LAB3

Aniket Sambher

Reg no-190905466

Roll no-58

1.

ALGORITHM BubbleSort(A[0..n − 1])

//Sorts a given array by bubble sort

//Input: An array A[0..n − 1] of orderable elements

//Output: Array A[0..n − 1] sorted in nondecreasing order

for i ←0 to n − 2 do

    for j ←0 to n − 2 − i do

        if A[j + 1]<A[j] swap A[j] and A[j + 1]

note-we make a slight modification for making the best case input to run in O(n) time by adding a flag which states whether elements have been swapped in a particular iteration or not

CODE

```
#include <stdio.h>

#include<stdlib.h>

void swap(int *xp, int *yp)

{

   int temp = *xp;

   *xp = *yp;

   *yp = temp;

}


// A function to implement bubble sort

void bubbleSort(int arr[], int n)

{

  int i, j,opcount=0;

  int swapped;

  for (i = 0; i < n-1; i++) {
```

```
      // Last i elements are already in place
      for (j = 0; j < n-i-1; j++) {
         swapped=0;
         opcount++;
         if (arr[j] > arr[j+1]){
           swap(&arr[j], &arr[j+1]);
             swapped=1;
         }
      }
         if(swapped ==0)
          break;
  }
      printf("\nnumber of operations=%d \n",opcount);
}
void printArray(int arr[], int size)
{
   int i;
   for (i=0; i < size; i++)
       printf("%d ", arr[i]);
   printf("\n");
}

// Driver program to test above functions

int main() {
 int *a;
 int i,n = 5;
 // generate worst case input of different input size
 for (int j=0; j < 4; j++) // repeat experiment for 4 trials
 {
```

```
    a = (int *)malloc(sizeof(int)*n);
    for (int k=0; k< n; k++)
        a[k] = n-k; // descending order list
    printf("Elements are ");
    for(i=0;i<n;i++)
        printf("%d ",a[i]);
    bubbleSort(a,n);
    printf("Elements after bubble sort ");
    for(i=0;i<n;i++)
        printf("%d ",a[i]);
    printf("\n");
    free(a);
    n = n + 5; // try with a new input size
  }

}
```

TIME COMPLEXITY ANALYSIS

We know that for worst case the loop will run for (n-1)*n/2 iterations in bubble sort

Thereby for worst case O(n^2)

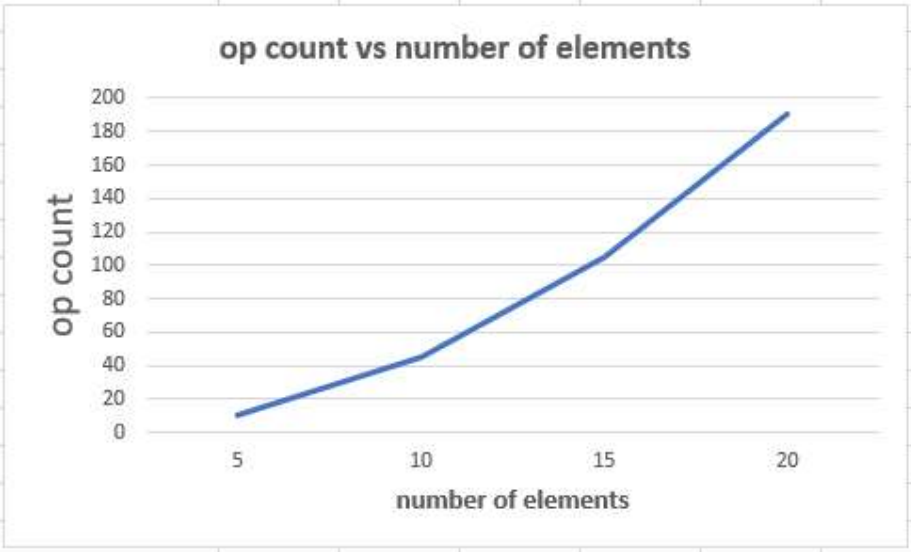
Now we make a slight modification to our bubble sort algorithm to make it run for n iterations if no swaps occur by using a flag “swapped” in the inner loop

Thereby for best case O(n)

For worst case O(n^2)

```
Elements are 5 4 3 2 1
number of operations=10
Elements after bubble sort 1 2 3 4 5
Elements are 10 9 8 7 6 5 4 3 2 1
number of operations=45
Elements after bubble sort 1 2 3 4 5 6 7 8 9 10
Elements are 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
number of operations=105
Elements after bubble sort 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Elements are 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
number of operations=190
Elements after bubble sort 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
PS C:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week3>
```

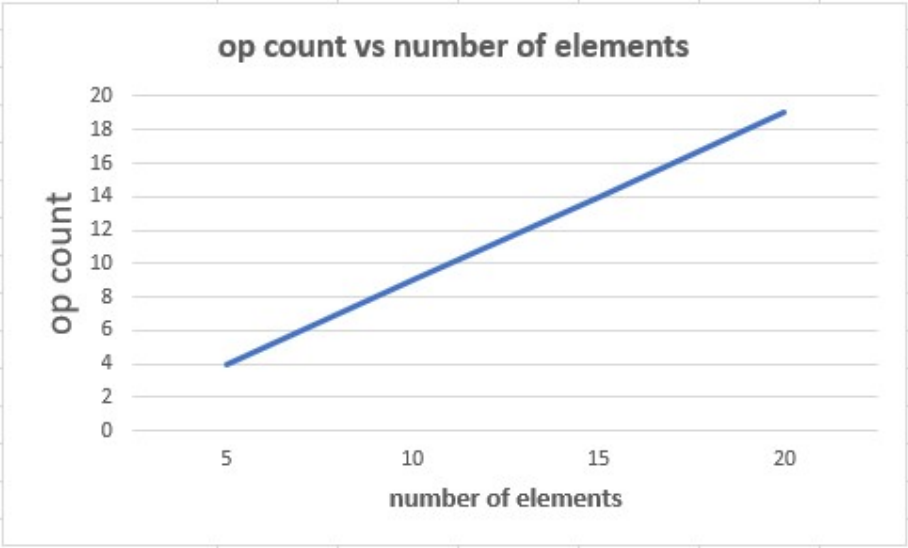
number of elements	opcount
5	10
10	45
15	105
20	190



For best case O(n)

```
Elements are 0 1 2 3 4
number of operations=4
Elements after bubble sort 0 1 2 3 4
Elements are 0 1 2 3 4 5 6 7 8 9
number of operations=9
Elements after bubble sort 0 1 2 3 4 5 6 7 8 9
Elements are 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
number of operations=14
Elements after bubble sort 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Elements are 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
number of operations=19
Elements after bubble sort 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

number of elements	opcount
5	4
10	9
15	14
20	19



2. ALGORITHM BruteForceStringMatch(T [0..n − 1], P[0..m − 1])

//Implements brute-force string matching

//Input: An array T [0..n − 1] of n characters representing a text and

// an array P[0..m − 1] of m characters representing a pattern

//Output: The index of the first character in the text that starts a

// matching substring or −1 if the search is unsuccessful

for i ←0 to n − m do

    j ←0

    while j <m and P[j]= T [i + j] do

            j ←j + 1

    if j = m return i

return −1

code

```
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
int stringmatch(char str[],char subs[]){
   int n=strlen(str);
   int m=strlen(subs);
   int flag=0,j,i,opcount=0;

   for(int i=0;i<=n-m;++i){
      for(j=0;j<m;++j){
         opcount++;
         if(str[i+j]!=subs[j])
            break;
      }
      if(j==n){
         flag=1;
         break;
```

```
        }
    }
    printf("operand count=%d",opcount);
    return flag;
}
int main(){
    char str[100] ,subs[100] ;
    printf("enter the string ");
    scanf("%s",str);
    printf("enter the substring ");
    scanf("%s",subs);
    if(stringmatch(str,subs))
        printf("\nmatch");
    else
        printf("\nnot match");

}
```

Outputs of worst cases

```
enter the string aaaaaaaaaa
enter the substring i
operand count=10
not match
hing.c -o stringmatching } ; if ($?) { .\stringmatching }
enter the string aaaaaaaaaa
enter the substring ai
operand count=18
not match
```

```
PS C:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week3> cd "c:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week3\" ; if ($?) { gcc stringmatc
hing.c -o stringmatching } ; if ($?) { .\stringmatching }
enter the string aaaaaaaaaa
enter the substring aai
operand count=24
not match
PS C:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week3> cd "c:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week3\" ; if ($?) { gcc stringmatc
hing.c -o stringmatching } ; if ($?) { .\stringmatching }
enter the string aaaaaaaaaa
enter the substring aaai
operand count=28
not match
```

TIME COMPLEXITY ANALYSIS

We know that outer loop runs for n-m+1 iterations and inner loop can run for at most n iterations

Thereby time complexity =O((n-m+1)*m)=O((n-m)*m)(approx)

This is when initial characters match but the last character doesn't match

Now for best case we know that if substring matches the string exactly the string we will take maximum m iterations

Thereby for best case O(m)

3.partition problem

CODE

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

int subsetpartitions(int arr[],int n)
{

	int sum = 0;
	int opcount=0;

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

		sum += arr[i];
	}
	if(sum%2!=0)
		return 0;


	sum = sum / 2;
	int flag = 0;

	for (int i = 0; i < pow(2, n); i++)
	{
		int j = i;
		int ind = 0;
```

```
int csum = 0;

for (int k = 0; k < n; k++)
{       opcount++;
        if (j & 1)
                csum += arr[ind];

        ind++;
        j = j >> 1;  //else shift left
}

if (csum == sum)
{
        printf("Subset found\n");
        j = i;
        ind = 0;
        printf("The set is: ");

        for (int k = 0; k < n; k++)
        {
                if (j & 1)
                        printf("%d ", arr[ind]);

                ind++;
                j = j >> 1;  //else shift left
        }

        printf("\n");
        j = i;
        ind = 0;
```

```
                        printf("The other set: ");

                        for (int k = 0; k < n; k++)
                        {
                                if (!(j & 1))
                                        printf("%d ", arr[ind]);

                                ind++;
                                j = j >> 1;  //else shift left
                        }

                        printf("\n");
                        flag = 1;

                        break;
                }
        }

printf("opcount=%d",opcount);
if(!flag)
        return 0;
else
        return 1;




}

int main(){
        int n;
```

```
        scanf("%d", &n);
        int arr[n];
        for (int i = 0; i < n; i++)
        {
                scanf("%d", &arr[i]);
        }
        if (!subsetpartitions(arr,n))
                printf("\nNo such subset possible\n");
}
```

OUTPUT

```
PS C:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week3> cd "c:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week3\" ; if ($?) { gcc subsetpart
ition.c -o subsetpartition } ; if ($?) { .\subsetpartition }
4
5 1 3 3
Subset found
The set is: 5 1
The other set: 3 3
opcount=16
PS C:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week3>
```

WORST CASE OUTPUT

```
PS C:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week3> cd "c:\Users\aniket\Desktop\desktop\sem4\daa\lab\daa\week3\" ; if ($?) { gcc subsetpart
ition.c -o subsetpartition } ; if ($?) { .\subsetpartition }
6
2 4 6 8 10 12
opcount=384
No such subset possible
```

TIME COMPLEXITY ANALYSIS

We can see that the outer loop runs for 2^n iterations (power set)and the inner loop can run for n interations

The worst case occurs when there is no partition possible and the partition sum is even (if not the loop will return from the function before entering the loop)

Thereby time complexity in worst case O((2^n)*n)