CS 575 - A0 Assignment 1

Submitted By -
BHUVAN CHADHA
B00815783

Q1. We are given two running times of two algorithms respectively. We need to find the smallest value of n, so that running time $100n^2$ is greater than 2^n.

i.e, $100n^2 < 2^n$.

This expression can easily be evaluated by trying various values of n and finding out the smallest value just after getting this expression true

$n = 5 \Rightarrow 100(5)^2 < 2^5$

$\Rightarrow 2500 < 32$ (false)

$n = 10 \Rightarrow 100(10)^2 < 2^{10}$

$\Rightarrow 10,000 < 1024$ (false)

$n = 15 \Rightarrow 100(15)^2 < 2^{15}$

$\Rightarrow 22,500 < 32768$ (True).

We will try one value smaller than 15.

$n = 14 \Rightarrow 100(14)^2 < 2^{14}$

$\Rightarrow 19,600 < 16,384$. (False)

Hence, the smallest value when $100n^2 < 2^n$ becomes true is $\underline{\underline{n = 15}}$.

Q 2. (a)

```
Line 1 : for (i=0; i<n^3; i++)
         {
Line 2:      for (j=0; j< n^4/5; j++)
             {
Line 3 :         Count Me
             }
         }
```

Count for Line 1 = n^3

Count for Line 2 = $n^3 \times \frac{n^4}{5}$

Hence, Count for the instruction Count Me = $n^3 \times \frac{n^4}{5} \Rightarrow \underline{\underline{\frac{n^7}{5}}}$

(b)

```
Line 1 : j = 2
Line 2 : while (j > 0) {
Line 3 :     Count Me
Line 4 :     j = j - 2; }
```

Since n is assigned to j, and in Line 4 $j = j-2$, Hence, Statement CountMe will have the count of $\frac{n}{2}$.

(c) Line 1: $j=1$
Line 2: while $(j<n)$ {
Line 3: CountMe
Line 4: $j = j*2$; }

Here, Line 4 will have the count $\log_2 n$. Hence, CountMe = $\log_2 n$.

Q3. Given an array X[1.. n]. To compute Aarray A[X[0] ... X[i]] for i = 0... n-1.

Prefix Average $A[i] = \frac{1}{i+1} \sum_{j=0}^{i} X[j]$

Algorithm is as follows -

```
prefixAvg (X, n)
{
    int A[], i, sum = 0;
    for (i=0 to n-1)
    {
```

```
        Sum = Sum + X[i];
        A[i] = Sum / (i+1);
    }
    Return A;
}
```

In the above algorithm, 'for loop' has time complexity $(n+1)$,
The 'Sum Statement' has the time complexity $(n-1)$
Also, the statement for Array A[i] will have the time complexity $(n-1)$.

For barometer operations, (+ and /), the time complexity can be calculated as

$T(n) = (n-1) + (n-1)$

$\Rightarrow \underline{2n-2} \rightarrow$ This is in $\Theta(n)$.

Q 5. (1) We are required to partition a list, into two sublists, each of size $n/2$, such that the difference between the sums of integers in the two sublists is maximized.

This can be done using a simple technique by sorting the list in ascending order

and then dividing the list in two Sublists. The first list will hold all the Small elements and the second list will hold all the elements greater than the ones in Sublist 1.
The Sorting will be done using Merge Sort because it's time Complexity is, $\Theta(n \log n)$ for the worst case as well as for best case.

```
Algorithm MergeSort (low, high)
    {
      if (low < high)
        {
          mid = (low + high)/2;

          MergeSort (low, mid);
          MergeSort (mid+1, high);

          Merge (low, mid, high);
        }
    }
Merge(int A[], int B[], int m, int n)
{
  int i=1, j=1, k=1;
    While (i<=m && j<=n)
    {
        if (A[i] < B[j])
```

```
        int C[k++] = A[i++];
    else
        C[k++] = B[j++];
    }
for(; i<=m; i++)
{
    C[k++] = A[i];
}
for(; j<=n; j++)
{
    C[k++] = B[j];
}
}
```

// Now, we have list C[] as the sorted array
// and we have to divide it into two Sublists

```
Divide(int S1[], int S2[])
{
    int C[n], i;
    for(i=0; i<n/2; i++)
    {
        S1[i] = C[i];        // Sublist 1
    }
    for(; i<n; ++i)
    {
        S2[i-n/2] = C[i];    // Sublist 2
    }
}
```

Time Complexity for MergeSort, $T(n) = 2T(\frac{n}{2}) + \Theta(n)$

Also, time Complexity for the Divide() function will be $T_2(n) = \frac{n}{2} + \frac{n}{2} \Rightarrow n$

Hence, total time Complexity of the whole algorithm will be

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

$$\Rightarrow n \log n + n$$

This is of the order $O(n)$.

Q5. (2) We are required to partition a list into two sublists, each of size $n/2$, such that the difference between the sums of integers in the two sublists is minimized.

```
Algorithm Nearest(int A[], Bool used[], int index,
                                              int n)
{
    int minIndex = -1;
    int min = maximum(A) - minimum(A);
    for (int i=0; i<n; i++)
    {
```

```
        if (i == index) Continue;
        if (used[i] == 1) Continue;

        int diff = abs( A[index] - A[i])
        if (diff < min)
        {
            min = diff;
            minIndex = i;
        }
    }
}
Return (minIndex);
}
void Partition ( int A[], int n, bool used[],
                 int par[])
{   int left, right; int sumL, sumR;
    for (int i=0; i<n; i++)
    {
    if (used[i] == 1) Continue;

    int j = Nearest ( A, used, i, n);
    used[i] = 1;
    used[j] = 1;
    if (sumL(left) <= sumR(right))
    {
        par[left++] = max(A[i], A[j]);
        par[right--] = min(A[i], A[j]);
    }
```

```
else
{   par [left++] = min(A[i], A[j]);
    par [right--] = max(A[i], A[j]);
}
}
}
```

The time complexity of this algorithm is of the order $O(n^2)$.

Q4). There are 30 Students in the University Ceremony. We are required to find out the students with tag "A". We are given that there are 5 Students with Tag "A". So, one Solution can be, that if a student sees 4 other Students with "A", he will know that he is the 5th one with tag "A" and hence, can proceed to collect his award.

Q6. Given Algorithm :-

```
int any-equal(int n, int A[][])
{
    Index i, j, k, m;
    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
            for (k=1; k<=n; k++)
                for (m=1; m<=n; m++)
                    if (A[i][j] == A[k][m] &&
                        !(i==k && j==m))
                        return 1;
    return 0;
}
```

(1) The Best Case will be on the condition when the first loop runs only once, i.e, $i=n$ in the first run itself.

Best Case Time Complexity = $\underline{O(1)}$.

(2) The Worst Case will be on the condition when all the four loops run from 1 till n.

Worst Case Time Complexity = $\underline{O(n^4)}$.

Q7. We are give two functions,
$6n^2 + 20n + 1 \in O(n^3)$ and,
$6n^2 + 20n + 1 \notin \Omega(n^3)$.

- Now, according to the definition of Big-Oh (O), the function $f(n) = O\ g(n)$, iff $\exists$ positive Constants c and n_0, such that

$$f(n) \leq c * g(n) \quad \forall \quad n \geq n_0.$$

Let $6n^2 + 20n + 1 = f(n)$

So, $6n^2 + 20n + 1 \leq c * g(n)$
$\Rightarrow 6n^2 + 20n + 1 \leq 6n^3 + 20n^3 + n^3$

$\Rightarrow 6n^2 + 20n + 1 \leq 27n^3$ ($c = 27$ & $g(n) = n^3$)
This expression is true for all $n \geq 1$ values

Therefore, $6n^2 + 20n + 1 \in O(n^3)$.

- Now, ~~we are~~ according to the definition of Big-Omega, the function, $f(n) = \Omega\ g(n)$, iff $\exists$ positive Constants c and n_0 Such that $f(n) \geq c * g(n) \quad \forall\ n \geq n_0$.

Let $6n^2 + 20n + 1 = f(n)$.

So, $6n^2 + 20n + 1 \geq 6n + 20n + n$

$\Rightarrow$ $6n^2 + 20n + 1 \geq 27n$

(Here, $c = 27$ and $g(n) = n$)

This statement is true for all $n \geq 1$ values.

Therefore, $g(n) \in \Omega(n)$ but doesn't hold true for $g(n) \in \Omega(n^3)$.

Hence, we can infer here that,
$6n^2 + 20n + 1 \in O(n^3)$ but
$6n^2 + 20n + 1 \notin \Omega(n^3)$.

Q9. We are required to arrange all white balls together followed by all black balls.
This can be done with a simple algorithm of Bubble Sort.
Let Black Ball = '1' and White Ball = '0'.
We are given the length of Array A = 2n.
Hence, running the loop from '2n-1' till 1 (i.e., in reverse order).
Comparing the other index (Say j), starting from the first position, (i.e., 0), if j finds any element (Say 0), ~~greater~~ smaller than the previous element (Say 1), the loop will run and the elements will be swapped

This will be the execution procedure until all the small elements (all 0s) are before all the large elements (all 1s)

Time Complexity for this algorithm will be $T(n) = O(n^2)$ as there will be two loops.

Q10. We are given $g(n) = O\ f(n)$
We are required to prove $f(n) + g(n) = \Theta f(n)$

Now, since, $f(n) + g(n) = \Theta(n)$,
and $g(n) = O\ f(n)$, — (given)

$\therefore f(n) + O\ f(n) = \Theta(n)$

Which shows, $f(n)$ is asymptotically tight bound for $f+g$.

Hence, $f(n) + g(n) = \Theta\ f(n)$.

Q8. Given Pseudocode –

```
MoreAbove(list, average, N)
    CountAbove = 0
    for j=1 to N do
        if list[j] > average then
            CountAbove = CountAbove + 1
            if CountAbove > N/2 then
                return true
    return false.
```

The Best Case count will be $\left(\frac{N}{2}+1\right)2$.

These should be $\frac{N}{2}+1$ elements which are greater than the average. So, the first $\frac{N}{2}+1$ elements will make the if condition true.

The Worst Case count of the algorithm is $N+\left(\frac{N}{2}+1\right)$, i.e., the average of the list should be between $\frac{N}{2}$ and $\left(\frac{N}{2}+1\right)$, which means, the if condition, if count#Above $> \frac{N}{2}$ will run $N+\left(\frac{N}{2}+1\right)$ times.