)$
Since O(1) and $\Omega(1)$ are both 1, $10^{20} = \Theta(1)$

2

a). the worst case is n^2

```
findCommonElement(A, D) :          Cost        Run time
   for i = 0 to length(A)-1 {      C1              n
    for j = 0 to length(D)-1 {     C2              (n(n+1))/2
        if (A[i] == D[j]) {        C3              ((n(n+1))/2)-1
        return TRUE
        }
}
```

T(n)=$C_1 n + C_2 \frac{n(n+1)}{2} + C_3 \frac{n(n+1))}{2}$
Drop all the constant and lower power, the big O for T(n) is O(n^2), $\Theta(n^2)$ is bounded of worst case, so big

b).the best case is the value of the first element of array A and array D are the same, for example A=4,5,6,7, D=4,3,2,1
The worst case is two array do not contain the same element, then, after the function need to go through all the interation, for example, A=1,2,3,4 B=9,8,7,6

c).

```
findCommonElement(int A,int D){
    j=A.length-1;
    for(i=0 to i<A.length-1){
        if(A[i]==D[j]){
return true;
        }else if(A[i]>D[j]){
            i++;
        }else{
            j--;
        }
    }
    return false;
}
```

3

a).

```
SubArraySum(A) {                          Cost     Line
    for i=1 to n                           C1       1
       for j = i+1 to n                    C2   '   2
          B[i,j] = 0                       C3       3
          for k = i to j                   C4       4
              B[i,j] = B[i,j] + A[k]       C5       5
  }
```

The time for Line 1 is n, for line 2 is $\Sigma_{i+1}^{n} \frac{n(n+1)}{2}$, for line 3 is $\Sigma_{i+1}^{n} \frac{n(n+1)}{2} - 1$, for line 4 is for line 3 is $\Sigma_{i+1}^{n} \frac{n(n+1)}{2} - 1$, for line 5 is $\Sigma_{i}^{n} \Sigma_{i+1}^{n} \frac{n(n+1)}{2} - 1$, therefore, the time for T(n)=$C_1 n + C2 \frac{n(n+1)}{2} + C3 \frac{n(n+1)}{2} + C4 \frac{n(n+1)}{2} n + C_4 \frac{n(n+1)}{2} n - 1$,drop all the constant and lower power, the big O is O(n^3)

b). Since there is no best case for this algorithm, the proof for Ω is the same with O therefore the $\Omega(n^3)$ as well, since the O and Ω are the same, there is a tight bound, the $\Theta = n^2$

c).

```
SubArraySum(A) {
    int sum=0;
    for(int i=1 to n){
        sum=A[i]
        for(int j=i+1 to n){
            sum=sum+A[j]
            B[i,j]=sum
        }
    }
}
```

Proof by induction:
Base case: if there is only one element in A, the sum is A, then B[i,j] is equal to A, which is true
Induction Hypothesis: Assume, when the length of A is n, the sum is A[0] to A[n], then B[i,j]=B[i,j-1]+A[j]. For size of A is n+1, the function will go through all the elements in A and add it all up, then store it in B[i,j], thus the sum is A[0] to A[n+1], B[i+1,j+1]=B[i+1,j]+A[j+1] which is true.therefore the algorithm is correct

4

a).

```
findMaxLL(myLL){                    Cost    Time
    int max=0;                       C1      1
    while(myLL!=NULL){               C2      n
        if(myLL.value>max){          C3      n-1
            max=myLL.value;          C4      n-1
        }
        myLL=myLL.next();            C5      n-1
    }
}
```

b).For the algorithm above, T(n)=$C_1 + C_2n + C_3(n-1) + C4_(n-1) + (C_5n)$,drop all constant and lower power, the big O is O(n)
For the Ω, since there is no best case for this algorithm (it has to go through all the elements in the linked list), the proof for Ω is the same with O therefore the $\Omega(n)$ as well, since the O and Ω are the same, there is a tight bound, the $\Theta = n$

c).No, it is not the same, for the worst case, searching and find max are the same, both are O(n), but for best case, find max is $\Omega(n)$ as well, but searching

is $\Omega(1)$, because the target value could be the head of the linked list, therefore, for searching, there is no Θ.

d).Proof by contradiction: Suppose the Θ is the same for searching and find max, then for best case and worst case, the O and Ω are both n. For the best case of searching, if the target element is at the head of the linked list, then the Ω is 1 which is not $\Omega(n)$, therefore, since the best case and worst case for searching is different, there no Θ for searching, therefore, Θ is different for searching and find max.

e). Proof by induction: Base case: if there is only one element in the node, the function will just check if it is equal to the target value
Induction hypothesis: if there is more than one element in the linked list, the fucntion will check if the first one equal to the target value, if it is the target value, return true, then if it is not, check the second one, third one, and so on, until the linked list all been checked, if still can't find it, return false.
Conclusion: Therefore, the algorithm is right.

```
searching(node,value){              Cost    Time
    while(node!=NULL){                C1      n
        if(node.value=value){         C2      n-1
            return true;              C3      n-1
        }
        node=node.next;               C4      n-1
    }
    return false;
}
```

For the algorithm above, T(n)=$C_1n + C_2(n-1) + C_3(n-1) + C_4(n-1)$ drop all the constant and lower power, big O=n
For the best case, if the target element is at the head of the linked list, then the Ω is 1, since O and Ω are not the same, there is no Θ.

f).Loop invariant: The whole linked list has been checked and there is no target value
Initialization: node is not NULL
Maintenance: For the node to NULL, there is no value equal to the target value, then it holds for the loop invariant
Terminate: when node is NULL when it is passed in or node is NULL after the loop which means it go through all the elements of the linked list and there is no target value.