Theoritical Assignment 2

Sahil Dhull
160607

January 27, 2018

Solution 1:
Let p1 and p2 be 2 pointers. Initially both point to the head of Linked List. We increment p1 by 1 and p2 by 2 each time.
If there is no loop in the linked list, then p2 will become NULL before p1 and hence we say no loop. But if there is a loop, then neither p1 will become NULL at any point nor p2 will become NULL, and after certain point, p1 will become equal to p2. So if (p1==p2) and p1!=NULL , p2!=NULL , then we say there is a loop in the linked list.
Consider a case where a loop exists, when p1 enters the cycle, p2 will already be there. Now p2 travels a distance 2 so the gap between p2 and p1 will reduce by 1 each time and hence they finally meet.
Pseudo Code:

```
p1=head
p2=head
flag=0
p1=p1->next
p2=p2->next->next
while(p1!=NULL and p2!=NULL)
    if(p1 equals p2)
        flag=1
        break
    p1=p1->next
    p2=p2->next->next
if(flag equals 0)
    No cycle
else
```

Cycle Exists

Time Complexity:

Let n be the number of nodes in the linked list that are out of loop.
Let x be the number of nodes in the loop (can be 0 in case of no loop).
The initial assignment statements take constant time.
In case of no loop:
Pointer p2 reaches the end of linked list in n/2 steps.
So Time $\propto$ n.
In case of loop:
Loop will end only when p1=p2 and in each execution constant time is required.
Now, till p1 hasn't entered the loop, the program will run for n times. After that p1 and p2 can meet anytime till p1 completes one cycle through the loop.
So in the worst case, it can take x more steps and in the best case it will take just one more case.
So in worst case, Time $\propto$ n+x and in the best case, Time $\propto$ n.
Hence, if number of elements in linked list is x, then
O = Ω = Θ = x.

Solution 2:

1. Part 1:

Description	Time
Memory And Initialization of sum	2
Memory And Initialization of i	2
Memory And Initialization of sum	2·N
Comparison for i	N+1
Comparison for j	$\frac{N\cdot(N+1)}{2}$
Addition and Assignment	$2 \cdot \frac{N\cdot(N-1)}{2}$
Increment of first loop	N
Increment of second loop	$\frac{N\cdot(N-1)}{2}$
Total Time Complexity	$2N^2$+2N+5

2. Part 2:
   [ ] denotes greatest integer and log is base 2.

Description	Time
Memory And Initialization of sum	2
Memory And Initialization of i	2
Memory And Initialization of sum	$2 \cdot ([logN] + 1)$
Comparison for i	$[logN]$+2
Comparison for j	$[logN]$+$2^{[logN]+1} - 1$
Addition and Assignment	$2 \cdot (2^{[logN]+1} - 1)$
Increment of first loop	$2 \cdot [logN]$
Increment of second loop	$2 \cdot [logN]$+$2^{[logN]+1} - 1$
Total Time Complexity	8n+$8 \cdot [logn]$+4

3. Part 3:
   The outer loop executes log_2N times and for each iteration of outer loop the inner loop executes log_2i times where i ranges from 1 to N.
   Time taken by the outer loop=c$\cdot log_2N$+d.
   Time taken by the inner loop is proportional to log_2i.Also i doubles after every iteration.
   Time taken by the inner loop=c1$\cdot(log_21 + log_22 + log_24 + log_28 + ... + (log_2N - 1))$+d1 $=c2 \cdot log_22 \cdot (log_2N)^2 + d2 + c3 \cdot log_2N$ Overall time complexity=$a \cdot (log_2N)^2 + b \cdot (log_2N) + c = O((log_2N)^2)$

4. Part 4:
   The Time complexity is ∞ as i is real(float) and in each loop it is getting halved and hence it will never reach zero, so program will take ∞ time to compute.

Solution 3:

1. f(n)=a·logn and g(n)=log_an
   a·logn=a·loga·$\frac{logn}{loga}$=a·loga·log_an
   So for c≥a·loga and for n>0, f(n)<c·g(n).
   Hence f(n)=O(g(n)).

2. f(n)=7^{4n} and g(n)=$2^{n/11}$
   Suppose f(n) is O(g(n)).
   Then there exists a constant c, such that for all n¿n_0, f(n)≤c·g(n).
   So c≥ $(\frac{7^{4n}}{2^{n/11}})$.
   But $7^{4n} > 2^{n/11}$ and will go on increasing, and hence c will have to increase with n. So our assumption was wrong and hence is not O(g(n)).

3. f(n)=$2^{\sqrt{logn}}$ and g(n)=$\sqrt{n}$
   Let f(n) be O(g(n)).
   So there exists a constant c, such that $2^{\sqrt{logn}} \leq c \cdot \sqrt{n}$.
   Taking log_2 on both sides, $\sqrt{logn} \leq logc + logn/2$
   So for c=2 and n≥2 this is true.
   Hence f(n) is O(g(n)).

4. f(n)=$\sum_{i=1}^{n} \frac{1}{i}$ and g(n)=logn.
   Let f(n) be O(g(n)). Then $\sum_{i=1}^{n} \frac{1}{i} \leq c \cdot logn$.
   $\sum_{i=1}^{n} \frac{1}{i}$=$\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ...\frac{1}{n}$
   $\leq \frac{1}{1} + \frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + ... + \frac{1}{2^k}$ (k=$log_2 n$).
   $\leq 1 + 1 + 1 + 1...$ ($log_2 n$ times)$\leq log_2 n$
   Thus f(n) ≤ c·g(n) for c=1 and n≥10.
   Thus f(n) is O(g(n)).

Solution 4:

1.

2. T(n)=c+T(n/k),k¿1.
   T(n)=c+(c+T(n/k^2))=2c+(c+T(n/k^3))
   This recursion will continue until the term becomes less than 2.
   Continuously unfolding this recursion,
   T(n)=$log_k n \cdot c$+T(1);where T(1)=c.
   T(n)=$(log_k n + 1) \cdot c; soT(n) <= c1 \cdot log_k n$.
   So T(n) is O($log_k n$).

3.

4. T(n)=c· n+T(n/k),k¿1.
   T(n)=cn+(c(n/k)+T(n/k^2))=cn+c(n/k)+(c(n/k2+T(n/k^3))
   Continuously unfolding this recursion we get-
   T(n)=cn+c(n/k)+c(n/k^2)+......+c(n/$k^{log_k n-1}$)+T(1);where T(1)=c as T(x)=c all x¡2.
   So T(n) can be expressed as sum of a geometric progression. So T(n) =cn(1+1/k+1/k2+1/k3+.....1/$k^{log_k n}$) So this sum if equal to (1 - $(1/k)^{log_k n}$)/(1-1/k)=k·(1-(1/n))/(k-1) T(n)=c1(n-1);so T(n)¡= c2n where c1 and c2 are constants. So T(n) is O(n).

Solution 5:

1. False
   f(n) is o(g(n)) if Lim n$->\infty$ $\quad |\frac{f(n)}{g(n)}|$=0.
   Here f(n)=4n+7, g(n)=n.
   So Lim n$->\infty$ $\quad |\frac{f(n)}{g(n)}|$=4.

2. True
   f(n) is o(g(n)) if Lim n$->\infty$ $\quad |\frac{f(n)}{g(n)}|$=0.
   Here f(n)=4n+7, g(n)=n^2.
   So Lim n$->\infty$ $\quad |\frac{f(n)}{g(n)}|$=0.

3. False
   f(n) is ω(g(n)) if Lim n$->\infty$ $\quad |\frac{f(n)}{g(n)}|=\infty$.
   Here f(n)=4n+7,g(n)=n.
   So Lim n$->\infty$ $\quad |\frac{f(n)}{g(n)}|$=4.

4. True
   f(n) is ω(g(n)) if Lim n$->\infty$ $\quad |\frac{f(n)}{g(n)}|=\infty$.
   Here f(n)=4n+7, g(n)=logn.
   So Lim n$->\infty$ $\quad |\frac{f(n)}{g(n)}|=\infty$ (based upon graphs of log as compared to linear graph).

Solution 6:

1.

Solution 7:
The algorithm used is much like merge sort in the sense that it also divides the array in 2 parts, computes the number of inversions in each part, then merges those 2 halves while sorting them and counting the number of cross inversions.
Cross inversion refers to such cases as (i,j) where i is in first half, j is in second half, and ith element is greater than jth element.
Dividing the array into two halves, the recursive calls return the number of inversions of the first half and the number of inversions in the second half respectively along with sorting them. Now we iterate through both the half arrays, comparing element by element, replacing elements, increasing the count of cross inversions accordingly.
Pseudo Code:

```
inv(Arr,start,end)
    if (start equals end)
        return 0
    else
        mid=[(start+end)/2]//here [ ] denote greatest integer
        i1=inv(Arr,start,mid)
        i2=inv(Arr,mid+1,end)
        ic=crossinv(Arr,start,mid,end)
        return i1+i2+ic

crossinv(Arr,start,mid,end)
    Copy Arr[start...mid] to A[1...mid - start + 1]
    Copy Arr[mid + 1...end] to B[1...end - mid]
    n1=mid-start+1
    n2=end-mid
    cross=0,i=1,j=1
    for k=start to end
        if A[i] < B[j]
            Arr[k] = A[i]
            i++
        else
            Arr[k] = B[j]
            j++
            cross=cross+n1-i+1
    return cross
```

Time Complexity:

Let T(n) be the time taken to count the number of inversion in the array of size n.

Now in the function inv, we are dividing an array into two half arrays and calling the same function along with callin the crossinv function.

So T(n)=2·T($\frac{n}{2}$)+Time complexity of crossinv.

Time Complexity of crossinv:

In the first step, to copy it will take $\Theta(n)$ time.

Similarly in second step, $\Theta(n)$ time is required.

Next three steps take $\Theta(1)$ time each.

Loop takes $\Theta(n)$ time.

So Time complexity of crossinv is $\Theta(n)$.

Therefore,

So T(n)=2·T($\frac{n}{2}$)+$\Theta(n)$
Using Master Theorem,
T(n)=$\Theta(nlogn)$.