Lab 3 :

Question 1 :

Write a program to sort set of integers using bubble sort. Analyze its time efficiency. Obtain the experimental result of order of growth. Plot the result for the best and worst case.

```
#include<stdio.h>
#include<stdlib.h>

int bubbleSort(int arr[], int n);
int swap(int arr[], int i, int j);

int main()
{
  int n;
  printf("Enter the number of elements in the array : ");
  scanf("%d", &n);
  int arr[n];
  printf("Enter the elements : ");
  for (int i = 0; i < n; i++)
  {
    scanf("%d", &arr[i]);
  }
  int count = bubbleSort(arr, n);
  printf("The array after sorting is : ");
  for (int i = 0; i < n; i++)
  {
    printf("%d ", arr[i]);
  }
  printf("\nThe number of operations are : %d\n", count);
  return 0;
}
```

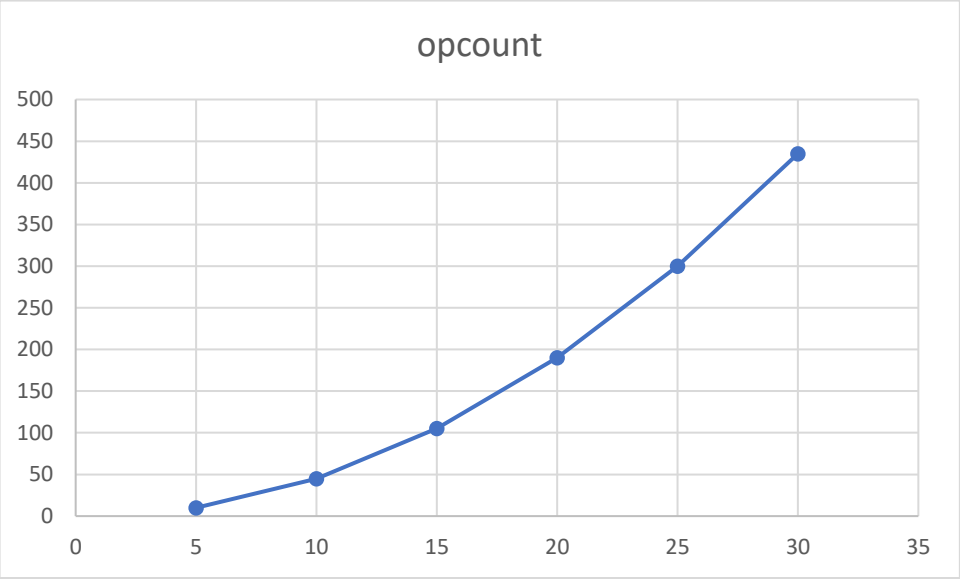
```
int bubbleSort(int arr[], int n)
{
  int cnt = 0;
  for (int i = 0; i < n - 1; i++)
  {
    for (int j = 0; j < n - i - 1; j++)
    {
      cnt++;
      if (arr[j] > arr[j + 1])
      {
        swap(arr, j, j + 1);
      }
    }
  }
  return cnt;
}

int swap(int arr[], int i, int j)
{
  int temp = arr[i];
  arr[i] = arr[j];
  arr[j] = temp;
}

```

```
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ make bubble
cc     bubble.c   -o bubble
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ ./bubble
Enter the number of elements in the array : 5
Enter the elements : 5 4 3 2 1
The array after sorting is : 1 2 3 4 5
The number of operations are : 10
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ ./bubble
Enter the number of elements in the array : 10
Enter the elements : 10 9 8 7 6 5 4 3 2 1
The array after sorting is : 1 2 3 4 5 6 7 8 9 10
The number of operations are : 45
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ ./bubble
Enter the number of elements in the array : 15
Enter the elements : 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
The array after sorting is : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
The number of operations are : 105
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ ./bubble
Enter the number of elements in the array : 20
Enter the elements : 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
The array after sorting is : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
The number of operations are : 190
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ ./bubble
Enter the number of elements in the array : 25
Enter the elements : 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
The array after sorting is : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
The number of operations are : 300
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ ./bubble
Enter the number of elements in the array : 30
Enter the elements : 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
The array after sorting is : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
The number of operations are : 435
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$
```

n	opcount
5	10
10	45
15	105
20	190
25	300
30	435



As we can observe from the plot that the time complexity for bubble sort in the **worst case** is **$O(n^2)$**;

Question 2 :

Write a program to implement brute-force string matching. Analyze its time efficiency.

```
#include<stdio.h>
#include<stdlib.h>

int BFSM(char str[], char pat[], int n, int m, int* cnt);

int main()
{
  char str[100], pat[100];
  printf("Enter the string : ");
  char c;
  int cntA = 0;
  while (1)
  {
    scanf("%c", &c);
    if (c == '\n')
    {
      str[cntA++] = '\0';
      break;
    }
    str[cntA++] = c;
  }
  int cntB = 0;
  printf("Enter the string to be searched : ");
  while (1)
  {
    scanf("%c", &c);
    if (c == '\n')
```

```
    {
      pat[cntB++] = '\0';
      break;
    }
    pat[cntB++] = c;
  }
  printf("String : ");
  for (int i = 0; i < cntA; i++)
  {
    printf("%c", str[i]);
  }
  printf("\nPattern : ");
  for (int i = 0; i < cntB; i++)
  {
    printf("%c", pat[i]);
  }
  printf("\n");
  int cnt = 0;
  int res = BFSM(str, pat, cntA - 1, cntB - 1, &cnt);
  if (res == -1)
  {
    printf("The string is not found, the opcount is : %d\n", cnt);
  }
  else
  {
    printf("The string is found at index %d and the opcount is : %d\n", res, cnt);
  }
  return 0;
}

int BFSM(char str[], char pat[], int n, int m, int* cnt)
```

```
{
  for (int i = 0; i <= n - m; i++)
  {
    int j = 0;
    while ((j < m) && (pat[j] == str[i + j]))
    {
      (*cnt)++;
      j++;
    }
    if (j == m)
      return i;
    (*cnt)++;
  }
  return -1;
}
```

```
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ make string
make: 'string' is up to date.
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ ./string
Enter the string : dipesh singh
Enter the string to be searched : singh
String : dipesh singh
Pattern : singh
The string is found at index 7 and the opcount is : 13
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ ./string
Enter the string : dipesh singh
Enter the string to be searched : chauhan
String : dipesh singh
Pattern : chauhan
The string is not found, the opcount is : 6
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ ./string
Enter the string : nononononot
Enter the string to be searched : not
String : nononononot
Pattern : not
The string is found at index 10 and the opcount is : 23
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ ./string
Enter the string : dipesh
Enter the string to be searched : s
String : dipesh
Pattern : s
The string is found at index 4 and the opcount is : 5
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ ./string
Enter the string : hello world
Enter the string to be searched : bye
String : hello world
Pattern : bye
The string is not found, the opcount is : 9
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$
```

As we can observe from the algorithm that for **best case**, when the string is found at index 0, the time complexity is **O(m)** where m is the length of the pattern to be searched. We can also observe that the **worst case** scenario, when the either the string is not found or is found at the maximum index possible, the time complexity is **O(nm)** where n is the length of the input string and m is the length of the pattern string.

Question 3 :

Write a program to implement solution to partition problem using brute-force technique and analyze its time efficiency theoretically. A partition problem takes a set of numbers and finds two disjoint sets such that the sum of the elements in the first set is equal to the second set. [Hint: You may generate power set]

```
#include<stdio.h>
#include<stdlib.h>

int solve(int arr[], int n);
int complement(int arr[], int n, int res[], int mn[], int mnn);
void print(int sub[], int ls, int mainarr[], int lm);

int main()
{
  int n;
  printf("Enter size of array to be input : ");
  scanf("%d", &n);
  int arr[n];
  printf("Enter the array : ");
  for (int i = 0; i < n; i++)
  {
    scanf("%d", &arr[i]);
  }
  int count = solve(arr, n);
```

```
  return 0;
}

int solve(int arr[], int n)
{
  int sum = 0;
  for (int i = 0; i < n; i++)
  {
    sum += arr[i];
  }
  if (sum % 2 == 1)
  {
    printf("Partition not possible\n");
    return 0;
  }
  int compare = sum / 2;
  // printf("half sum = %d\n", compare);
  int count = 0;
  int power = 1 << n;
  // printf("power = %d\n", power);
  for (int i = 1; i < power; i++)
  {
    int m[n];
    int cnt = 0;
    for (int j = 0; j < n; j++)
    {
      if (i & (1 << j))
      {
        m[cnt++] = arr[j];
      }
    }
```

```
    int arrSum = 0;
    for (int j = 0; j < cnt; j++)
    {
      arrSum += m[j];
    }
    // printf("array sum : %d\n", arrSum);
    if (arrSum == compare)
    {
      printf("Partition Possible\n{\t");
      for (int j = 0; j < cnt; j++)
      {
        printf("%d\t", m[j]);
      }
      printf("}\n{\t");
      print(m, cnt, arr, n);
      return count;
    }
  }
  printf("Partition not possible\n");
  return count;
}

void print(int sub[], int ls, int mainarr[], int lm)
{
  int l = lm - ls;
  int i, j, k = 0, flag;
  for (i = 0; i < lm; i++)
  {
    flag = 1;
    for (j = 0; j < ls; j++)
    {
```

```
      if (mainarr[i] == sub[j])
      {
        flag = 0;
        break;
      }
    }
    if (flag == 1)
    {
      printf("%d\t", mainarr[i]);
    }
  }
  printf("}\n");
}
```

```
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ make partition
make: 'partition' is up to date.
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ ./partition
Enter size of array to be input : 4
Enter the array : 1 6 6 11
Partition Possible
{       6       6       }
{       1       11      }
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ ./partition
Enter size of array to be input : 5
Enter the array : 1 2 3 5 6
Partition not possible
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ ./partition
Enter size of array to be input : 2
Enter the array : 2 3
Partition not possible
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$ ./partition
Enter size of array to be input : 3
Enter the array : 1 2 3
Partition Possible
{       1       2       }
{       3       }
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 3$
```

As we can observe from the algorithm, we create the power set of the set given. If the number of elements in the input set is n then the number of elements of power set is 2^n. Also the recurrence relation of the algorithm is given by **T(n) = 2 * T(n-1)**, which gives the time complexity as $O(2^n)$.