Experiment #4		Student ID	
Date		Student Name	

Experiment Title: Implementation of programs on Sorting and Searching problems.

Aim/Objective: To understand the concept and implementation of programs on Sorting and Searching problems.

Description: The students will understand and able to implement programs on Sorting and Searching problems.

Pre-Requisites:

Knowledge: Sorting, Searching, Arrays in C/C++/Python

Pre-Lab:

1. Sort the following elements using Quick sort technique:

   54, 26, 93, 17, 77, 31, 44, 55, 20

- **Procedure/Program:**

```
#include <studio.h>
void swap(int*a,int*b){
    int f= *a;
      *a= *b;
      *b =f;{
int partition (int arr [],int low,int high)
  {
    int pivot = arr [high];
    int i= (low-1);
For(int j= low; j<=high-1;j++){
    if(arr[i] < Pivot){
        i++;
    }
swap( arr[i], &arr[j]);
  }}
swap( arr[i+1], &arr[high]);
    return(i+1);
  }
void quicksort(int arr[],int lo,
                int high)
  {
    if(low < high){
  int Pi= Partition (arr,low,high)
```

```
quicksort(arr, low,Pi-1);
quicksort(arr,Pi+1, high);
  }
void Printarray(int arr[],int size)
  {
For(int i=0; i< size ; i++)
  Printf("%d", arr [i]);
  printf("\n");
  }
```

Experiment #4		Student ID	
Date		Student Name	

- **Data and Results:** [54, 26, 93, 17, 77, 31, 44, 55, 20]

P = 20, L = 17, R = [54, 26, 93, 77, 31, 44, 55]

P = 55, L = 26, 31, 44, P = 54, R = 93, 77

L = 17 P = 20 R = 26, 31, 44, 55, 77, 93]

final sort as = [17, 20, 26, 31, 44, 54, 55, 77, 93]

- **Analysis and Inferences:**

Quick sort is an efficient comparison-based sorting algorithm with an average of O(n log n). it performs well on large data sets

2. Stefan is a guy who is suffering with OCD. He always like to align things in an order. He got a l of strings for his birthday party as gifts. He like to sort the strings in a unique way. He wants h strings to be sorted based on the count of characters that are present in the string.

Input

aaabbc

aabbcc

Output

cbbaaa

aabbcc

If in case when there are two characters is same, then the lexicographically smaller one will be printe first

Input:

aabbccdd

aabcc

Output:

aabbccdd

baacc

- **Procedure/Program:**

Experiment #4		Student ID	
Date		Student Name	

```
int compare (const void *a,const void *b)
{
  char freq *(F1=(charfreq*)a
  char freq *(F2=(charfreq*)b
:f(F1->frequency != F2->frequency)
{
return (F2->frequency-(F1->frequency
return(F1->character-(F2->character
}
void sortstring (char *str)
{
  int len=strlen(str);
  int freq[200]={0};
charfreq charfreq[200];
for(int i=0;i<len;i++)
freq[(int) str[i]]++;
```

```
  int index=0;
for (int i=0; i<200; i++)
{
if (freq[i]>0) {
charfreq[index].character
                (char)i;
charfreq[index].frequency=
                freq[i];
  index ++;
}
qsort( charfreq ,index,sizeof(charfreq)
                compare);
int str index=0;
for(int i=0; i<index ; i++){
for(int j=0;j<charfreq[i].freq ;j++)
```

- **Data and Results:**

  cbbaaa
  aabbcc
  aabbcc dd, The space complexity o(n) for string
  bacc

- **Analysis and Inferences:** Time complexity The time time complexi is by the sorting step o(n) and sorting take o(n log n)

In-Lab:

1. Given an array arr[] of N strings, the task is to sort these strings according to the number of upper-case letters in them try to use zip function to get the format.

Input arr[] = {poiNtEr, aRRAy, cOde, foR}

Output: [('cOde', 1), ('foR', 1), ('poiNtEr', 2), ('aRRAy', 3)]

The input array describes the following:

"aRRAy" R, R, A->3 Upper Case Letters

"poiNtEr" N, E->2 Upper Case Letters

"cOde" O->2 Upper Case Letters

"foR" R->3 Upper Case Letters

- **Procedure/Program:**

```
int countuppercase(const char* str){
    int count=0;
    while(*str){
    if(is upper(*str)){
     count++;
     }
    str++;
    }
return count;
int compare(const void*a, const void*b)
 const char*strA = * (const char **)a;
 const char*strB = * (const char *)b;
 int countA= countuppercase(strA);
```

- **Data and Results:**

(('code',1),('For',1), ('Points',1),('arrA',3))

* Time complexity the upper case letter O(n*m)

time complexity where of stringer

- **Analysis and Inferences:** space complexity is O(1)

2. Andrea is working in a photo studio where his boss has given him a task to arrange the photos of family members. He is French and he do not know English somehow, he managed to send the list of names to you (his friend). Help Andrea to sort the photos.

(Note: implement the odd even merge algorithm)

Input

5

Neil Katherine Harry Stefan Dennis

Experiment #4		Student ID	
Date		Student Name	

Output

Dennis Harry Katherine Neil Stefan

- **Procedure/Program:**

```
void oddevenmerge(char*arr[], int lo, int hi, int s)
  int step = s*2;
  if(step<(hi-lo))
  oddevenmerge(arr,lo,hi,step);
  oddevenmerge(arr,lo+s,hi,step);
  for(int i=lo+s; i<hi-s; i+=step)
  {
    if(strcmp(arr[i],arr[i+s])>0)
    char *temp = arr[i];
      arr[i]=arr[i+s];
      arr[i+s]=temp;
  }
```

```
else
if(strcmp(arr[lo],arr[lo+s])>0){
char *temp = arr[lo];
arr[lo] = arr[lo+s];
arr[lo+s] = temp;
}
}
```

```
void oddevenmerged(char*arr[]
if(high){ int lo, int hi);
int mid=(hi-lo)/2;
oddevenmerge(arr,lo,mid);
oddevenmerge(arr,mid,hi);
```

- **Data and Results:**

Dennis Mary Katherine Neil Stefan

- **Analysis and Inferences:** The time complexity of odd-even merge sort $O(n \log^2 n)$

Space complexity is $O(1)$ if sorting is done in-place

Post-Lab:

1. James is sharing his information with his friend secretly in a chat. But he thinks that message should not understandable to anyone only for him and his friend. So he sent the message in the following format.

Experiment #4		Student ID	
Date		Student Name	

Input

a1b2c3d4e

Output

abbdcfdhe

Explanation:

The digits are replaced as follows:

- s[1] -> shift('a',1) = 'b'
- s[3] -> shift('b',2) = 'd'
- s[5] -> shift('c',3) = 'f'
- s[7] -> shift('d',4) = 'h'

- **Procedure/Program:**

```
charshift (char1, int num){
    setw (num);
void Process message (char* input, char* output)
    {int len = strlen (input);
     intout Index=0;
for (int i=0; i<len; i++){
    if (is digit (input[i])){
int shiftvalue = input[i] -'0';
output [out Index -1] = shift (output Index-1), shiftvalue
}else{
output[outIndex++] = input[i];
    }}
    output [out index] = '\0';
```

Course Title	Design and Analysis of Algorithms	ACADEMIC YEAR: 2024-25
Course Code(s)	23CS2205R	Page 19 of 93

Experiment #4		Student ID	
Date		Student Name	

```
int main(){
 char input[] = "abcdeg";
 char output[100];
 Processmessage(input, output);
 printf("output: %s\n", output);
      return
```

- **Data and Results:**

a1b2c3de
O/P:- abbdcfdke

- **Analysis and Inferences:** *The complexity is O(n) and also space complexity
*The approach checks that the string main tags The secret message format as acquired by Jane

- **Sample VIVA-VOCE Questions (In-Lab):**

1. Can linear search be used on unsorted arrays? Why or why not?
2. How would you modify a linear search to return the index of all occurrences of a target value?
3. What are the advantages and disadvantages of linear search compared to other search algorithms?
4. What are the time complexities of binary search in the best, worst, and average cases?
5. Compare binary search and linear search in terms of efficiency and use cases.

Evaluator Remark (if Any):	Marks Secured: 40 out of 50
	Signature of the Evaluator with Date R. Vijaya

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Student ID
Student Name

puti chast output)

put-Index-1) shiftvalue

);

W-3

① Linear search can be used to search for a small to large values of an unsort list the list is searching to match

② The linear search to return the index not break who matrix is found.

③ • Does not recquire the data to be sort
 • each item may need not be checked

④ Worst case o(n logn)

Best case o(1)

Average case o(n logn)

⑤ • Binary search we used for an sorted Data
 • The linear search we used an un sorted data.