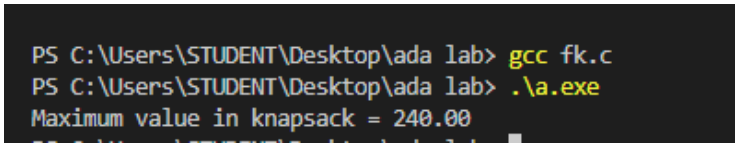
ADA labs

1. Implement fractional knapsack problem using Greedy technique.

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

typedef struct {
    int weight, value;
    float ratio;
} Item;

int compare(const void *a, const void *b) {
    Item *i1 = (Item *)a;
    Item *i2 = (Item *)b;
    return (i2->ratio > i1->ratio) - (i2->ratio < i1->ratio);
}

float fractionalKnapsack(Item items[], int n, int capacity) {
    qsort(items, n, sizeof(Item), compare);
    float totalValue = 0.0;

    for (int i = 0; i < n && capacity > 0; i++) {
        if (items[i].weight <= capacity) {
            totalValue += items[i].value;
            capacity -= items[i].weight;
        } else {
            totalValue += items[i].ratio * capacity;
            break;
        }
    }

    return totalValue;
}

int main() {
    int n = 3, capacity = 50;
    Item items[] = {{10, 60}, {20, 100}, {30, 120}};
    for (int i = 0; i < n; i++)
        items[i].ratio = (float)items[i].value / items[i].weight;

    float maxValue = fractionalKnapsack(items, n, capacity);
    printf("Maximum value in knapsack = %.2f\n", maxValue);
    return 0;
}
```

OUTPUT:



2. Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

```
#include <stdio.h>

void heapify(int arr[], int n, int i) {
    int largest = i, left = 2*i + 1, right = 2*i + 2;

    if (left < n && arr[left] > arr[largest]) largest = left;
    if (right < n && arr[right] > arr[largest]) largest = right;

    if (largest != i) {
        int temp = arr[i]; arr[i] = arr[largest]; arr[largest] = temp;
        heapify(arr, n, largest);
    }
}

// Heap sort function
void heapSort(int arr[], int n) {
    for (int i = n/2 - 1; i >= 0; i--) heapify(arr, n, i);
    for (int i = n-1; i >= 0; i--) {
        int temp = arr[0]; arr[0] = arr[i]; arr[i] = temp;
        heapify(arr, i, 0);
    }
}

int main() {
    int n, i;
    printf("Enter the number of elements to sort : ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter the elements to sort : ");
    for (i = 0; i < n; i++) scanf("%d", &arr[i]);

    heapSort(arr, n);

    printf("The sorted list of elements is : ");
    for (i = 0; i < n; i++) printf("%d ", arr[i]);
    printf("\n");

    return 0;
}
```

OUTPUT:

```
PS C:\Users\STUDENT\Desktop\ada lab> gcc hs.c
PS C:\Users\STUDENT\Desktop\ada lab> .\a.exe
Enter the number of elements to sort: 5
Enter the elements to sort: 8 5 6 3 1
The sorted list of elements is: 1 3 5 6 8
```

3. Implement “N-Queens Problem” using Backtracking

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int x[20],count=1;
void queens(int,int);
int place(int,int);


void main()
{
int n,k=1;
clrscr();
printf("\n enter the number of queens to be placed\n");
scanf("%d",&n);
queens(k,n);
}
void queens(int k,int n)
{
int i,j;
for(j=1;j<=n;j++)
{
if(place(k,j))
{
x[k]=j;
if(k==n)
{
printf("\n %d solution",count);
count++;
for(i=1;i<=n;i++)
printf("\n \t %d row <---> %d
  column",i,x[i]);
getch();
}
else
queens(k+1,n);
}
}
}
int place(int k,int j)
{
int i;
for(i=1;i<k;i++)
if((x[i]==j) || (abs(x[i]-j))==abs(i-k))
return 0;
return 1;
}
```

OUTPUT:

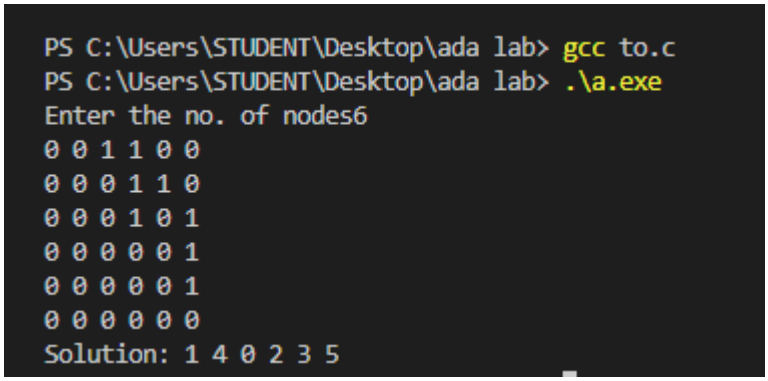
```
Enter the number of queens to be placed: 4

Solution 1:
Row 1 <--> Column 2
Row 2 <--> Column 4
Row 3 <--> Column 1
Row 4 <--> Column 3

Solution 2:
Row 1 <--> Column 3
Row 2 <--> Column 1
Row 3 <--> Column 4
Row 4 <--> Column 2
```

4. Write program to obtain the Topological ordering of vertices in a given digraph.

```
#include <stdio.h>


int n, a[10][10], res[10], s[10], top = 0;


void dfs(int, int, int[][10]);
void dfs_top(int, int[][10]);


int main()
{
   printf("Enter the no. of nodes");
   scanf("%d", &n);


   int i, j;
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         scanf("%d", &a[i][j]);
      }
   }


   dfs_top(n, a);
   printf("Solution: ");
   for (i = n - 1; i >= 0; i--) {
      printf("%d ", res[i]);
   }
```

```
    return 0;
}


void dfs_top(int n, int a[][10]) {
    int i;
    for (i = 0; i < n; i++) {
        s[i] = 0;
    }
    for (i = 0; i < n; i++) {
        if (s[i] == 0) {
            dfs(i, n, a);
        }
    }
}


void dfs(int j, int n, int a[][10]) {
    s[j] = 1;
    int i;
    for (i = 0; i < n; i++) {
        if (a[j][i] == 1 && s[i] == 0) {
            dfs(i, n, a);
        }
    }
    res[top++] = j;
}
```

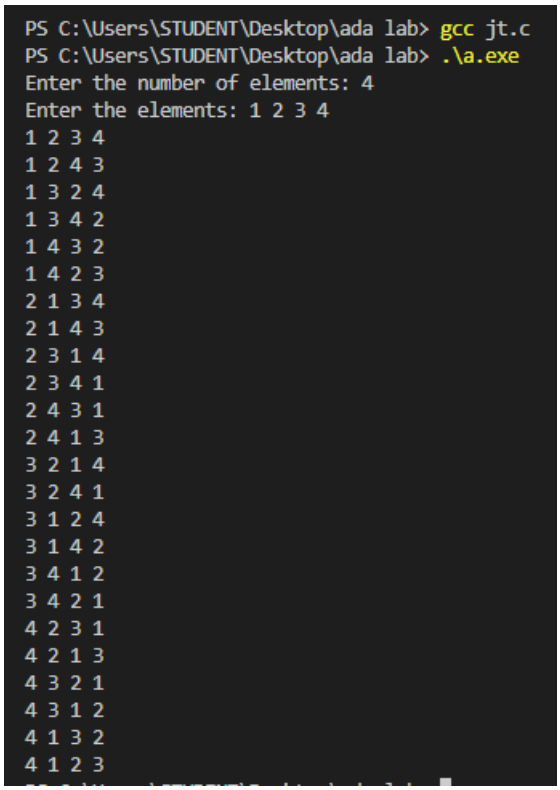
OUTPUT:



5. Implement Johnson Trotter algorithm to generate permutations.

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

void swap(int* a, int* b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}

void generatePermutations(int arr[], int start, int end) {
   if (start == end) {
      for (int i = 0; i <= end; i++) {
         printf("%d ", arr[i]);
      }
      printf("\n");
   } else {
      for (int i = start; i <= end; i++) {
         swap(&arr[start], &arr[i]);
         generatePermutations(arr, start + 1, end);
         swap(&arr[start], &arr[i]); // backtrack
      }
   }
}

int main() {
   int n;
   printf("Enter the number of elements: ");
   scanf("%d", &n);

   int* arr = (int*)malloc(n * sizeof(int));
   printf("Enter the elements: ");
   for (int i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   generatePermutations(arr, 0, n - 1);

   free(arr);

   return 0;
}
```

OUTPUT:


PS C:\Users\STUDENT\Desktop\ada lab> gcc jt.c
PS C:\Users\STUDENT\Desktop\ada lab> .\a.exe
Enter the number of elements: 4
Enter the elements: 1 2 3 4
1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 3 2
1 4 2 3
2 1 3 4
2 1 4 3
2 3 1 4
2 3 4 1
2 4 3 1
2 4 1 3
3 2 1 4
3 2 4 1
3 1 2 4
3 1 4 2
3 4 1 2
3 4 2 1
4 2 3 1
4 2 1 3
4 3 2 1
4 3 1 2
4 1 3 2
4 1 2 3