ASSIGNMENT - 1.

1> ALGORITHM :

i] DIFFERENCE B/W THE INTEGERS IN THE SUBLISTS IS MAXIMIZED :

Solution :

* sort the list in ascending order.
* Divide the array into two equal halves.
* calculate the sum of each array.
* Difference b/w the sum of each array will be Maximum.

example

[4, 2, 1, 3, 8, 5]

sorted → [1, 2, 3, 4, 5, 8]

A = [1 2 3] $\xrightarrow{\Sigma}$ 6

B = [4 5 8] $\xrightarrow{\Sigma}$ 17

Difference ⇒ 17 - 6 = 11. // Maximum.

ALGORITHM

```
for (i=0; i<n; i++) {
    Get the elements in Array
    :
}
// sort → Ascending
for (i=0; i<n; i++) {
  for(j=0; j<n; j++) {
    if (a[i] < a[j])
    {
        temp = a[i];
        a[i] = a[j];
        a[j] = temp
    }
// splitting into two equal halves
    if (count %. 2 == 0)
        count_A = count / 2;
  for(i=0; i<count_A; i++) {
        split_A[i] = a[i] ;          // split A
    }
```

```
for(i=0; i<cart_B; i++) {
    split_B[i] = a[i];
}
// sum of each Array
sum_A = 0;   sum_B = 0;
for(i=0; i<count; i++) {
    sum_A = sum_A + split_A[i];
    sum_B = sum_B + split_B[i];
// Difference -
Difference = sum_A - sum_B;
```

Then the difference value will be maximum.

TIME COMPLEXITY :

Since we used for loop (nested) only once, it takes n x n times

Time complexity = $O(n^2)$ //.

1> ii] DIFFERENCE BETWEEN THE INTEGERS IN THE SUBLISTS IS MINIMIZED:

Solution:

* Sort the array in ascending order.
* Place first two elements in two different array.
* Then take the next two elements and place the highest among the two in first and lowest in second.
* Calculate the sum, and place the next element based on this sum of the array. Follow the steps till the end of the array.
* Calculate the sum of each array.
* Determine the difference between each array.

Example:

A = [6 8 9 5 4 3 2 7]

sort ⇒ [2 3 4 5 6 7 8 9]

A

[2]

[2 5] ⇒ 7

[2 5 6] ⇒ 13

:

B

[3]

7 ⇐ [3 4]

14 ⇐ [3 4 7

:

Split - A = [2 5 6 9] Σ ⇒ 22

① ② ③ ④

Split - B = [3 4 7 8] Σ ⇒ 22

Difference = SplitA - Split-B = 0

= Minimum.

ALGORITHM

for (i=0; i<n; i++) {

Get the elements in Array

:

}

// sort → ascending order.

for (i=0; i<n; i++) {
for (j=0; j<n; j++) {
if (a[i] < a[j+1]) {

temp = a[i];
a[i] = a[j+1];
a[j] = temp;

}

// problem and adding next element

```
next_element ( int A[], int n, int i){

    split_A[0] = a[0];
    split_B[0] = a[1];

sum_A = 0;   sum_B = 0;

for (i=2; i<n; i++) {

    if ( a[i] + sum_A < sum_B) {

        split_A[i] = a[i];
    }
} else { split_B[i] = a[i]; }
for (i=0; i<n; i++) {

    sum_A = sum_A + split_A[i];
    sum_B = sum_B + split_B[i];
}

Difference = sum_A - sum_B;

        = Minimized //.
```

TIME COMPLEXITY = $O(n^2)$ //.

As we used nested for loop once, which takes $n \times n$.

2> 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;
}
```

* BEST CASE : when n=1,

Time complexity = O(1)

* WORST CASE : when it loops 'n' times

since it has four nested for loops, it takes n x n x n x n times the time

Time complexity = O(n^4).

* It's efficiency can be improved by avoiding nested for loops.

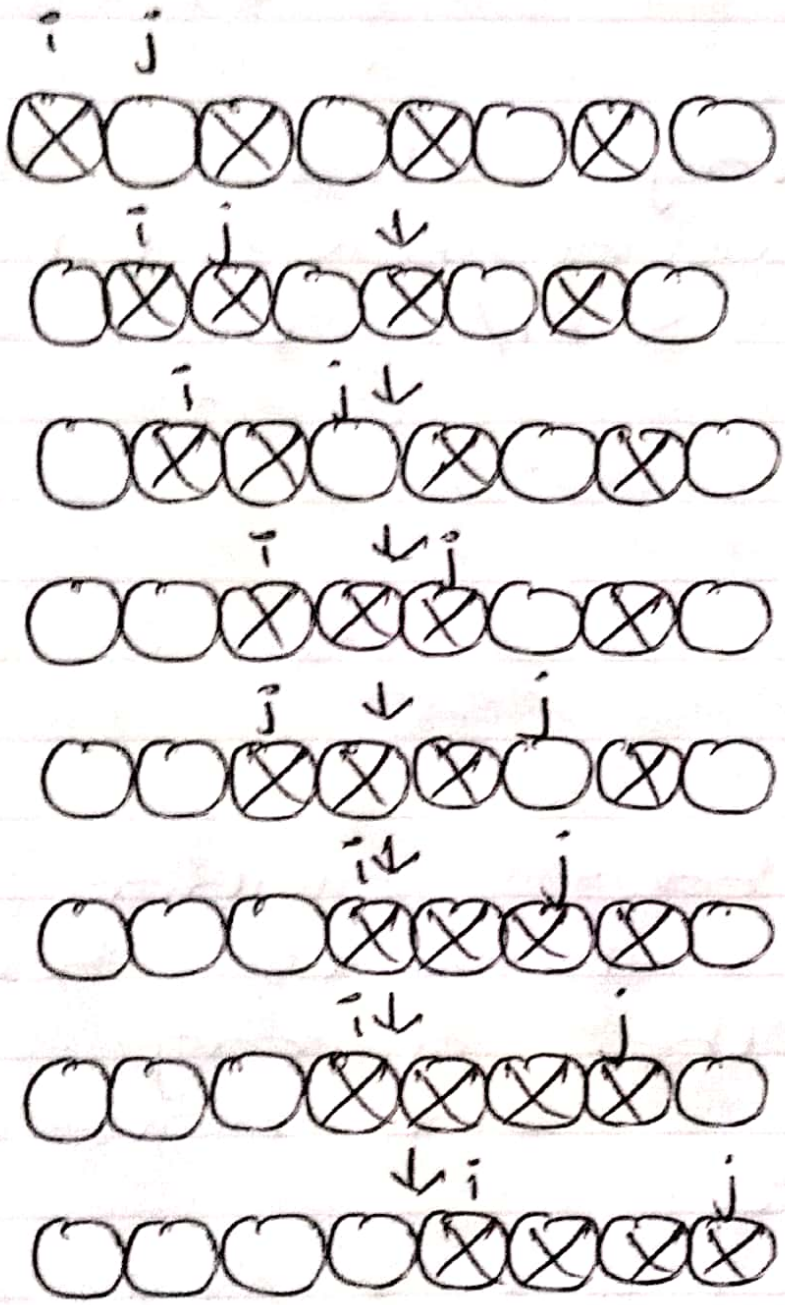
3> BRUTE - FORCE ALGORITHM :

ALGORITHM :

```
for (i=0; i<n; i++) {
  for (j=1; j<n; j++) {
    if (a[i] == "light")
      { swap (a[i] & a[j]);
      }
    else { i = i-1; }

swap (a, b) {
    temp = a;
    a = b;
    b = temp;
}
```

example



4> PROG:

```
MoreAbove (list, average, N)
count Above = 0
for j = 1 to N do
    if list[j] > average then
        count Above = count Above + 1
        if count Above > N/2 then set true

return false
```

BEST CASE :

→ for the best case solution the first $N/2 + 1$ elements should be greater than the average. As the maximum of count Above gives the best case.

Time complexity = $O(N/2 + 1)$

WORST CASE :

→ However one element will be less than the average, so the worst case will be when it traverse through all elements ~~except one~~.

Time complexity = $O(N * 1) \Rightarrow O(N)$

5> a] Line 1 : for(i=0; i<n²+1; i++) {

Line 2 : for(j=0; j<=n³/5; j++) {

Line 3 : count me }}

→ at line 1, it will get executed for $(n^2+1)+1$ times

$\Rightarrow n^2+2$

→ at line 2, it will get executed for $(n^3/5+1)\ n^2$

$\Rightarrow n^5/5 + n^2$

so, count me will get executed for

$n^2 \times n^3/5 = n^5/5$ times.

b] Line 1 : $j = n^2$

Line 2 : while(j>0) {

Line 3 : count me

Line 4 : j = j - 2; }

Since the value of j is decremented by 2 times ($j = j - 2$) countMe will get executed for $n^2/2$ times

$$\text{Time} = O(n^2/2) \begin{cases} \text{odd} = \frac{n^2+1}{2} \\ \text{Even} = n^2/2. \end{cases}$$

c] Line 1 : $j = 1$

Line 2 : while ($j <= n^2$) {

Line 3 : countMe

Line 4 : $j = 2 * j$; }

As $n = 2^k$, and j get multiplied by 2 time every time ($j = 2 * j$) countMe will get executed for $2k+1$ times

Example → When $n = 4$ (2^2) ⇒ $k = 2$

countMe will get executed for 3 times.

$2k + 1$ times.

b> a] $4^{2n+1} \in \Theta(16^n)$ → To prove

Let $f(n) = 4^{2n+1}$

$g(n) = 16^n$

$$\lim_{n \to 100} \frac{f(n)}{g(n)} = \lim_{n \to 100} \frac{4^{2n+1}}{16^n}$$

$$= \lim_{n \to 100} \frac{4^{2n} \times 4}{4^{2n}}$$

$$= \lim_{n \to 100} 4 \Rightarrow 4$$

= constant

Therefore

$$f(n) \in \Theta(g(n))$$

so using limits rule,

$$4^{2n+1} \in \Theta(16^n)$$

b] TO prove

$$n^5 - n^2 + 2n = \omega(n)$$

Let $f(n) = n^5 - n^2 + 2n$

$g(n) = n$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{n^5 - n^2 + 2n}{n}$$

$$= \lim_{n \to \infty} \frac{n(n^4 - n + 2)}{n}$$

$$= \lim_{n \to \infty} n^4 - n + 2$$

$$= \text{constant}$$

If $n = 0$, to $\lim_{n \to 0} \frac{f(n)}{g(n)} = \text{constant}$

Then

$$f(n) \in \omega(n)$$

Therefore,

$$n^5 - n^2 + 2n \in \omega(n)$$

7) To prove:

$10n + 8 \in O(n^2)$ but

$10n + 8 \notin \Omega(n^2)$

* As per Big-oh (O),

$f(n) = O\, g(n)$, if $\exists$ positive constants c and n_0 such that

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

let $f(n) = 10n + 8$

so $10n + 8 \leq c * g(n)$

$10n + 8 \leq 10n^2 + 8n$ // Multiply by 'n'

$f(n) = c * g(n)$ // similar

these for

$$10n + 8 \in O(n^2)$$

* As per Big - omega (Ω)

$f(n) = \Omega\, g(n)$, if $\exists$ +ve constants such that $f(n) \geq c * g(n)$ $\forall\, n \geq n_0$

Let $10n + 8 = f(n)$

$f(n) \geq c * g(n)$

$10n + 8 \geq 1 * n^2$

so $\Omega(n^2)$ //

Since Ω is lower bound expression we cannot say that

$10n + 8 \in \Omega(n^2)$

Therefore, $10n + 8 \notin \Omega(n^2)$

Here, we can infer that,

$10n + 8 \in o(n^2)$ but

$10n + 8 \notin \Omega(n^2)$

8] To prove,

$$2n^3 + 2n^2 + 10n + 1 = \Theta(n^3)$$

As per Big Θ,

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

Let

$$f(n) = 2n^3 + 2n^2 + 10n + 1$$

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

$$c_1 \cdot n^3 \leq 2n^3 + 2n^2 + 10n + 1 \leq c_2 \cdot n^3$$

Simplified

$$\rightarrow c_1 \leq 2n^3 + 2n^2 + 10n \geq 2n^3 + 2n^2 + 10n + 1 \quad \rightarrow (1)$$

for all +ve values of n, the above equation holds true and always be a constant.

$$\rightarrow c_1 \cdot 2n^2 + 10n + 1 \leq 2n^3 + 2n^2 + 10n + 1 \quad \rightarrow (2)$$

Sub $n = 1$ in eq (1) and (2)

$$c_1 = 15 \rightarrow (1)$$

$$c_1 = 13 \rightarrow (2)$$

so for all the positive values of n, the eq (2) will be true. for $\forall\ n \geq 1$.

therefore,

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \in \Theta(n^3)$$

for $\forall\ n \geq 1$ where $c_1 = 13$ and $c_2 = 15$.

9)

TO PROVE

$$f(n) + g(n) = \Theta f(n)$$

Given :

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

Assume $f(n) = 2n$, then $g(n) = n$

$$\therefore f(n) + g(n) = 2n + n = 3n$$

As per Θ notation,

$f(n) = \Theta g(n)$ if $\exists$ +ve constants c_1, c_2, n_0 such that

$$c_1 * g(n) \leq f(n) \leq c_2 * g(n)$$

then $c_1 * g(n) \leq 3n \leq 2n$

therefore, $\Theta(f) = \Theta(2n)$

$= \Theta(n)$

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

10] For i = 1, 2 ... n
    For j = i+1, i+2, ... n
     Add up entries A[i] through A[j]
     Store the result in B[i,j]
    end for
  End for

* TIME COMPLEXITY

Adding up entries will be done through a loop to n so since it has three nested loop, its time complexity will be

$$O[n^3]$$

* EFFICIENCY

```
sum_A = 0 ;    sum_B = 0 ;
for(i=0; i<n; i++) {
  for(j=1; j<n; j++) {

     B[i][j] = A[i] + A[j];

  }
}
```

* EXAMPLE

let n = 3

```
for (i .... n)
   for (j1.... n)  // [j = i+1]
```

now,

$A[1][1] = A[1] + A[2] +$
$\vdots$
$A[1] + A[2] +$
$A[2] + A[3] \cdots +$
$A[3] + A[3]$

for $A[n][n] = A[1] + A[2]$
$\vdots$
$A[n] + A[n+1]$

Therefore, it considers as one loop during computing the sum which results in increase in efficiency.