Theoretical Assignment 2

Amit Yadav 160099

January 27, 2018

Problem 1

Let we are given a pointer to head of a linked list which is to be checked for a cycle. We initialize two pointers with pointing to the head of the linked list.
Pseudo Code:

```
    struct node{
        int val;
        struct node* next;
    }NODE;

    void check_cycle(NODE* head){
        NODE* p_one=head->next;;
        NODE* p_two=head->next->next;

        while(1){
            if(p->one==NULL or p->two==NULL or p_two->next==NULL ){
                return 0; //no cycle
            }
            if(p_one==p_two){
                return 1; //cycle present
            }
            p_one=p_one->next; //shift by one node
            p_two=p_two->next->next; //shift by two nodes


        }
    }
```

Termination

- If linked list doesnot contain any cycle, then code will terminate when p_one or p_two reaches NULL pointer.

- If cycle is present, then the program will terminate when p_one and p_two point to same node. Since, the distance between p_one and p_two decreases by one in every iteration and maximum initial distance can be equal to the size of the cycle, so it's certain that they will meet/point to same node before p_one reaches the end of the cycle.

Time complexities

1. **Big Oh(O):** Since the loop will always terminate before the p_one pointer reaches end of the cycle, so the loop will iterate at most N times (size of the linked list). Hence

$$\text{time} = \text{O(n)}$$

2. **Big omega(Ω):** If the cycle includes all elements/nodes of the linked list, the loop will iterate at N or N+1 times. Else, it will take less then N iteration, therefore

$$\text{time} = \Omega\text{(n)}$$

3. **Big Theta(Θ):** Since time complexity is in both O(n) and Ω(n), implies time is also in Θ(n)

$$\text{time} = \Theta\text{(n)}$$

Problem 2

1. First loop iterates N times. And 2nd nested loop iterates 'i' times for i in range(0,N) with N excluded.

i	0	1	2	3	.	.	.	N-1
j		0	0	0	.	.	.	0
			1	1	.	.	.	1
				2	.	.	.	2
					.	.	.	.
						.	.	.
							.	.
								N-2

So total number of executions of statement

sum+=j;

is

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

2. Our inner loop will run till nearest smaller power of 2 times i.e $2^{floor(\log_2(N-1))}$ times. For ex. if N=16, then $2^{floor(\log_2 15)} = 8$. So i will range in (0..8).

i	1	2	4	8	.	.	.	$2^{floor(\log_2(N-1))}$
j	0	0	0	0	.	.	.	0
		1	1	1	.	.	.	1
			2	2	.	.	.	2
			3	.	.	.	.	.
				7		.	.	.
							.	.
								$2^{floor(\log_2(N-1))}$-1

Therefore,

$$Time = \sum_{i=1}^{\log_2(N-1)} 2^i$$

For ex. if N=16, time = 1+2+4+8 = 15 units = N-1 Hence,

$$\text{time = O(n)}$$

3. Outer loop iterates $\log_2(N-1)$ times. While for every iteration, nested loop runs $\log_2(i)$ times.

i	1	2	4	8	.	.	.	N
j		1	1	1	.	.	.	1
			2	2	.	.	.	2
				4	.	.	.	4
					.	.	.	.
						.	.	.
							.	.
								2^{-1}

$$Time = \sum_{i=1}^{\log(N)-1} i$$

i.e

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

4. Let our computer can store smallest positive real number = y. Then after x iterations,

$$i = \frac{N}{2^x} > y$$

$$\log N = \log y + x$$

$$x = O(\log N)$$

Hence, $time = O(\log n)$

Problem 3

1. $a \times \log n \quad and \quad \log_a n$

$$f(n) = a \log n$$
$$g(n) = \log_a n = \frac{\log n}{\log a}$$

Let

$$c \times \frac{\log n}{\log a} \geq a \times \log n$$

It holds for $c = 2 \times a \times \log a \; and \; n \geq 1$
Hence,

$$f(n) = O(g(n)$$

2. $7^{4n} \quad and \quad 2^{n/11}$
Let

$$c \times 2^{n/11} \geq 7^{4n}$$
$$c \geq \frac{7^{4n}}{2^{n/11}} \geq \frac{7^{4n}}{2^n} \geq \frac{7^{4n}}{7^n} = 7^{3n}$$

This means, when $n \to \infty$, c should be infinity, which is not possible. Therefore

$$f(n) \neq O(g(n))$$

3. $2^{\sqrt{\log n}}$ and $\sqrt{n}$

$$2^{\sqrt{\log n}} = 2^{\frac{\log n}{\sqrt{\log n}}} = (2^{\log n})^{\frac{1}{\sqrt{\log n}}} = n^{\frac{1}{\sqrt{\log n}}}$$

Now,

$$f(n) = n^{\frac{1}{\sqrt{\log n}}} \quad and \quad g(n) = \sqrt{n}$$

Clearly,

$$c \times g(n) \geq f(n) \; \forall n \geq 2^4 \; and \; c = 1$$

Therefore,

$$f(n) = O(g(n)$$

4. $\sum_{i=1}^{n} \frac{1}{i} \quad and \quad \log n$
Both functions are strictly increasing with decreasing slope. Let's see which of them is increasing faster. We calculate $f(n+1) - f(n) \; and \; g(n+1) - g(n)$

$$f(n+1) - f(n) = \frac{1}{n+1}$$
$$\frac{1}{n+1} \leq g(n+1) - g(n) \leq \frac{1}{n}$$

because $\frac{d(\log n)}{dn} = \frac{1}{n}$ and it's slope is decreasing.
Now, clearly $g(n)$ is increasing faster than $f(n)$. Therefore,

$$f(n) = O(g(n))$$

Problem 4

1. $T(n) = c + T(n/k)$

$$T(n) = c + T(n/k)$$
$$T(n) = c + c + T(n/k^2)$$
$$T(n) = c + c + ... + c_{i-times} + T(n/k^i)$$
$$T(n) = c \times \log_k n + T(1)$$

Therefore, time complexity of T(n) is

$$time = O(\log_k n) = O(\log n)$$

2. $T(n) = c + T(n^{1/\alpha})$

$$T(n) = c + T(n^{1/\alpha})$$
$$T(n) = c + c + T(n^{1/\alpha^2})$$
$$T(n) = c + c + c + ... + c + T(n^{1/\alpha^i})$$
$$T(n) = c \times \frac{\log(\log_2 n)}{\log \alpha} + T(x)$$

Because $n^{1/\alpha^i} = 2$

$$T(n) = c \times \frac{\log(\log_2 n)}{\log \alpha} + c$$

Therefore. time complexicity of T(n) is

$$time = O(\log \log n)$$

3. $T(n) = c \times n + T(n/k)$

$$T(n) = c \times n + \frac{c \times n}{k} + \frac{c \times n}{k^2} + ... + \frac{c \times n}{k^i} + T(n/k^i)$$
$$T(n) = c \times n + \frac{c \times n}{k} + \frac{c \times n}{k^2} + ... + \frac{c \times n}{k^i}_{\log_k n-times}$$
$$T(n) = cn \times (1 + 1/k + 1/k^2 + ...)_{\log_k n-times}$$
$$T(n) = cn \times \frac{(1 - k^{\log_k 1/n})}{1 - 1/k}$$
$$T(n) = O(n)$$

Problem 5

1. 4n+7 is o(n) : Flase
   For 4n+7 to be in o(n), $4n + 7 < K \times n$ for all $k > 0$ and some $n > n_0$, which is not the case here. If k=1, we can't find any n_0 which satisfy this condition.

2. 4n+7 is $o(n^2)$: True
   $4n + 7 < k \times n^2$ for all $k > 0$ and $n > n_0$ since

$$\lim_{n\to\infty} \frac{4n+7}{n^2} = 0$$

3. $4n + 7\ is\ \omega(n)$: Flase
   Similiar to first part. We can't find n_0 such that $4n + 7 > k \times n \forall n > n_0\ and\ k = 5$ or greater.

4. $4n + 7\ is\ \omega(\log n)$: True

$$\lim_{n\to\infty} \frac{4n+7}{\log n}\ is\ \infty$$

   Therefore, $\log n$ is an loose lower bound of 4n+7.

Problem 6

1. Depends on the values of c and k
   (a) $f(n)$ is $O(g(n))$: Possible if $c \le 1$ and $k > 1$
   (b) $f(n)$ is $\Omega(g(n))$: True for all values of $c > 1$
   (c) $f(n)$ is $\Theta(g(n))$: Only possible if $c = 1$ and $k = 0$

2. $f(n) = \log_2 n$, $g(n) = ln(n)$:
   (a) $f(n)$ is $O(g(n))$: True. $c \times \log_e n \ge \log_2 n$ for $c \ge \log_2 e$ and $n \ge 1$
   (b) $f(n)$ is $\Omega(g(n))$: True. $c \times \log_e n \le \log_2 n$ for $c \le \log_2 e$ and $n \ge 1$
   (c) $f(n)$ is $\Theta(g(n))$: Since $f(n)$ is in both $O(g(n))$ and $\Omega(g(n)$, so $f(n)$ is in $\Theta(g(n))$.
   This can also be done by $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ which is finite.

3. $f(n) = n^2 \log_2 n$, $g(n) = n \log_2 n^3$

$$g(n) = n \log_2 n^3 = 3n \log_2 n$$

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

   Therefore,

$$f(n) = \Omega(g(n))$$

Problem 7

Instead of trivial iterative solution, we can use divide and conquer method. We follow the same algorithm of merge sorting and also keep counting the number of inversions.
Pseudo code:

```
int count=0; //number of inversions
void sort(int *a,int left, int right){
    if(left<right){
        mid=(left+right)/2;
        sort(int *a,left,mid)
        sort(int *a,mid+1,right)
        merge(int *a,left,mid,right)
        return;
    }
    else return;
}

merge(int *a,left,mid,right){
    int i=left, j=mid+1, k=0;
    int b[right-left+1];
    while(i$\leq$ mid && j$\leq$ right){
        if(a[i]<a[j]){
            b[k++]=a[i++];
        }
        else{
            count=count+(mid-i+1); //if element of left array
// is larger, then there will be (size_of_array-i) inversions
            b[k++]=a[j++];
        }
    }
    while(i<=mid) b[k++]=a[i++];
    while(j<=right) b[k++]=a[j++];

    for(k=0,i=left;k<right-left+1;k++){
        a[i++]=b[k];
    }
    return;
}
```

Time comlexity:

$$T(n) = 2T(\frac{n}{2}) + n$$

$$T(n) = 2[2T(\frac{n}{4}) + \frac{n}{2}] + n$$

$$T(n) = 2^i T(\frac{n}{2^i}) + in$$

$$T(n) = 2^{\log_2 n} T(1) + n \log_2 n$$

$$T(n) = nT(1) + n \log n$$

$$\mathbf{T(n)=O(n}\log n)$$