Experiment #9		Student ID	
Date		Student Name	

- **Procedure/Program:**

```
#include <stdio.h>
int binomial (int n, int k){
if ( k==0) || (k==n) return 1;
return binomial (n-1, k-1) + binomial
                              (n-1, k);
}
int main (){
    int T, N, P, result;
    Scanf ("%d", &T);
    while (T--){
        Scanf ("%d %d", &N, &P);
            result =0;
    for(int k=0; k<=P; k++){
        result += binomial(N, k);
    }
    Printf ("%d\n", result);
    }
    return 0;
}
```

- **Data and Results:**

- **Analysis and Inferences:**

Data

:

Test cases include values for N and P

Result

:

Sum of subsets for given N and P

Analysis

:

Recursive computation of binomial coefficients provides required values efficiently

Inferences

:

Dynamic Programming optimizes combina~~tion~~ orial calculations for large inputs

2. Given N cities numbered from 1 to N. Your task is to visit the cities. Here K cities are already visited. You can visit i^{th} city if $(i-1)^{th}$ or $(i+1)^{th}$ city is already visited. Your task is to determine the number of ways you can visit all the remaining cities.

Input format:

First line: Two space-separated integers N and K

Second line: K space-separated integers each denoting the city that is already visited

Output format:

Print an integer denoting the number of ways to visit the remaining cities.

Sample Input:

6 3

1 2 6

Sample Output:

4 ({3, 4, 5}, {3, 5, 4}, {5, 3, 4}, {5, 4, 3})

- **Procedure/Program:**

```
# include <stdio.h>
#include <stdlib.h>
long long factorial (int n){
    long long result = 1;
    for (int i = 2; i<=n; i++)
        result *= i;
    return result;
}
int main(){
    int N, K;
```

```
scanf("%d %d", &N, &k);
int visited[k];

for (int i =0; i<k; i++)
scanf("%d", &visited[i]);

long long totalways =1;

for (int i = 0; i<=k ; i++){
int left = (i== 0) ? 0 : visited[i-1];
int right = (i==k) ? N+1: visited[i];
int unvisited count = right - left -1;

if(unvisited count >0)

totalways *= factorial(unvisited count);

}
printf("%lld\n", totalways);

return 0;

}
```

Experiment #9		Student ID	
Date		Student Name	

- **Data and Results:**

Data
⇒ cities numbered 1 to N with k already visited cities

Result
⇒ Number of unique ways to visit remaining unvisited cities

- **Analysis and Inferences:**

Analysis
⇒ segment unvisited cities between visited ones; calculate arrangements.

In-Lab:

1. Given an undirected graph and N colors, the problem is to find if it is possible to color the graph with at most N colors, which means assigning colors to the vertices of the graph such that no two adjacent vertices of the graph are colored with the same color. Print "Possible" if it is possible to color the graph as mentioned above, else print "Not Possible".

Input

1
3 2
0 2
1 2
2

Output

Possible

```
#include <stdio.h>
 #define   MAX 100
int graph [MAX] [MAX],    color [MAX];
int N, M
 int is safe (int  v,  int c) {
 for (int  i =0;    i<N ;   i++)
 if (graph [v] [i]  &&   color [i]  == c)
       return 0;
    return  1;
}
int    graph coloring util1 (int  v) {
   if (v == N)   return 1;
 for (int  c  = 1;    c<= M  ; c++) {
        if (issafe (v, c)) {
    color [v]  = c;
 if (graph coloring util (v+1))    return1;
           color [v]= 0;
           }
   }
     return  0;
  }
```

- **Procedure/Program:**

```
int main(){
    scanf ("%d %d", &N, &M);
for (int u, v; scanf ("%d %d", &u,&v)
                        != EOF;){
 graph [u] [v] = graph [v] [u] = 1;
 }
 printf("%s\n", graphcoloringutil(0)?
         "Possible" : "Not possible");

    return 0;
    }
```

- **Data and Results:**

graph with vertices and edges; coloring possible with given constraints

- **Analysis and Inferences:**

Backtracking confirms feasible coloring when adjacent vertices have different

colors.

2. Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum. A subarray is a contiguous part of an array.

Example 1:

Input: nums = [-2,1,-3,4,-1,2,1,-5,4]

Output: 6

Explanation: [4,-1,2,1] has the largest sum = 6.

Example 2:

Input: nums = [1]

Output: 1

Example 3:

Input: nums = [5,4,-1,7,8]

Output: 23

- **Procedure/Program:**

```
#include <stdio.h>
int maxSubArray(int* nums, int size){
    int max_sum = nums[0];
    int current_sum = nums[0];

  for(int i = 1; i < size; i++){

  current_sum = current_sum > 0 ? current_sum
      + nums[i] : nums[i];

  max_sum = current_sum > max_sum ?
                  current_sum : max_sum;
  }
  return max_sum;
```



```
}
```

```
int  main(){
int nums[] = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
int size = sizeof (nums) / sizeof (nums[0]);

int result = maxsubArray (nums, size);
Printf("%d\n", result);
  return 0;
}
```

- **Data and Results:**

Input array contains integers; largest sum of contiguous subarray is 6

- **Analysis and Inferences:**

kadane's algorithm efficiently finds maximum contiguous subarray sum

Post-Lab:

Karthik was given a problem in an online interview, but he cannot solve the solution help him solve the question. There are n students whose ids vary between 0 to 9 digits; two students can have same id's. You will be given x numbers which also vary from 0 to 9. You need to find the total number of student id's subsets which contains all the x numbers.

Input:

Student ID's: "333", "1", "3"

X = {3, 1, 3}

Output

6

- **Procedure/Program:**

```
#include <stdio.h>
#include <string.h>
int countsubsets (char ids[][10], int n,
        int x[], int xsize){
  int counts [10] = {0};
  for(int i=0; i<n; i++){
```

```
for (int j = 0;   j < strlen (ids [i]) ; j++){

    counts [ids [i] [j]  - '0' ] ++;
  }
 }
long  long  result  =1;
for (int i=0;   i< xsize ; i++){
 result * =  ((counts [x] [i]]  >0) ?  (1 <<
                                   counts [x [i]])
                                        : 0;
 }
return  result;
}
int main(){
char  studentIDs [][10]  = {"333",  "1", "3"};
int x []= {3, 1, 3};
int n = sizeof (studentIDs) /sizeof (studentIDs
                                       [0]);
int  xsize = sizeof(x) /sizeof (x[0]);
int  totalSubsets  =  countSubsets (studentIDs,
                  n, x,  xsize);
Printf ("%d\n",   totalSubsets);
     return 0;
  }
```

- **Data and Results:**

Input IDS: "333", "1", "3"

output: 6 valid subsets

- **Analysis and Inferences**

counted occurrences, calculated subsets; subsets include required numbers efficiently.

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

1. Can you explain the graph coloring problem and its significance?
2. How does backtracking help in solving the graph coloring problem?
3. What is dynamic programming, and how is it applied to the graph coloring problem?
4. Explain the subset sum problem and its relevance.
5. How does backtracking assist in solving the subset sum problem?

Evaluator Remark (if Any):	**Marks Secured:____out of 50**
	Signature of the Evaluator with Date

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

1) graph coloring assigns colors ensuring adjacent nodes differ significantly

2) Backtracking explores all coloring possibilities while discarding invalid configurations.

3) Dynamic Programming stores solutions for overlapping subproblems in graph coloring

4) subset sum finds if any subset matches a given sum.

5) Backtracking explores possible subsets, backtracks on exceeding target sum.