**190905513 MOHAMMAD DANISH EQBAL**

---

# LAB 3

## Lab Exercises:

**1). 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.**

**Algorithm:**

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]

**Program:**

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

void bubble_sort(int a[], int n){
          int temp,i,ii,j;
          int opcount=0;
          for(ii=0; ii<n-1; ii++){
          for (j=0; j<n-ii-1; j++){
            opcount++;
      if (a[j]>a[j+1]){
                              temp=a[j];
                              a[j]=a[j+1];
                              a[j+1]=temp;

          }
      else break;
          }}
   printf("\nOperation Count= %d\nSorted Array:",opcount);
   for(i=0;i<n;i++)
   {printf("%d ",a[i]);}

}

int main(){
  int n, i;
  printf("Enter the array size: ");
```

```
  scanf("%d",&n);
  printf("Enter the array: ");
  int a[1000];
  for(i=0;i<n;i++)
    { scanf("%d",&a[i]);}

  bubble_sort(a,n);
  return 0;
}
```

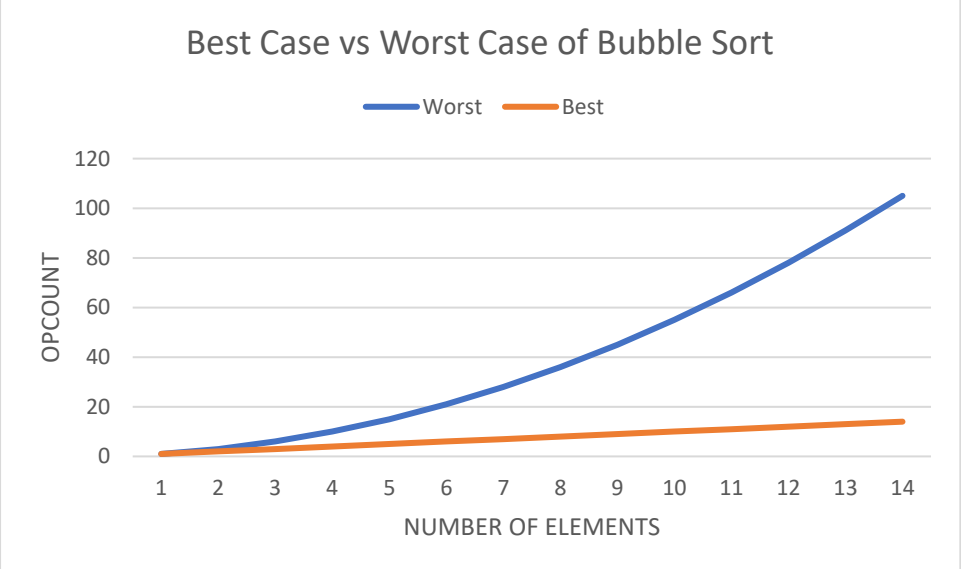
**Output:**

```
Enter the array size: 5
Enter the array: 1 2 3 4 5

Operation Count= 4
Sorted Array:1 2 3 4 5
Enter the array size: 5
Enter the array: 5 4 3 2 1

Operation Count= 10
Sorted Array:1 2 3 4 5
Process returned 0 (0x0)   execution time : 14.014 s
Press any key to continue.
```

**Order of Growth:**




**Time Complexity Analysis:**

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

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

$T(n) = \sum_{i=0}^{n-2}[(n-i]$

$T(n) = n(n-1)/2 \in O(n^2)$

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

**Algorithm:**

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

**Program:**

```
int stringMatch(char arr1[],char arr2[])
{
   int i,j,opcount=0;
   int n = strlen(arr1);
   int m = strlen(arr2);
   for(i=0;i<=n-m;i++){
      opcount++;
      j=0;
      while((j<m) & (arr2[j]==arr1[i+j])){
         opcount++;
         j++;
      }
      if(j==m){
         printf("Opcount = %d \n",opcount);
         return i;
      }
   }
   printf("Opcount = %d \n",opcount);
   return -1;
}
int main()
{
   char arr1[]={"aaaaaaaaaaaaaaaaaaaaaaaaaab"};
   char arr2[]={"aaac"};
   int index = stringMatch(arr1,arr2);
   if(index==-1)
      printf("Not Found");
   else
      printf("Found at index %d",index);
   return 0;
}
```

**Time Complexity Analysis:**

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

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

$T(n) = m(m-n+1) \in O(mn)$

**Output:**

```
Opcount = 100
Not Found
Process returned 0 (0x0)   execution time : 4.059 s
Press any key to continue.
```

**3). 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]**

**Program:**

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

bool isSubset(int arr[],int n,int sum)
{
   if(sum==0)
      return true;
   else if(n==0 && sum!=0)
      return false;
   else if(arr[n-1]>sum){
      return isSubset(arr,n-1,sum);
   }
   else{
      return ((isSubset(arr,n-1,sum)) || (isSubset(arr,n-1,sum-arr[n-1])));
   }
}

bool ifPartition(int arr[],int n)
{
   int sum =0;
   for(int i=0;i<n;i++){
      sum+= arr[i];
   }
   if(sum%2==0){
      return isSubset(arr,n,sum/2);
   }
   else{
      return false;
   }
```

```
}
int main()
{
   int arr[100];
   int n,i;
   printf("Enter Size of Array: ");
   scanf("%d",&n);
   printf("Enter the Array: ");
   for(i=0;i<n;i++)
   {
      scanf("%d",&arr[i]);
   }
   if(ifPartition(arr,n)==true){
      printf("TRUE");
   }
   else{
      printf("FALSE");
   }
   return 0;
}
```

**Output:**

```
Enter Size of Array: 5
Enter the Array: 6 11 2 3 0
TRUE
Process returned 0 (0x0)   execution time : 19.327 s
Press any key to continue.
```

**Time Complexity Analysis:**

The running time T(n) for partition algorithm is given by recurrence relation:
$T(n) = 2T(n-1)$
By applying the method of backward substitution, we get:
$T(n) \in O(2^n)$