Test case:

I/P : a = 2 b = 6

Iteration 1 : s = 1 * 2 = 2

Iteration 2 : s = 2 * 2 = 4

Iteration 3 : s = 4 * 2 = 8

Iteration 4 : s = 8 * 2 = 16

Iteration 5 : s = 16 * 2 = 32

Iteration 6 : s = 32 * 2 = 64

a^b $2^6 = 64$

Test case:

I/P : a = 2 b = 6

Iteration 1 : s = 1 * 2 = 2

Iteration 2 : s = 2 * 2 = 4

Iteration 3 : s = 4 * 2 = 8

Iteration 4 : s = 8 * 2 = 16

Iteration 5 : s = 16 * 2 = 32

Iteration 6 : s = 32 * 2 = 64

a^b $2^6 = 64$

Exercise - 1

Implementation of Simple Algorithm using Brute Force Algorithm

Aim:

To implement simple algorithm such as compute a^n, selection sort and bubble sort using Brute Force Algorithm.

1. compute a^n

ALGORITM computePower(a, n)

// Input : non - negative integer n and a

// Output : non negative integer a^n

// Process

$s \leftarrow 1$

for $i \leftarrow 0$ to $n-1$ do

$s \leftarrow s * a$

return s

* Step 1: Input size

no. of bits used to represent input data 'n' in the memory

* Step 2: Basic Operation

multiplication at $s = s * a$

* Step 3: The number of times basic operations is to executed depends only on the input size.

Hence, no need to compute best, worst and average case efficiency

* Step 4: To set up sum formula

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

* Step 5: $T(n) = n - 1 + 0 + 1$

$= n$

Time Efficiency : $O(n)$

Problem code:

```
#include <stdio.h>
int power (int a, int n){
    int s=1;
    for(int i=0; i<n; i++){
        s=s*a;
    }
    return s;
}
int main (){
    int a, n;
    printf("Enter a non-negative integer 'a':");
    scanf("%d", &a);
    printf("Enter a non-negative integer 'n':");
    scanf("%d", &n);
    printf("Result '%d^%d' : %d", a, n,
                                power(a,n));
}
```

2. Selection sort

ALGORITM: Selection Sort (A[], n)

// Input: A is an array of size n, a non-negative Integer

// Output: Sorted array

```
for i ← 0 to n/2 do
    min ← i
    for j ← i+1 to n-1 do
        if (A[min] > A[j])
            min = j
    end for
    swap (A[min], A[i])
end for
```

* Step 1: Input Size

non-negative integer 'n'

Size of the array

* Step 2: Basic Operation

comparision at (A[min] > A[j])

* Step 3: The number of times basic operation is to be executed depends only on the input size. Hence, no need for best, worst and average case efficiency

* Step 4: To setup sum function

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

* Step 5: $T(n) = \sum_{i=0}^{n-2} n-1-i-1+1$

$= (n-1) + (n-2) + \cdots + (n-1-n+2)$

$= (n-1) + (n-2) + \cdots + (1)$

$= \frac{n(n-1)}{2}$

$= \frac{n^2-n}{2}$

Time efficiency: $O(n^2)$

Problem code:

```
#include <stdio.h>
int selectionSort (int A[], int n) {
    for(int i=0; i < n-1; i++) {
        int min = i;
        for(int j= i+1; j<n; j++){
            if (A[min] > A[j]) {
                min = j;
            }
        }
        int temp = A[min];
        A[min] = A[i];
        A[i] = temp;
    }
    printf("Sorted Array : ");
    for( int i=0; i<n; i++) {
        printf("%d", A[i]);
    }
}
int main() {
    int n;
    printf(" Enter n : ");
    scanf("%d", &n);
    int A[n];
    for( int i=0; i<n; i++){
        printf(" Enter element %d :", i+1);
        scanf("%d", &A[i]);
    }
    selectionSort(A, n);
}
```

Test case:

Initial IP: 70 8 6 23 15 16 2 9 45
(2 = min)

No. of Pass: 9 No. of swaps: maximum 9

Pass 1: 2 | 8 6 23 15 16 70 9 45
(2 = i, 6 = min)

Pass 2: 2 6 | 8 23 15 16 70 9 45
(6 = i, 8 = min)

Pass 3: 2 6 8 | 23 15 16 70 9 45
(no swap) (8 = i, 9 = min)

Pass 4: 2 6 8 9 | 15 16 70 23 45
(9 = i, 15 = min)

Pass 5: 2 6 8 9 15 | 16 70 23 45
(no swap) (15 = i, 16 = min)

Pass 6: 2 6 8 9 15 16 | 70 23 45
(no swap) (16 = i, 23 = min)

Pass 7: 2 6 8 9 15 16 23 | 70 45
(23 = i, 45 = min)

Pass 8: 2 6 8 9 15 16 23 45 70

Best case:

Initial I/p : 22 ↔ 75 18 30 5 15
no swap

Pass 1 : 22 75 ↔ 18 30 5 15
swap

22 18 75 ↔ 30 5 15
swap

22 18 30 75 ↔ 5 15
swap

22 18 30 5 75 ↔ 15
swap

22 ↔ 18 30 5 15 | 75
swap

Pass 2 : 18 22 ↔ 30 5 15 | 75
no swap

18 22 30 ↔ 5 15 | 75
swap

18 22 5 30 ↔ 15 | 75
swap

18 ↔ 22 5 15 | 30 75
no swap

Pass 3 : 18 22 ↔ 5 15 | 30 75
swap

18 5 22 ↔ 15 | 30 75
swap

18 ↔ 5 15 | 22 30 75
swap

Pass 4 :

5 18 ↔ 15 | 22 30 75
swap

5 ↔ 15 | 18 22 30 75
no swap

Pass 5 : 5 | 15 18 22 30 75

3. Bubble sort

```
ALGORITM Bubble sort (A,n)
//Input : Array A of size n
//output : sorted array
for i ← 0 to n-2 do
    for j ← 0 to n-1-i do
        if (A[j] > A[j+1])
            swap (A[j],A[j+1])
    end for
end for
```

* Step 1 : Input Size

'n' - Array of 'n' element

For the given Bubble sort algorithm array of size n is processed, hence input size is n

* Step 2 : To determine Basic operation

The basic operation of bubble sort algorithm is comparing adjacent element in A[j] > A[j+1]

* Step 3 : To determine the execution pattern of basic operation as, no. of times comparison took place is only depending on the input size, no need to find best, worst and average case efficiency.

* Step 4 : To set up sum formula

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

* Step 5 : To determine the algorithm efficiency class by solving the sum formula in Step 4 by using Σ standards and sum manipulation rules

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

$$= \sum_{i=0}^{n-2} n-1-i-0+1$$

$$= \sum_{i=0}^{n-2} n-i$$

$$= n + (n-1) + (n-2) + \cdots + 2$$

$$= \frac{n(n+1)}{2} - 1$$

$$= \frac{n^2+n-2}{2}$$

Time efficiency : $O(n^2)$

Problem code:

```
#include <stdio.h>
void bubbleSort (int A[], int n) {
    for(int i=0; i<n-1; i++) {
        for(int j=0; j<n-1-i; j++) {
            if (A[j] > A[j+1]) {
                int temp = A[j];
                A[j] = A[j+1];
                A[j+1] = temp;
            }
        }
    }
    printf("Sorted array : ");
    for(int i=0; i<n; i++) {
        printf("%d", A[i]);
    }
}
int main() {
    int n;
    printf("Enter n :");
    scanf("%d", &n);
    int
```

```
int i=0; i<n; i++) {
  printf("Enter element %d :", i+1);
  scanf("%d", &A[i]);

ubble Sort(A, n);
```