Experiment - 1

Aim: Write a code to perform Merge Sort.

Apparatus Used: Windows 11, Dev C++.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m-l+1;
    int n2 = r-m;
    int L[n1], R[n2];
    for(i=0; i<n1; i++){
        L[i] = arr[l+i];
    }
    for(j=0; j<n2; j++){
        R[j] = arr[m+1+j];
    }
    i=0;
    j=0;
    k=l;
    while(i<n1 && j<n2){
        if(L[i] <= R[j]){
            arr[k] = L[i];
            i++;
        }
        else{
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while(i<n1){
        arr[k] = L[i];
        i++;
        k++;
    }
```

```
    while (j < n2){
        arr[k] = R[j];
        j++;
        k++;
    }
}
void mergeSort(int arr[], int l, int r){
    if(l < r){
        int m = l + (r-l)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}
void printArray(int A[], int size){
    int i;
    for(i=0; i<size; i++){
        printf("%d", A[i]);
    }
    printf("\n");
}
int main(){
    int arr[] = {20, 18, 15, 5, 9, 7};
    int arr_size = sizeof(arr)/sizeof(arr[0]);
    printf("Given array is: \n");
    printArray(arr, arr_size);
    mergeSort(arr, 0, arr_size-1);
    printf("\n Sorted array is: \n");
    printArray(arr, arr_size);
    return 0;
}
```

Output:

Given array is:
20 18 15 5 9 7
Sorted array is:
5 7 9 15 18 20

Time complexity: O(nlogn)

Experiment - 2

Aim: Write a code to perform Quick Sort.

Apparatus Used: Windows 11, Dev C++

Source code:

```
# include <stdio.h>
void swap(int *a, int *b){
    int t = *a;
    *a = *b;
    *b = t;
}
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[j] <= pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i+1], &arr[high]);
    return (i+1);
}
void quicksort(int a[], int p, int r){
    if (p < r){
        int q;
        q = partition(a, p, r);
        quicksort(a, p, q-1);
        quicksort(a, q+1, r);
    }
}
void printArray(int a[], int size){
    int i;
    for(i=0; i<size; i++){
        printf("%d", a[i]);
    }
    printf("\n");
}
```

```
int main()
{
    int arr[50];
    int n;
    printf("Enter the number of elements you want to add in the array: ")
    scanf("%d", &n);
    printf("Enter the elements: \n");
    for(int i=0; i<n; i++){
        scanf("%d", &arr[i]);
    }
    quicksort(arr, 0, n-1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Output:

```
Enter the number of elements you want to add in the array: 10
Enter the elements:
12
53
25
60
39
87
9
78
95
36
Sorted array:
9 12 25 36 39 53 60 78 87 95
```

Time complexity: O(nlogn)

Experiment - 3

Aim: Given a sorted array and a number x, search two elements of the array such their sum is x. Expected time complexity is $O(n^2)$ and $O(n)$.

Apparatus Used: Windows 11, Dev c++

Source Code:

```
#include<stdio.h>
void BindPair(int arr[], int n, int target){
    int left=0, right = n-1;
    while(left < right){
        int sum = arr[left] + arr[right];
        if(sum == target){
            printf("Pair Formed: %d, %d\n", arr[left], arr[right]);
            return;
        }
        else if(sum < target){
            left++;
        }
        else{
            right--;
        }
    }
    printf("Pair not Formed\n");
}
int main(){
    int arr[] = {1, 3, 5, 7, 9, 11, 13};
    int n = sizeof(arr)/sizeof(arr[0]);
    int target = 16;
    BindPair(arr, n, target);
    return 0;
}
```

Output:

Pair Formed: 3, 13

Time complexity: O(N)

Experiment - 4

Aim: Given a sorted array and a number x, write a function that counts the occurrences of x in the array. Expected time complexity is O(n) and O(logn).

Apparatus Used: Windows 11, Dev c++.

Source code:

```
#include<stdio.h>
int binarySearch(int arr[], int l, int r, int x){
    if(r<l){
        return -1;
    }
    int mid = l+(r-l)/2;
    if(arr[mid]==x){
        return mid;
    }
    if(arr[mid]>x){
        return binarySearch(arr,l,mid-1,x);
    }
    return binarySearch(arr,mid+1,r,x);
}
int countOccurrences(int arr[], int n, int x){
    int ind = binarySearch(arr,0,n-1,x);
    if(ind==-1){
        return 0;
    }
    int count = 1;
    int left = ind-1;
    while(left>=0 && arr[left]==x){
        count++;
        left--;
    }
    int right = ind+1;
    while(right<n && arr[right]==x){
        count++;
        right++;
    }
    printf("Occurance of %d in %d\n", x, count);
}
```

```
int main(){
    int arr[] = {1,2,2,2,2,3,1,7,8,8};
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 2;
    countOccurrences(arr, n, x);
    return 0;
}
```

Output :

Occurrence of 2 in 1

Time complexity : O(logn)

Experiment - 5

Aim: Implement Binary Search using Divide and Conquer.

Apparatus Used: Windows 11, Dev C++

Source code:

```
#include<stdio.h>
int binary_search(int A[], int key, int len){
    int low = 0;
    int high = len - 1;
    while(low <= high){
        int mid = low + ((high - low)/2);
        if(A[mid] == key){
            return mid;
        }
        if(key < A[mid]){
            high = mid - 1;
        }
        else{
            low = mid + 1;
        }
    }
    return -1;
}
int main(){
    int a[10] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 22};
    int key = 3;
    int position = binary_search(a, key, 10);
    if(position == -1){
        printf("Not found");
        return 0;
    }
    printf("Found %d at position %d", key, position);
    return 0;
}
```

Output:

Found 3 at position 1.

Time complexity: O(log n).

Experiment - 6

Aim: Implement a greedy algorithm to solve the fractional knapsack problem.

Apparatus Used: Windows 11, Dev C++

Source code:

```
#include <stdio.h>
int main() {
  float weight[50], profit[50], ratio[50], totalvalue, temp, capacity, amount;
  int n, i, j;
  printf("Enter the number of items: ");
  scanf("%d", &n);
  for(i=0; i<n; i++){
      printf("Enter weight and profit for item[%d]: \n", i);
      scanf("%f %f", &weight[i], &profit[i]);
  }
  printf("Enter the capacity of knapsack: \n");
  scanf("%f", &capacity);
  for(i=0; i<n; i++){
     ratio[i] = profit[i]/weight[i];
  }
  for(i=0; i<n; i++){
     for(j=i+1; j<n; j++){
        if(ratio[i] < ratio[j]){
           temp = ratio[j];
           ratio[j] = ratio[i];
           ratio[i] = temp;
           temp = weight[j];
           weight[j] = weight[i];
           weight[i] = temp;
           temp = profit[j];
           profit[j] = profit[i];
           profit[i] = temp;
        }
        printf("Knapsack Problem using Greedy Algorithm: \n");
        for(i=0; i<n; i++){
           if(weight[i] > capacity){
              break;
           }
```

```
                else {
                    totalvalue = totalvalue + profit[i];
                    capacity = capacity - weight[i];
                }
            }
            if (i<n){
                totalvalue = totalvalue + (ratio[i] * capacity);
            }
        }
    }
    printf("\n The maximum value is : %f \n", totalvalue);
    return 0;
}
```

Output :

```
Enter the number of items : 2
Enter weight and Profit for item[0]:
5
5
Enter weight and Profit for item[1]:
2
10
Enter the capacity of knapsack:
4
Knapsack Problem using Greedy Algorithm:
The maximum value is: 12.000000
```

Time complexity: $O(N^2)$.

Experiment - 3

Aim: Find the largest and smallest number simultaneously in an array using Divide & Conquer Principle.

Apparatus Used: Windows 11, Dev c++.

Source code:

```
#include<stdio.h>
int max, min;
int a[100];
void maxmin(int i, int j){
    int max1, min1, mid;
    if(i== j){
        max = min = a[i];
    }
    else{
        if(i== j-1){
            if(a[i] < a[j]){
                max = a[j];
                min = a[i];
            }
            else{
                max= a[i];
                min = a[j];
            }
        }
        else{
            mid = (i+ j)/2;
            maxmin(i, mid);
            max1 = max;
            min1 = min;
            maxmin(mid+1, j);
            if(max < max1){
                max = max1;
            }
            if(min > min1){
                min = min1;
            }
        }
    }
}
```

```
int main() {
    int i, num;
    printf("\nEnter the total number of elements : ");
    scanf("%d", &num);
    printf("Enter the numbers : \n");
    for( i=1 ; i<=num ; i++) {
        scanf("%d", &a[i]);
    }
    max = a[0];
    min = a[0];
    maxmin(1, num);
    printf("Minimum element in array : %d \n", min);
    printf("Maximum element in array : %d \n", max);
    return 0;
}
```

Output:

```
Enter the total number of elements : 4
Enter the numbers:
33
52
89
15
Minimum element in array : 15
Maximum element in array : 89
```

Time complexity : O(logn)

Experiment - 8

Aim: Implement the greedy algorithm to solve the problem of the Job sequencing with deadlines.

Apparatus Used: Windows 11, Dev c++.

Source code:

```
#include<stdio.h>
#define MAX 100
typedef struct Job{
    char id[5];
    int deadline;
    int profit;
} Job;
void JobSequencingWithDeadline(Job jobs[], int n);
int minvalue(int x, int y){
    if(x<y){
        return x;
    }
    return y;
}
int main(void){
    int i, j;
    Job jobs[5] = {
        {"j1", 2, 60},
        {"j2", 1, 100},
        {"j3", 3, 20},
        {"j4", 2, 40},
        {"j5", 1, 20},
    };
    Job temp;
    int n = 5;
    for(i=1; i<n; i++){
        for(j=0; j<n-i; j++){
            if(jobs[j+1].profit > jobs[j].profit){
                temp = jobs[j+1];
                jobs[j+1] = jobs[j];
                jobs[j] = temp;
```

```
            }
        }
    }
    printf("%-10s %-10s %-10s \n", "Job", "Deadline", "Profit");
    for(i=0; i<n; i++) {
        printf("%-10s %-10i %-10i \n", jobs[i].id, jobs[i].deadline,
                jobs[i].profit);
    }
    jobSequencingWithDeadline(jobs, n);
    return 0;
}
void jobSequencingWithDeadline(Job jobs[], int n) {
    int i, j, k, maxprofit;
    int timeslot[MAX];
    int filledTimeslot = 0;
    int dmax = 0;
    for(i=0; i<n; i++) {
        if(jobs[i].deadline > dmax) {
            dmax = jobs[i].deadline;
        }
    }
    for(i=1; i<=dmax; i++) {
        timeslot[i] = -1;
    }
    printf("dmax : %d \n", dmax);
    for(i=1; i<=n; i++) {
        k = minValue(dmax, jobs[i-1].deadline);
        while(k>=1) {
            if(timeslot[k] == -1) {
                timeslot[k] = i-1;
                filledTimeslot++;
                break;
            }
            k--;
        }
        if(filledTimeslot == dmax) {
            break;
        }
    }
}
```

```
printf("\nRequired Jobs: ");
for(i=1; i<=dmax; i++){
    printf("%s", jobs[timeslot[i]].id);
    if(i<dmax){
        printf("-->");
    }
}
maxprofit = 0;
for(i=1; i<=dmax; i++){
    maxprofit += jobs[timeslot[i]].profit;
}
printf("\nMax Profit: %d\n", maxprofit);
}
```

Output:

Job	Deadline	Profit
j2	1	100
j1	2	60
j4	2	40
j3	3	20
j5	1	20

dmax: 3

Required Jobs: j2 --> j1 --> j3

Max Profit: 180

Time complexity: $O(N^2)$

Experiment - 9

Aim: Apply Strassen's Matrix Multiplication strategy for odd dimensional square matrices.

Apparatus Used: Windows 11, Dev c++

Source code:

```
#include<stdio.h>
#define Row_1 3
#define col_1 3
#define Row_2 3
#define col_2 3
void MatMultiplication(int Mat_A[][col_1], int Mat_B[][col_2]){
    int result[Row_1][col_2];
    for(int i=0; i<Row_1; i++){
        for(int j=0; j<col_2; j++){
            result[i][j]=0;
            for(int k=0; k<Row_2; k++){
                result[i][j] += Mat_A[i][k] * Mat_B[k][j];
            }
            printf("%d", result[i][j]);
        }
        printf("\n");
    }
}
int main(){
    int Mat_A[Row_1][col_1] = {{1,2,3},{4,5,6},{7,8,9}};
    int Mat_B[Row_2][col_2] = {{6,1,1},{9,2,4},{10,3,6}};
    if(col_1 != Row_2){
        printf("Matrix Multiplication not possible\n");
    }
    MatMultiplication(Mat_A, Mat_B);
    return 0;
}
```

Output:

```
54  14  27
129 32  60
204 50  93
```

Time complexity: $O(N^3)$

Experiment - 10

Aim: KMP string Matching: Given a text txt[0..n-1] and a pattern pat[0..m-1], write a function search(char pat[], char txt[]) that prints all occurrences of pat[] in txt[]. You may assume that n>m. Text: A A B A A C A A D A A B A A B A. Pattern: A A B A.

Apparatus Used: Windows 11, Dev c++

Source code:

```
#include<stdio.h>
#include<string.h>
void computeLPSArray(char* pat, int m, int* lps){
    int len = 0;
    lps[0] = 0;
    int i = 1;
    while(i<m){
        if(pat[i] == pat[len]){
            len++;
            lps[i] = len;
            i++;
        }
        else{
            if(len != 0){
                len = lps[len-1];
            }
            else{
                lps[i] = 0;
                i++;
            }
        }
    }
}
void KMPsearch(char* pat, char* txt){
    int M = strlen(pat);
    int N = strlen(txt);
    int cnt = 0;
    int lps[M];
    computeLPSArray(pat, M, lps);
    int i = 0;
    int j = 0;
```

```
    while (i < N) {
        if (pat[j] == txt[i]) {
            j++;
            i++;
        }
        if (j == M) {
            printf("Pattern found at index: %d\n", i-j);
            cnt++;
            j = lps[j-1];
        }
        else if (i<N && pat[j] != txt[i]){
            if (j != 0){
                j = lps[j-1];
            }
            else{
                i++;
            }
        }
    }
}
int main(){
    char txt[] = "AABAAC AAD AAB AABA";
    char pat[] = "AABA";
    KMPsearch(pat, txt);
    return 0;
}
```

Output:

Pattern found at index: 0
Pattern found at index: 9
Pattern found at index: 12
Occurrence of Pattern in Text: 3

Time complexity: O(N+M)

Experiment - 11

Aim: Implement AP strategy to solve the Travelling Salesman Problem (TSP).

Apparatus Used: Windows 11, Dev C++.

Source code:

```
#include<stdio.h>
const int n= 4;
const int MAX = 1000000;
int dist[n+1][n+1] ={{0,0,0,0,0},{0,0,10,15,20},{0,10,0,25,25},
                     {0,15,25,0,30},{0,20,25,30,0}};
int memo[n+1][1<<(n+1)];
int min(int a, int b){
    return a<b? a : b;
}
int fun(int i, int mask){
    if(mask == ((1<<i)|3)){
        return dist[1][i];
    }
    if(memo[i][mask] != 0){
        return memo[i][mask];
    }
    int res = MAX;
    for(int j = 1; j<= n; j++){
        if((mask & (1<<j)) && j != i && j != 1){
            res = min(res, fun(j, mask & (~(1<<i))) + dist[j][i]);
        }
    }
    return memo[i][mask] =res;
}
int main(){
    int ans = MAX;
    for(int i = 1; i<=n; i++){
        ans = min(ans, fun(i, (1<<(n+1))-1) + dist[i][1]);
    }
    printf("The cost of most efficient path: %d", ans);
    return 0;
}
```

Output:

The cost of most efficient path: 50

Time complexity: $O(n^2 * 2n)$