**Ayush Goyal**

**190905522 CSE D 62**

**DAA Lab (Week 3) Brute Force Technique**

**Q1) Write a program to sort set of integers using bubble sort. Analyse its time**

**efficiency. Obtain the experimental result of order of growth. Plot the result for**

**the best and worst case.**

**CODE:**

#include<stdio.h>

#include<stdlib.h>

void BubbleSort(int a[],int n, int \*count){

    int i,flag,temp;

    for(;;){

        flag =0;

        for(i=0;i<(n-1);i++){

            (\*count)++;

            if(a[i]>a[i+1]){

                flag = 1;

                temp = a[i];

                a[i]=a[i+1];

                a[i+1]=temp;

            }

        }

        n--;

        if(flag == 0)

            break;

    }

    return;

}

int main(){

    int count,c,n,i,j;

    printf("\nEnter the number of test cases : ");

    scanf("%d",&c);

    for(i=0;i<c;i++){

        count=0;

        printf("Enter size of array : ");

        scanf("%d",&n);

        int a[n];

        printf("\nEnter the array elements : ");

        for(j=0;j<n;j++){

            scanf("%d",&a[j]);

        }

        BubbleSort(a,n,&count);

        printf("Count = %d\tn = %d\t",count,n);

        printf("\nSorted Array : \n");

        for(j=0;j<n;j++){

            printf("%d ",a[j]);

        }

        printf("\n");

    }

}

**INPUT/OUTPUT:**

**Text

Description automatically generated**

**Graph Plot for Time Efficiency for best and worst cases:**

**Chart, line chart

Description automatically generated**

**Time Efficiency Analysis:**

The best case for this bubble sort algorithm is when the input array is already sorted and then the time complexity is **O(n)**. The worst case for this algorithm is when the input array is in decreasing order and then the time complexity is **O(n2)**.

**Q2) Write a program to implement brute-force string matching. Analyse its time**

**efficiency.**

**CODE :**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int checksubstring(char str[], char sub[]){

    int c=0,c1,flag=1;

    int i,j,n,m;

    n = strlen(str);

    m = strlen(sub);

    for(i=0;i<=(n-m);i++){

        c1=0;

        flag = 1;

        for(j=i; c1<m; j++,c1++){

            c++;

            if(str[j] != sub[c1]){

                flag = 0;

                break;

            }

        }

        if(flag == 1)

            break;

    }

    if(flag == 1)

        printf("\nSubstring Found!");

    else

        printf("\nSubstring not found !");

    return c;

}

int main(){

    printf("\nEnter a string : ");

    char str[20],sub[20];

    scanf("%[^\n]%\*c",str);

    printf("\nEnter a substring : ");

    scanf("%[^\n]%\*c",sub);

    int count = checksubstring(str,sub);

    printf("\nCount = %d",count);

}

**INPUT/OUTPUT:**

**Text

Description automatically generated**

**Time Efficiency Analysis:**

As observed from the algorithm, the best case is when the string is found at index 0 and its time complexity is **O(m)** where m is the length of the substring to be found/searched.

We can see that the worst case is when either the string is found at the maximum index possible or not found at all. Then its time complexity is **O(nm)** where m is the length of the substring to be searched and n is the length of the original input string to be searched from.

**Q3) Write a program to implement solution to partition problem using brute-force**

**technique and analyse 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]**

**CODE:**

#include<stdio.h>

#include<stdlib.h>

#include<stdio.h>

#include<math.h>

void printsubarr(int sub[],int ls,int mainarr[],int lm){

    int l=lm-ls;

    int i,j,k=0,flag;

*//int \*comparr = (int\*)calloc(l,sizeof(int));*

    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){

*//comparr[k++] = mainarr[i];*

            printf("%d ",mainarr[i]);

        }

    }

    printf("}\n");

}

void solve(int arr[], int n){

    int totsum =0,i;

    for(i=0;i<n;i++){

        totsum+=arr[i];

    }

    if(totsum % 2 != 0){

        printf("\nNot Possible.");

        return;

    }

    totsum/=2;

    unsigned int pow\_set\_size = pow(2, n);

    int counter, j,k;

    for(counter = 0; counter < pow\_set\_size; counter++)

    {

        int subarr[n];

        int c=0;

        for(j = 0; j < n; j++){

            if(counter & (1<<j))

                subarr[c++] = arr[j];

        }

        int sum =0;

        for(k=0;k<c;k++){

            sum+=subarr[k];

        }

        if(sum == totsum){

            printf("\nPossible.\n{ ");

            for(k=0;k<c;k++){

                printf("%d ",subarr[k]);

            }

            printf("} , { ");

            printsubarr(subarr,c,arr,n);

            return;

        }

    }

    printf("\nNot Possible");

    return;

}

int main(){

    int n,i;

    printf("\nEnter number of elements : ");

    scanf("%d",&n);

    int arr[n];

    printf("\nEnter elements : ");

    for(i=0;i<n;i++){

        scanf("%d",&arr[i]);

    }

    solve(arr,n);

    return 0;

}

**INPUT/OUTPUT :**

**Text

Description automatically generated**

**Time Efficiency Analysis:**

We can see from the algorithm, that we create the power set of the given input set. If the number of elements in the input is n then the number of elements in the power set is 2n .

The recurrence relation of the algorithm is given by:

**T(n) = 2 \* T(n-1)**  This, on solving gives us the time complexity of this algorithm as **O(2n)**

**THE END**