Experiment 1: Implement Selection sort using C language

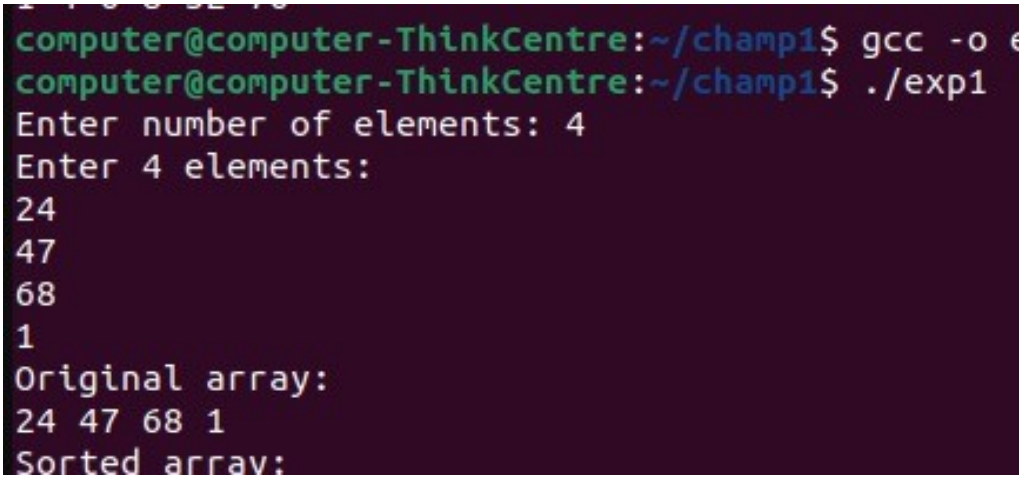
Program:

```
#include <stdio.h>

void selectionSort(int arr[], int n) {// Function to perform selection sort
   int i, j, min_idx, temp;
   for (i = 0; i < n - 1; i++) {
      min_idx = i;
      for (j = i + 1; j < n; j++) {
         if (arr[j] < arr[min_idx])
            min_idx = j;
      }
      if (min_idx != i) {
         temp = arr[i];
         arr[i] = arr[min_idx];
         arr[min_idx] = temp;
      }
   }
}
void printArray(int arr[], int n) {// Function to print an array
   int i;
   for (i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");
}
int main() {
   int n, i;
   printf("Enter number of elements: ");
   scanf("%d", &n);
```

```
    int arr[n];  // Create array of size n
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Original array: \n");
    printArray(arr, n);
    selectionSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Output:



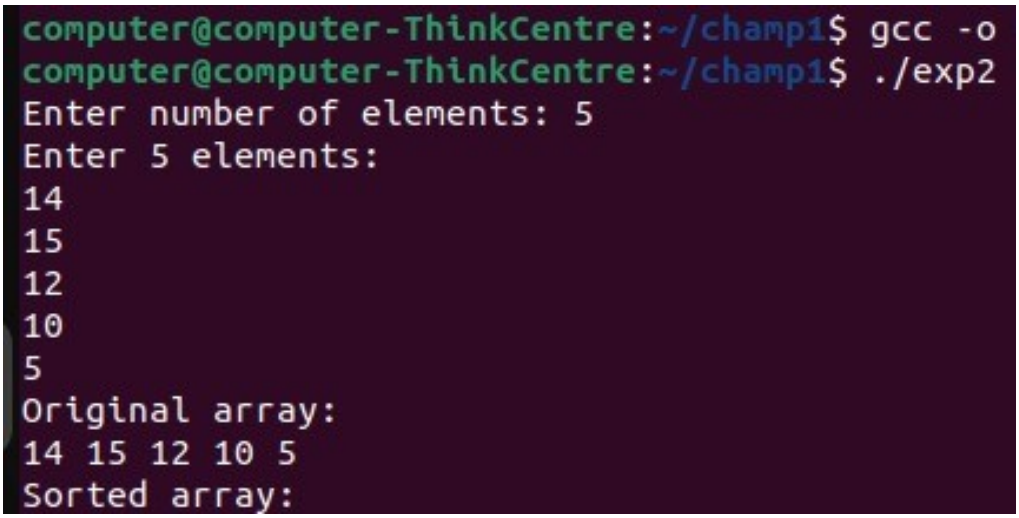
Experiment 2: Implement Insertion Sort using C language

Program:

```
#include <stdio.h>
// Function to perform insertion sort
void insertionSort(int arr[], int n) {
   int i, key, j;
   for (i = 1; i < n; i++)
   {
      key = arr[i]; // Element to be inserted
      j = i - 1;
      // Move elements of arr[0..i-1], that are greater than key,
      // to one position ahead of their current position
      while (j >= 0 && arr[j] > key)
      {
         arr[j + 1] = arr[j];
         j = j - 1;
      }
      arr[j + 1] = key;
   }
}
// Function to print an array
void printArray(int arr[], int n)
{
   int i;
   for (i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");
}
```

```
int main()
{
   int n, i;
   printf("Enter number of elements: ");
   scanf("%d", &n);
   int arr[n];  // Create array of size n
   printf("Enter %d elements:\n", n);
   for (i = 0; i < n; i++)
   {
      scanf("%d", &arr[i]);
   }
   printf("Original array: \n");
   printArray(arr, n);
   insertionSort(arr, n);
   printf("Sorted array: \n");
   printArray(arr, n);
   return 0;
}
```

Output:



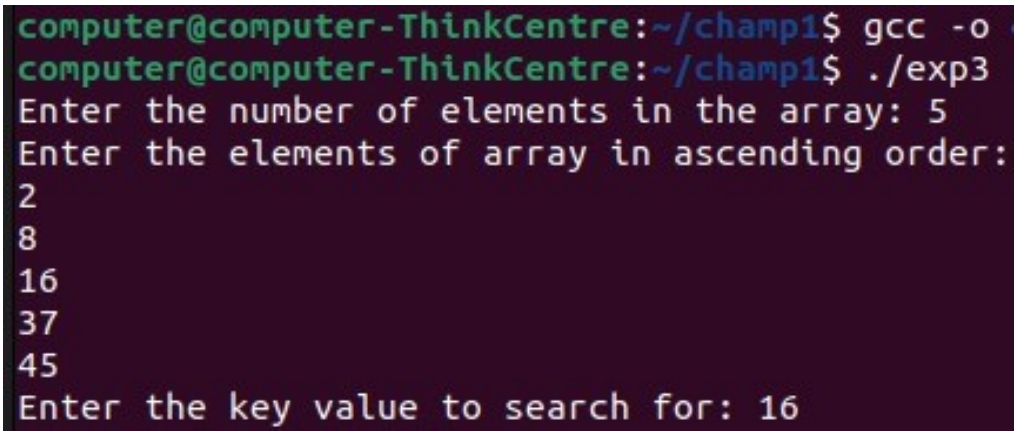
Experiment 3: Implement C Program for Binary Search using Divide and Conquer Approach

Program:

```
#include<stdio.h>
//Function to perform binary search
int binarySearch(int arr[], int left, int right, int key){
	if(left>right)//base case
		return -1;
	int mid=left+(right-left)/2;
	//if key is at middle index, return the index
	if(arr[mid]==key)
		return mid;
	//If key is smaller than middle element
	if(arr[mid]>key)
		return binarySearch(arr, left, mid-1, key);
	//If key is larger than middle element
	return binarySearch(arr, mid+1, right, key);}
int main(){
	int n, key;
	printf("Enter the number of elements in the array: ");
	scanf("%d", &n);
	int arr[n];
	printf("Enter the elements of array in ascending order:\n");
	for(int i=0;i<n;i++)
		scanf("%d", &arr[i]);
	printf("Enter the key value to search for: ");
	scanf("%d", &key);
	//Call binarySearch to search for key
	int result=binarySearch(arr,0, n-1, key);
```

```
    //Print the result
    if(result!=-1)
        printf("Element found at index: %d\n", result);
    else
        printf("Element not found.\n");
    return 0;
}
```

Output:



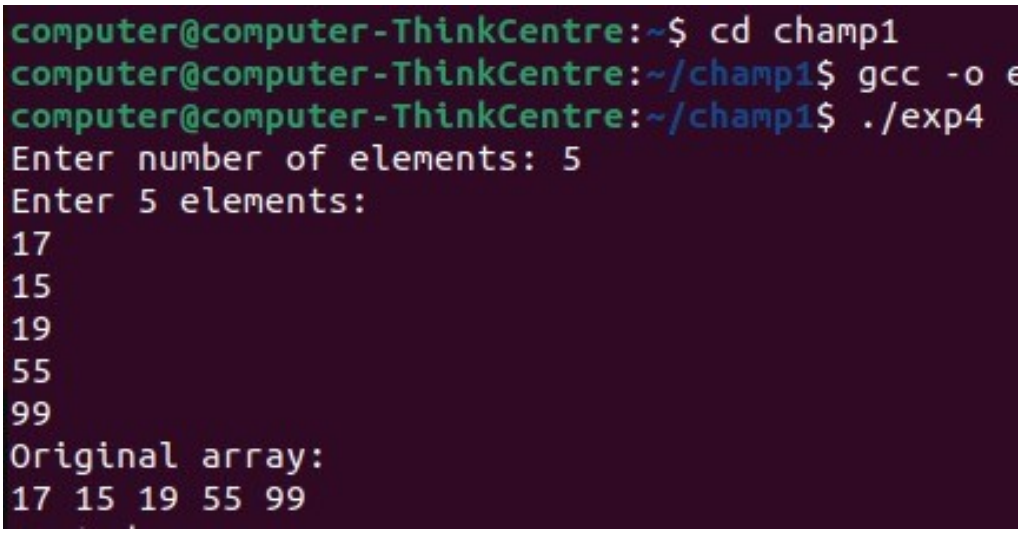
Experiment 4: Implement Merge sort using C language

Program:

```
#include <stdio.h>
// Function to merge two halves
void merge(int arr[], int left, int mid, int right) {
   int n1 = mid - left + 1; // Size of left subarray
   int n2 = right - mid;    // Size of right subarray
   int L[n1], R[n2]; // Temporary arrays
   // Copy data to temp arrays
   for (int i = 0; i < n1; i++)
      L[i] = arr[left + i];
   for (int j = 0; j < n2; j++)
      R[j] = arr[mid + 1 + j];
   int i = 0, j = 0, k = left;
   // Merge the temp arrays back into arr
   while (i < n1 && j < n2) {
      if (L[i] <= R[j]) {
         arr[k] = L[i];
         i++;}
   else {
         arr[k] = R[j];
         j++;}
      k++;}
   // Copy any remaining elements of L[]
   while (i < n1) {
      arr[k] = L[i];
      i++;
      k++;}
   // Copy any remaining elements of R[]
   while (j < n2) {
      arr[k] = R[j];
      j++;
      k++;}
}
// Function to perform merge sort
void mergeSort(int arr[], int left, int right) {
   if (left < right) {
      int mid = left + (right - left) / 2; // Avoids overflow
      mergeSort(arr, left, mid);  // Sort first half
      mergeSort(arr, mid + 1, right); // Sort second half
```

```
        merge(arr, left, mid, right);   // Merge the sorted halves
    }
}
// Function to print an array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    printf("Original array:\n");
    printArray(arr, n);
    mergeSort(arr, 0, n - 1);
    printf("Sorted array:\n");
    printArray(arr, n);
    return 0;
}
```

Output:



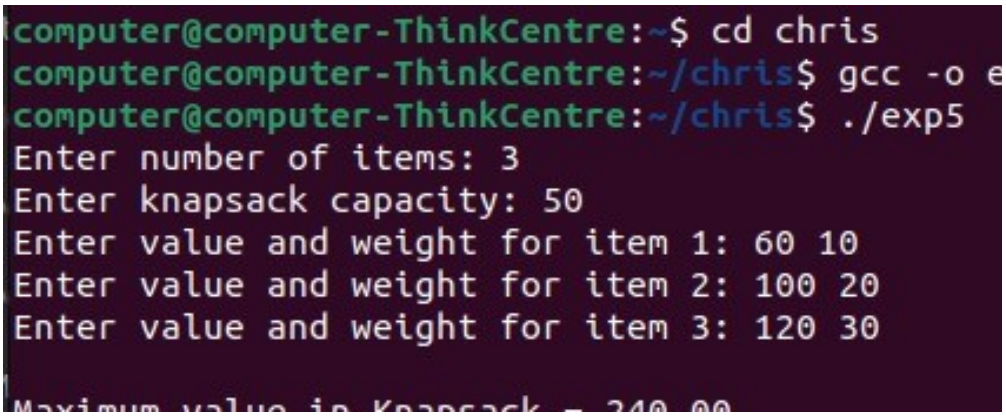
Experiment 5: Implement Fractional Knapsack problem using C language

Program:

```
#include <stdio.h>
#include <stdlib.h>
struct Item {
   int value, weight;
};
// Compare items by value/weight ratio (descending)
int cmp(const void* a, const void* b) {
   double r1 = (double)((struct Item*)a)->value / ((struct Item*)a)->weight;
   double r2 = (double)((struct Item*)b)->value / ((struct Item*)b)->weight;
   return (r2 > r1) - (r2 < r1);
}
double knapsack(int cap, struct Item arr[], int n) {
   qsort(arr, n, sizeof(struct Item), cmp);
   double val = 0.0;
   for (int i = 0; i < n && cap > 0; i++) {
      if (arr[i].weight <= cap) {
         cap -= arr[i].weight;
         val += arr[i].value;
      }
      else {
         val += arr[i].value * (double)cap / arr[i].weight;
         break;
      }
   }
   return val;
}
int main() {
```

```
    int n, cap;
    printf("Enter number of items: ");
    scanf("%d", &n);
    printf("Enter knapsack capacity: ");
    scanf("%d", &cap);
    struct Item items[n];
    for (int i = 0; i < n; i++) {
        printf("Enter value and weight for item %d: ", i + 1);
        scanf("%d %d", &items[i].value, &items[i].weight);
    }
    printf("\nMaximum value in Knapsack = %.2f\n", knapsack(cap, items, n));
    return 0;
}
```

Output:



Experiment 6: Implement Minimum cost spanning tree-Kruskal using C language

Program:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct { int u,v,w; } Edge;
int p[100];
int find(int x)
{
        return p[x]==x?x:(p[x]=find(p[x]));
}
int cmp(const void *a,const void *b)
{
        return ((Edge*)a)->w-((Edge*)b)->w;
}
int main(){
   int V,E;
   printf("Enter the number of vertices: ");
   scanf("%d",&V);
   printf("Enter the number of edges: ");
   scanf("%d",&E);
   Edge e[E];
   printf("Enter each edge as: source destination weight\n");
   for(int i=0;i<E;i++){
      printf("Edge %d: ",i+1);
      scanf("%d%d%d",&e[i].u,&e[i].v,&e[i].w);
   }
   for(int i=0;i<V;i++)
        p[i]=i;
   qsort(e,E,sizeof(Edge),cmp);
```

```
    int cost=0,c=0;
    printf("\nEdges in the Minimum Spanning Tree:\n");
    for(int i=0;i<E&&c<V-1;i++)
        if(find(e[i].u)!=find(e[i].v)){
            printf("%d -- %d == %d\n",e[i].u,e[i].v,e[i].w);
            cost+=e[i].w; p[find(e[i].u)]=find(e[i].v); c++;
        }
    printf("Total cost of MST: %d\n",cost);
}
```

Output:

```
computer@computer-ThinkCentre:~$ cd chris
computer@computer-ThinkCentre:~/chris$ gcc -o exp6 exp6.c
computer@computer-ThinkCentre:~/chris$ ./exp6
Enter the number of vertices: 4
Enter the number of edges: 5
Enter each edge as: source destination weight
Edge 1: 0 1 10
Edge 2: 0 2 6
Edge 3: 0 3 5
Edge 4: 1 3 15
Edge 5: 2 3 4

Edges in the Minimum Spanning Tree:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Total cost of MST: 19
computer@computer-ThinkCentre:~/chris$
```

Experiment 7: Implement All pair shortest path – Floyd Warshall using C language

Program:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define INF 99999
void floydWarshall(int n, int dist[][n]) {
   for (int k = 0; k < n; k++)
      for (int i = 0; i < n; i++)
         for (int j = 0; j < n; j++)
            if (dist[i][k] + dist[k][j] < dist[i][j])
               dist[i][j] = dist[i][k] + dist[k][j];
   printf("\nShortest distances between every pair of vertices:\n");
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         if (dist[i][j] == INF)
            printf("%5s", "INF");
         else
            printf("%5d", dist[i][j]);
      }
      printf("\n");
   }
}
int main() {
   int n;
   printf("Enter number of vertices: ");
   scanf("%d", &n);

   int graph[n][n];
```

```
    char temp[20];
    printf("Enter adjacency matrix (use INF for no edge):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%s", temp);
            if (strcmp(temp, "INF") == 0)
                graph[i][j] = INF;
            else
                graph[i][j] = atoi(temp);
        }
    }
    floydWarshall(n, graph);
    return 0;
}
```

Output:

```
(base) computer@computer-ThinkCentre:~$ cd champ
(base) computer@computer-ThinkCentre:~/champ$ gcc -o
(base) computer@computer-ThinkCentre:~/champ$ ./exp7
Enter number of vertices: 4
Enter adjacency matrix (use INF for no edge):
0 2 INF INF
INF 0 3 INF
INF INF 0 1
6 INF INF 0

Shortest distances between every pair of vertices:
     0     2     5     6
    10     0     3     4
```

Experiment 8: Implement Longest common subsequence using C language

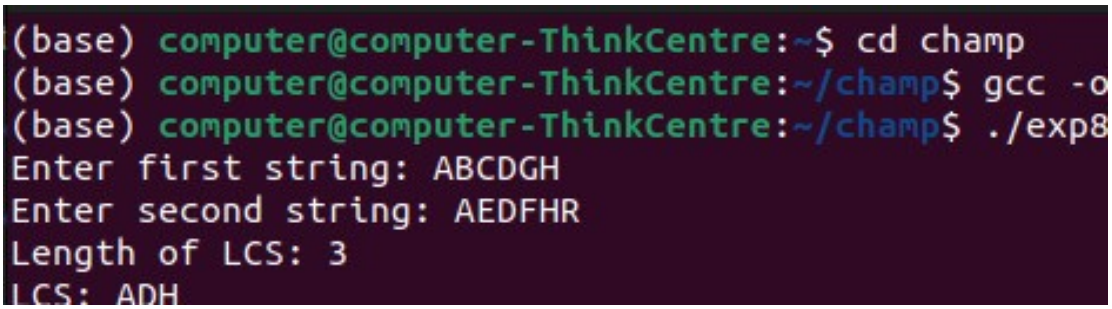
Program:

```
#include <stdio.h>
#include <string.h>
// Function to find max of two numbers
int max(int a, int b) {
    return (a > b) ? a : b;
}
// Function to compute LCS
void LCS(char X[], char Y[]) {
    int m = strlen(X);
    int n = strlen(Y);
    int dp[m + 1][n + 1];
    // Build LCS table in bottom-up manner
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0)
                dp[i][j] = 0;
            else if (X[i - 1] == Y[j - 1])
                dp[i][j] = 1 + dp[i - 1][j - 1];
            else
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
    }
    // Length of LCS
    int length = dp[m][n];
    printf("Length of LCS: %d\n", length);

    // Backtrack to find LCS string
    char lcs[length + 1];
```

```
    lcs[length] = '\0';
    int i = m, j = n, index = length - 1;
    while (i > 0 && j > 0) {
        if (X[i - 1] == Y[j - 1]) {
            lcs[index--] = X[i - 1];
            i--; j--;
        } else if (dp[i - 1][j] > dp[i][j - 1])
            i--;
        else
            j--;
    }
    printf("LCS: %s\n", lcs);
}
int main() {
    char X[100], Y[100];
    printf("Enter first string: ");
    scanf("%s", X);
    printf("Enter second string: ");
    scanf("%s", Y);
    LCS(X, Y);
    return 0;
}
```

Output:



Experiment 9: Implement Sum of subsets using C language

Program:

```
#include <stdio.h>
int n, target;
int set[20], subset[20];
void displaySubset(int size) {
   printf("{ ");
   for (int i = 0; i < size; i++)
      printf("%d ", subset[i]);
   printf("}\n");
}
void sumOfSubsets(int index, int currentSum, int subsetSize) {
   if (currentSum == target) {
      displaySubset(subsetSize); // Found a valid subset
      return;
   }
   if (index == n || currentSum > target)
      return;
   // Include current element
   subset[subsetSize] = set[index];
   sumOfSubsets(index + 1, currentSum + set[index], subsetSize + 1);
   // Exclude current element
   sumOfSubsets(index + 1, currentSum, subsetSize);
}
int main() {
   printf("Enter number of elements: ");
   scanf("%d", &n);
   printf("Enter elements of the set:\n");
   for (int i = 0; i < n; i++)
```

```
        scanf("%d", &set[i]);
    printf("Enter target sum: ");
    scanf("%d", &target);
    printf("Subsets with sum %d are:\n", target);
    sumOfSubsets(0, 0, 0);
    return 0;
}
```

Output:

```
(base) computer@computer-ThinkCentre:~$ cd champ
(base) computer@computer-ThinkCentre:~/champ$ gcc -o
(base) computer@computer-ThinkCentre:~/champ$ ./exp9
Enter number of elements: 5
Enter elements of the set:
1 2 3 4 5
Enter target sum: 6
Subsets with sum 6 are:
{ 1 2 3 }
{ 1 5 }
```

Experiment 10: Implement Naïve string-matching Algorithms using C language

Program:

```
#include <stdio.h>
#include <string.h>
void naiveStringMatch(char text[], char pattern[]) {
   int n = strlen(text);
   int m = strlen(pattern);
   printf("Pattern found at positions: ");
   int found = 0;
   for (int i = 0; i <= n - m; i++) {
      int j;
      for (j = 0; j < m; j++) {
         if (text[i + j] != pattern[j])
            break;
      }
      if (j == m) {
         printf("%d ", i);  // pattern found at index i
         found = 1;
      }
   }
   if (!found)
      printf("None");
   printf("\n");
}
int main() {
   char text[100], pattern[50];
   printf("Enter the text: ");
   scanf("%[^\n]", text);  // read full line including spaces
   getchar();              // consume newline
```

```
    printf("Enter the pattern to search: ");
    scanf("%[^\n]", pattern);
    naiveStringMatch(text, pattern);
    return 0;
}
```

Output: