(1) Assignment no 1

NAME :- NIHAR MUNIRAJU.
ID :- 2072857.
email :- nmuniraj@depaul.edu

1) a) $f(n) = n-100$ and $g(n) = n-200$.

Ans Let us first consider $f(n)$ & $g(n)$ as functions

$\Rightarrow f(n) = n-100$.

$\therefore f(n) = \theta(n)$.

$\Rightarrow g(n) = n-200$.

$\therefore g(n) = \theta(n)$.

$\therefore n-100 = \theta(n-200)$

$\Rightarrow$ After the observation of two functions we get "$f = \theta(g)$" $\Rightarrow$ BigTheta.

b) $f(n) = n^{1/2}$ and $g(n) = n^{2/3}$.

Ans Let us consider functions $f(n)$ & $g(n)$ as.

$\Rightarrow f(n) = n^{1/2}$.

$\therefore f(n) = \theta(n^{1/2})$

$\Rightarrow g(n) = n^{2/3}$

$\therefore g(n) = \theta(n^{2/3})$.

$\therefore n^{1/2} = O(n^{2/3})$

$\Rightarrow$ After the observation of difference b/w the powers when compared to the growth rate we get "$f = O(g)$" $\Rightarrow$ Big-O.

c) $f(n) = 100n + \lg n$ & $g(n) = n + (\lg n)^2$

Ans $\Rightarrow f(n) = 100n + \lg n$

$\Rightarrow f(n) = \theta(100n, \lg n)$.

$\Rightarrow f(n) = \theta(n)$.

$\Rightarrow g(n) = n + (\lg n)^2$

$\Rightarrow g(n) = \theta(n, \lg n^2)$

$\Rightarrow g(n) = \theta(n)$.

$\Rightarrow$ After comparsion of both the f'n we get "$f = \theta(g)$" $\Rightarrow$ BigTheta.

$\therefore 100n + \lg n = \theta(n + (\lg n)^2$

(d) $f(n) = n \lg n$ and $g(n) = 10n \lg(10n)$.

$\Rightarrow f(n) = n \lg n$

$\Rightarrow f(n) = \Theta(n \lg n)$

Similarly $g(n) = 10n \lg(10n)$.

WKT $\lg(ab) = \lg a + \lg b$

$\therefore 10n^* (\lg(10) + \lg n) \Rightarrow$ From the property.

$\Rightarrow g(n) = 10n^* (1 + \log n)$ From $\lg 10 = 1$.

$\Rightarrow g(n) = \Theta[10n, 10n \lg n)$

$\Rightarrow g(n) = \Theta(n \lg n)$.

$\therefore n \lg n = \Theta(10n \lg(10n))$

$\therefore$ From both the function, we get "$f = \Theta(g)$" $\Rightarrow$ Big Theta.

(e) $f(n) = 10 \lg n$ & $g(n) = \lg(n^2)$.

$\Rightarrow f(n) = 10 \lg n$

$f(n) = \Theta(\lg n)$.

Similary $\Rightarrow g(n) = \lg(n^2)$

W.K.T Using the property,

we get, $g(n) = 2 \lg(n)$.

$\Rightarrow g(n) = \Theta(\lg n)$

$\therefore$ From both the functions being compared we get "$f = \Theta(g)$" $\Rightarrow$ Big Theta.

$\therefore 10 \lg n = \Theta(\lg(n^2))$.

(f) $f(n) = n^2 / \lg n$ and $g(n) = n(\lg n)^2$

(g) $f(n) = n^{0.1}$ and $g(n) = (\lg n)^{10}$.

$\Rightarrow f(n) = n^{0.1}$

$f(n) = \Omega(n^{0.1})$.

Similaly $\Rightarrow g(n) = (\lg n)^{10}$.

$g(n) = \Omega(\lg n)^{10}$.

Hence "$f = \Omega g$" $\Rightarrow$ Big Omega

$\therefore n^{0.1} = \Omega \lg n^{10}$.

h) $f(n) = \sqrt{n}$ & $g(n) = (\lg n)^3$

$\Rightarrow f(n) = \sqrt{n}$

$\Rightarrow f(n) = \Omega(\sqrt{n})$

Similary $\Rightarrow g(n) = (\lg n)^3$.

$= \Omega(\lg n)^3$.

$\therefore f(n)$ is more better than $g(n)$ = Big Omega "$f = \Omega(g)$".

$\sqrt{n} = \Omega(\log n)^3$

i) $f(n) = n2^n$ and $g(n) = 3^n$.

$\Rightarrow f(n) = n2^n$

$\Rightarrow f(n) = \Theta(n2^n)$.

Similary $\Rightarrow g(n) = 3^n$.

$\Rightarrow g(n) = \Theta(3^n)$.

$\therefore$ functions $2^n$ is dominated by $3^n$ hence. "$f = O(g)$".

$\therefore n2^n = O(3^n) \Rightarrow$ Big O

j) $f(n) = 2^n$ and $g(n) = 2^{n+1}$.

$\Rightarrow f(n) = 2^n$

$f(n) = O(2^n)$.

$\Rightarrow g(n) = 2^{n+1}$.

$= O(2^n \times 2^1)$.

$\therefore f = O(g)$.

$\therefore$ function of $2^n$ is dominated by $2^n \times 2^1$ "$f = O(g) \Rightarrow$ Big O.

(2)

⇒ The two arrays are been sorted in a way to search and keep the track of both the iterations for the collections of X for Nuts [1... n] and similarly for Bolts [1... n].

⇒ To Calculate the two sorted arrays. Lets us consider a, b where we first try to find the similar identical numbers in both the arrays in the loop.

⇒ Next we try to check if nuts Y Bolts, then we'll check the smaller number of the array weather if it is Nuts. if it is the Bolts will increase and it becomes bigger and we can compare and check the Loop and stop it if it is True.

⇒ Next we have to check the Overlap condition because of the time taken it might or might not match. If loops to be used.

⇒ We get to know X for Nuts is smaller then we again increment to match Y for Bolts and check the range of the Integers to match the size of both Nuts & Bolt at a certain point.

Psuedocode :—

```
a, b ← 1;
while a < n and b < n do
    if Nuts[x] = Bolts[y] then
    return true;
    else if Nuts[x] > Bolts[y] then
      y = y+1;
    else
      x = x+1;
  end
end return False;
```

3) a) Assuming that the A is sorted has an array of two containers
⇒ (A[], ar_size, sum). Initalizing the size of the two vivid index variables to find the candidate x & y in the sorted array

⇒ There are two ways to solve the problem by using the array an initializing first to the left most index: l=0 and r. Initializing the second element to the rightmost index:

r = ar_size-1.

⇒ After considering this loop we have to send it one more loop where l<r. Lets us take three conditions. if (A[l] +A[r]==sum) then we can return true. Else if (A[l]+ A[r]<sum) then increment l++. Else we can decrement r--.

⇒ If the containees in the entire array is return to 0 then. the time complexity O(N) is traversing the entire array once only.

3 b) The complexity is reduced to $O(n^2)$ for sorting the array first and then using the sorted array as the input.

⇒ As we insert the element A[i] from where i is 0 to the size of -2. After using the 3 sum way of using the first element of triplets, we find the other two elements using the sorting of the array.

3b

Pseudocode:—

```
bool find3Numbers (int A[], int arr_size, int sum)
{
  int l, r;  // Sort the elements.
  Sort(A, A+arr_size);
  for (int i=0; i<<arr_size-2; i++)
  {
  l=i+1.
  r= arr_size-1; // index of the last element.
  while (l<r)
  {
  if (A[i] + A[l]+ A[r]==sum).
    {
  printf(" 3 elements are %d, %d, %d", A[i], A[l], A[r]);
   return true;
    }
else if (A[i]+A[l]+A[r]<sum).
    l++;

else
    r--;
    }
    } // If we cannot find return false.
 return false;
  }.
```

(4) ⇒ Yes, Pinocchio is right way to solve this problem. Since the array is sorted A we have to map integers to the hash function. As told in the class the hash functions can be used to find the solution for these kind of Problems.

⇒ By using hash function we will be able to find the Overall Time Complexibity by avoiding the problem of collosion.

⇒ The two integess are paired and are sent to to hash table. indivisally by using the two Sum Algorithm methord, where A[1...n] is stored in hash Table as specified in class.

⇒ Hence Both the size of an Array is been checked for each pair. by using the methord we will be able to find weather the array takes O(n) time.

Pseudocode :—

```
for j ← 1 to n do
    x ← 1000 − A[j];
    if x in A[1 .. j] then
        return true;
    end;
end.
else return false;
```