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;

}

```

Text

Description automatically generated

**A picture containing chart

Description automatically generated**

As we can observe from the plot that the time complexity for bubble sort in the **worst case** is **O(n2)**;

**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;

}

```

Text

Description automatically generated

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");

}

```

Text

Description automatically generated

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 **2n**. Also the recurrence relation of the algorithm is given by **T(n) = 2 \* T(n-1)**, which gives the time complexity as **O(2n)**.