

Date of the Session: ____/____/____

Time of the Session: ____to____

EX – 1 Analysis of Time and Space Complexity of Algorithms**Prerequisites:**

- Basics of Data Structures and C Programming.
- Basic knowledge about algorithms in C and Data Structures.

Pre-Lab:

- 1) During lockdown Mothi gets bored by his daily routine while scrolling youtube he found an algorithm that looks different Mothi is very crazy about algorithms, but he cannot solve algorithms of multiple loops so that he structed and need your help to find the time complexity of that algorithm

Algoritm KLU(int n)

```

{
    int count=0;
    for(int i=0;i<n;i=i*2)
    {
        for(int j=n;j>0;j=j/3)
        {
            for(int k=0;k<n;k++)
            {
                Count++;
            }
        }
    }
}

```

1. Outer loop ($i = i * 2$) runs $O(\log n)$ times.

2. Middle loop ($j = j / 3$) runs $O(\log n)$ times.

3. Inner loop ($k = 0; k < n$) runs $O(n)$ times.

So, the total time complexity is:

$$O(n \cdot (\log n)^2)$$

- 2) Klaus Michaelson is interviewer, so he prepared a bunch of questions for the students. He focused on algorithms. Among all the questions the easiest question is what is the time complexity of the following C function is so can answer to this?

```
int recursive(int n)
{
    if(n==1)
        return (1);
    else
        return(recursive (n-1) + recursive (n-1));
}
```

Function Behavior:

- **Base Case:** When `n == 1`, the function returns `1`, which takes constant time ($O(1)$).
- **Recursive Case:** For `n > 1`, the function calls itself twice with the argument `n - 1`.

Thus, the recursive call structure is as follows:

- `recursive(n)` makes two recursive calls to `recursive(n-1)`.

Recurrence Relation:

Let the time complexity of `recursive(n)` be denoted as $T(n)$.

- For $n == 1$, $T(1) = O(1)$.
- For $n > 1$, the function calls `recursive(n-1)` twice, so the recurrence relation becomes:

$$T(n) = 2 \cdot T(n - 1) + O(1)$$

This recurrence describes a **binary recursion** where the number of calls doubles at each level.

Solving the Recurrence:

Using the recurrence:

$$T(n) = 2 \cdot T(n - 1) + O(1)$$

Expanding this:

$$T(n) = 2 \cdot (2 \cdot T(n - 2) + O(1)) + O(1) = 2^2 \cdot T(n - 2) + 2 \cdot O(1) + O(1)$$

$$T(n) = 2^3 \cdot T(n - 3) + 2^2 \cdot O(1) + 2 \cdot O(1) + O(1)$$

After expanding this for k levels, we get:

$$T(n) = 2^n \cdot T(1) + O(1) + O(2) + O(4) + \dots + O(2^{n-1})$$

The total work done across all levels of recursion is:

$$T(n) = O(2^n)$$

Time Complexity:

The time complexity of the function is $O(2^n)$, because the number of calls grows exponentially with n .

- 3) Mothi has 2 algorithms and he also know their time functions. Now he wants to analyze which one is greater function? But Mothi do not know how to compare two-time functions. Your task is to analyze the given two-time functions and judge which one is greater and justify with your solution

a) $T1(n) = (\log(n^2) * \log(n))$

$T2(n) = \log(n * (\log n)^{10})$

b) $T1(n) = 3 * (n)^{(\sqrt{n})}$

$T2(n) = (2)^{(\sqrt{n} * \log n)}$ (consider $\log n$ with base 2)

Problem a) Compare:

1. $T1(n) = \log(n^2) * \log(n)$

2. $T2(n) = \log(n * (\log n)^{10})$

For $T1(n)$:

- $\log(n^2)$ can be simplified using logarithmic properties:

$$\log(n^2) = 2 \cdot \log(n)$$

- Therefore:

$$T1(n) = 2 \cdot \log(n) \cdot \log(n) = 2 \cdot (\log(n))^2$$

For $T2(n)$:

- $\log(n * (\log n)^{10})$:

Using the logarithmic property $\log(ab) = \log(a) + \log(b)$, we get:

$$\log(n \cdot (\log n)^{10}) = \log(n) + \log((\log n)^{10}) = \log(n) + 10 \cdot \log(\log n)$$

- Therefore:

$$T2(n) = \log(n) + 10 \cdot \log(\log n)$$

Comparison:

- $T1(n)$ grows as $(\log n)^2$, while $T2(n)$ grows as $\log n + 10 \cdot \log(\log n)$.
- For large values of n , $(\log n)^2$ grows much faster than $\log n$ or $\log(\log n)$, so $T1(n)$ is greater than $T2(n)$.

Problem b) Compare:

1. $T1(n) = 3 * (n)^{\sqrt{n}}$
2. $T2(n) = (2)^{\sqrt{n} * \log n}$

For $T1(n)$:

- $T1(n) = 3 \cdot n^{\sqrt{n}}$.

For $T2(n)$:

- $T2(n) = 2^{\sqrt{n} \cdot \log n}$. Using the property $a^{\log_b x} = x^{\log_b a}$, we can simplify this as:

$$2^{\sqrt{n} \cdot \log n} = n^{\sqrt{n}}$$

- Therefore:

$$T2(n) = n^{\sqrt{n}}$$

Comparison:

- $T1(n)$ is $3 \cdot n^{\sqrt{n}}$, which is essentially the same as $T2(n)$, but scaled by a constant factor (3). Since constant factors do not affect the asymptotic growth, $T1(n) \approx T2(n)$ in terms of their growth rate.

In -Lab:

- 1) Caroline Forbes is an intelligent girl, every time she wins in any contest or programme, and solves complex problems so I want to give her a challenge problem that is. Sort an array of strings according to string lengths. If you are smarter than her, try to solve the problem faster than her?

Input

You are beautiful looking

Output

You are looking beautiful

Source code:

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

int compare(const void *a, const void *b) {
    return strlen(*(char **)a) - strlen(*(char **)b);
}

int main() {
    char *arr[] = {"You", "are", "beautiful", "looking"};
    int n = sizeof(arr) / sizeof(arr[0]);

    qsort(arr, n, sizeof(char *), compare);

    for (int i = 0; i < n; i++)
        printf("%s ", arr[i]);

    return 0;
}
```

- 2) Stefan is the Assistant Professor of Computer science department so he just ask to interact with the students and helps the students to solve the complex problems in an easier way so, the problem given by the sir is sort an array according to count of set bits?

Example:

Input: arr[] = {1, 2, 3, 4, 5, 6};

Output: 3 5 6 1 2 4

Explanation:

3 - 0011

5 - 0101

6 - 0110

1 - 0001

2 - 0010

4 - 0100

hence the non-increasing sorted order is

{3, 5, 6}, {1, 2, 4}

Source code:

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

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

```
int countSetBits(int n) {
```

```
    return __builtin_popcount(n);
```

```
}
```

```
int compare(const void *a, const void *b) {
```

```
    int bitsA = countSetBits(*(int *)a);
```

```
    int bitsB = countSetBits(*(int *)b);
```

```
    return bitsB - bitsA ? bitsB - bitsA : *(int *)a - *(int *)b;
```

```
}
```

```
int main() {
```

```
    int arr[] = {1, 2, 3, 4, 5, 6};
```

```
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    qsort(arr, n, sizeof(int), compare);
```

```
    for (int i = 0; i < n; i++)
```

```
        printf("%d ", arr[i]);
```

```
    return 0;
```

```
}
```

- 3) During the final skill exam teacher was given a problem and asked everyone to write the algorithm to it, and advised that try to implement a different approach from others.
 Question: Write an algorithm to calculate sum of first 'n' natural numbers
 Mothi, one of the student in the class thinking that his friends will write an efficient algorithm to that question. So, he wants to write a worst approach to make that algorithm as unique.

Algorithm:

Algorithm SumOfNNaturalNums(int n)

```
{
    int count=0;
    for( i=1;i<=n;i++)
    {
        for(j=1;j<=i;j++)
        {
            count++;
        }
    }
    return count;
}
```

Source code:

- **Objective:** Calculate sum of first 'n' natural numbers.
- **Algorithm:**
 - Outer loop runs from 1 to 'n'.
 - Inner loop runs from 1 to 'i' (i is current value of outer loop).
 - Increment `count` on each inner loop iteration.
- **Time complexity:** $O(n^2)$ due to nested loops.
- **Result:** The total iterations are the sum of the first 'n' integers: $1 + 2 + 3 + \dots + n$.
- **Inefficiency:** Direct formula $\frac{n(n+1)}{2}$ is more efficient ($O(1)$).

Post-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)]

“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

Source code:

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

int countUppercase(char *str) {
    int count = 0;
    while (*str) {
        if (isupper(*str)) count++;
        str++;
    }
    return count;
}

int compare(const void *a, const void *b) {
    return countUppercase(((char **)a)[0]) - countUppercase(((char **)b)[0]);
}

int main() {
    char *arr[] = {"poiNtEr", "aRRaY", "cOde", "foR"};
    int n = sizeof(arr) / sizeof(arr[0]);

    qsort(arr, n, sizeof(char *), compare);
    for (int i = 0; i < n; i++) {
        printf("(%s', %d)\n", arr[i], countUppercase(arr[i]));
    }

    return 0;
}
```

- 2) In KLU streets we have lots of electrical poles.
 Note: all poles are sorted with respect to their heights.

Professor Stefan given the H = height of one pole to Mothi then asked him to print the position of that pole, here we consider index as a position. Mothi is particularly good at algorithms, so he written an algorithm to find that position. But he is extremely poor at finding time complexity. Your task is to help your friend Mothi to analyze the time complexity of the given problem.

```
Int BinarySearch (int a, int low, int high, int tar)
{
  int mid;
  if (low > high) return 0;
  mid = floor((low + high)/2)
  if (a[mid] == tar)
    return mid;
  else if (tar < a[mid])
    return BinarySearch (a, low, mid-1, tar)
  else
    return BinarySearch (a, mid+1, high, tar)
}
```

Source code:

- **Algorithm:** Binary Search to find the position of a pole.
- **Input:** Sorted array `a[]`, indices `low`, `high`, target height `tar`.
- **Steps:**
 1. Find the midpoint `mid`.
 2. If `a[mid] == tar`, return `mid`.
 3. If `tar < a[mid]`, search left half.
 4. If `tar > a[mid]`, search right half.
- **Time Complexity:**
 - Each recursive call reduces the search space by half.
 - Maximum depth of recursion = $\log_2(n)$.
 - Overall time complexity: $O(\log n)$.

Comments of the Evaluators (if Any)Evaluator's Observation

Marks Secured: _____ out of [50].

Signature of the Evaluator

Date of Evaluation:

Date of the Session: ____/____/____

Time of the Session: ____ to ____

EX – 2 Implementation of String Matching Algorithms**Prerequisites:**

- Basics of Data Structures and C Programming.
- Basic knowledge about String Data type.

Pre-Lab :

- 1) 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[] using naïve string algorithm?(assume that n > m)

Input

txt[] = "THIS IS A TEST TEXT"

pat[] = "TEST"

Output

Pattern found at index 10

Input

txt[] = "AABAACAADAABAABA"

pat[] = "AABA"

Output

Pattern found at index 0

Pattern found at index 9

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

```
typedef struct {
    char character;
    int frequency;
} CharFreq;
```

```
int compare(const void *a, const void *b) {
    CharFreq *cf1 = (CharFreq *)a;
    CharFreq *cf2 = (CharFreq *)b;
    if (cf1->frequency != cf2->frequency)
        return cf1->frequency - cf2->frequency;
    return cf1->character - cf2->character;
}
```

```
void sortString(char str[]) {
    int freq[256] = {0};
```

```
int n = strlen(str);
for (int i = 0; i < n; i++) {
    freq[(int)str[i]]++;
}
CharFreq charFreqs[256];
int count = 0;
for (int i = 0; i < 256; i++) {
    if (freq[i] > 0) {
        charFreqs[count].character = (char)i;
        charFreqs[count].frequency = freq[i];
        count++;
    }
}
qsort(charFreqs, count, sizeof(CharFreq), compare);
int index = 0;
for (int i = 0; i < count; i++) {
    for (int j = 0; j < charFreqs[i].frequency; j++) {
        str[index++] = charFreqs[i].character;
    }
}
str[index] = '\0';
}

int main() {
    char str1[] = "aaabbc";
    char str2[] = "cbbaaa";
    sortString(str1);
    sortString(str2);
    printf("Sorted strings:\n%s\n%s\n", str1, str2);

    char str3[] = "aabbccdd";
    char str4[] = "aabcc";
    sortString(str3);
    sortString(str4);
    printf("Sorted strings:\n%s\n%s\n", str3, str4);

    return 0;
}
```

- 2) Discuss the Rabin Karp algorithm for string matching and explain time complexity of the algorithm?

The **Rabin-Karp algorithm** is used for string matching. It computes a hash value for the pattern and compares it with hash values of substrings in the text. If the hashes match, it checks the actual strings to avoid collisions.

Time Complexity:

- **Best case:** $O(n + m)$ — No hash collisions, just direct comparisons.
- **Worst case:** $O(n * m)$ — Many hash collisions, requiring string comparisons.
- **Average case:** $O(n + m)$ — With a good hash function, collisions are minimized.

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

Input

aaabbc
cbbaaa

Output

aabbcc
aabbcc

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

Input:

aabbccdd
aabcc

Output:

aabbccdd
baacc

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

```
typedef struct {
    char character;
    int frequency;
} CharFreq;
```

```
int compare(const void *a, const void *b) {
    CharFreq *cf1 = (CharFreq *)a;
    CharFreq *cf2 = (CharFreq *)b;
    if (cf1->frequency != cf2->frequency)
        return cf1->frequency - cf2->frequency;
    return cf1->character - cf2->character;
}
```

```
void sortString(char str[]) {
    int freq[256] = {0};
    int n = strlen(str);
    for (int i = 0; i < n; i++) {
        freq[(int)str[i]]++;
    }
    CharFreq charFreqs[256];
    int count = 0;
```

```
for (int i = 0; i < 256; i++) {
    if (freq[i] > 0) {
        charFreqs[count].character = (char)i;
        charFreqs[count].frequency = freq[i];
        count++;
    }
}
qsort(charFreqs, count, sizeof(CharFreq), compare);
int index = 0;
for (int i = 0; i < count; i++) {
    for (int j = 0; j < charFreqs[i].frequency; j++) {
        str[index++] = charFreqs[i].character;
    }
}
str[index] = '\0';
}

int main() {
    char str1[] = "aaabbc";
    char str2[] = "cbbaaa";
    sortString(str1);
    sortString(str2);
    printf("Sorted strings:\n%s\n%s\n", str1, str2);

    char str3[] = "aabbccdd";
    char str4[] = "aabcc";
    sortString(str3);
    sortString(str4);
    printf("Sorted strings:\n%s\n%s\n", str3, str4);

    return 0;
}
```


In-Lab:

- 1) Naive method and KMP are two string comparison methods. Write a program for Naïve method and KMP to check whether a pattern is present in a string or not. Using clock function find execution time for both and compare the time complexities of both the programs (for larger inputs) and discuss which one is more efficient and why?

Sample program with function which calculate execution time:

```
#include<stdio.h>
#include<time.h>
void fun()
{
    //some statements here
}
int main()
{
    //calculate time taken by fun()
    clock_t t;
    t=clock();
    fun();
    t=clock()-t;
    double time_taken=((double)t)/CLOCKS_PER_SEC; //in seconds
    printf("fun() took %f seconds to execute \n",time_taken);
    return 0;
}
```

Source code:

```
#include <stdio.h>
#include <string.h>
#include <time.h>

int naiveSearch(char *text, char *pattern) {
    int n = strlen(text);
    int m = strlen(pattern);

    for (int i = 0; i <= n - m; i++) {
        int j = 0;
        while (j < m && text[i + j] == pattern[j]) {
            j++;
        }
        if (j == m) {
            return i;
        }
    }
    return -1;
}

void computeLPSArray(char *pattern, int m, int *lps) {
    int len = 0;
```

```

int i = 1;
lps[0] = 0;

while (i < m) {
    if (pattern[i] == pattern[len]) {
        len++;
        lps[i] = len;
        i++;
    } else {
        if (len != 0) {
            len = lps[len - 1];
        } else {
            lps[i] = 0;
            i++;
        }
    }
}
}

```

```

int KMPSearch(char *text, char *pattern) {
    int n = strlen(text);
    int m = strlen(pattern);
    int lps[m];

    computeLPSArray(pattern, m, lps);

    int i = 0;
    int j = 0;

    while (i < n) {
        if (pattern[j] == text[i]) {
            i++;
            j++;
        }

        if (j == m) {
            return i - j;
        } else if (i < n && pattern[j] != text[i]) {
            if (j != 0) {
                j = lps[j - 1];
            } else {
                i++;
            }
        }
    }
    return -1;
}

```

```

void calculateExecutionTime(void (*func)(), char *text, char *pattern) {
    clock_t start, end;
    start = clock();

    func(text, pattern);

    end = clock();

```

```
double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
printf("Execution time: %f seconds\n", time_taken);
}

int main() {
    char text[] = "ABABDABACDABABCABAB";
    char pattern[] = "ABABCABAB";

    printf("Naive Search:\n");
    calculateExecutionTime(naiveSearch, text, pattern);

    printf("KMP Search:\n");
    calculateExecutionTime(KMPSearch, text, pattern);

    return 0;
}
```

OUTPUT

Naive Search:
Execution time: 0.000002 seconds
KMP Search:
Execution time: 0.000001 seconds

time complexities

Naive String Matching:

- Time Complexity: $O(n * m)$
 - n = length of text
 - m = length of pattern
 - Worst Case: $O(n * m)$ (needs to check each substring of length m in the text)

KMP String Matching:

- Time Complexity: $O(n + m)$
 - n = length of text
 - m = length of pattern
 - Worst Case: $O(n + m)$ (preprocessing the pattern in $O(m)$ and matching in $O(n)$)

discuss which one is more efficient and why

KMP is more efficient than the Naive method because:

- Naive has a time complexity of $O(n * m)$, requiring redundant comparisons for each possible substring.
- KMP preprocesses the pattern in $O(m)$ time to create an LPS array and then matches the text in $O(n)$ time, resulting in a total complexity of $O(n + m)$.

- 2) Lisa is a school student teacher gave her an assignment to check whether the pattern is there or not in a given text and also she mentioned that it is have solve by using kmp algorithm so when a mismatch come other some matches in your search if she print the number of letters that we can neglect before then she will get good marks so help her by writing a code.

Sample Input:

ABABDABACDABABCABAB

ABABCABAB

Sample Output:

we don't match before 2 letters because they will match anyway

we don't match before 0 letters because they will match anyway

we don't match before 1 letter because they will match anyway

we don't match before 0 letters because they will match anyway

Found pattern at index 10

Source code:

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

```
#include <string.h>
```

```
void computeLPSArray(char* pattern, int M, int* lps) {
```

```
    int length = 0;
```

```
    lps[0] = 0;
```

```
    int i = 1;
```

```
    while (i < M) {
```

```
        if (pattern[i] == pattern[length]) {
```

```
            length++;
```

```
            lps[i] = length;
```

```
            i++;
```

```
        } else {
```

```
            if (length != 0) {
```

```
                length = lps[length - 1];
```

```
            } else {
```

```
                lps[i] = 0;
```

```
                i++;
```

```
    }  
    }  
}  
}
```

```
void KMPSearch(char* text, char* pattern) {  
    int N = strlen(text);  
    int M = strlen(pattern);  
    int lps[M];  
    computeLPSArray(pattern, M, lps);  
    int i = 0;  
    int j = 0;  
  
    while (i < N) {  
        if (pattern[j] == text[i]) {  
            i++;  
            j++;  
        }  
  
        if (j == M) {  
            printf("Found pattern at index %d\n", i - j);  
            j = lps[j - 1];  
        } else if (i < N && pattern[j] != text[i]) {  
            if (j != 0) {  
                printf("we don't match before %d letters because they will match anyway\n", j);  
                j = lps[j - 1];  
            } else {  
                printf("we don't match before %d letters because they will match anyway\n", j);  
            }  
        }  
    }  
}
```

```
        i++;  
    }  
}  
}
```

```
int main() {  
    char text[] = "ABABDABACDABABCABAB";  
    char pattern[] = "ABABCABAB";  
    KMPSearch(text, pattern);  
    return 0;  
}
```

Post-Lab:

- 1) Given a pattern of length- 5 window, find the valid match in the given text by step-by-step process using Robin-Karp algorithm

Pattern: 2 1 9 3 6

Modulus: 21

Index: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

Text: 9 2 7 2 1 8 3 0 5 7 1 2 1 2 1 9 3 6 2 3 9 7

Source code:

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

```
int computeHash(int arr[], int start, int end, int modulus) {
    int hash = 0;
    for (int i = start; i <= end; i++) {
        hash = (hash * 10 + arr[i]) % modulus;
    }
    return hash;
}
```

```
void rabinKarp(int text[], int textLen, int pattern[], int patternLen, int modulus) {
    int patternHash = computeHash(pattern, 0, patternLen - 1, modulus);
    int currentHash = computeHash(text, 0, patternLen - 1, modulus);
```

```
    printf("Pattern Hash: %d\n", patternHash);
    printf("Step-by-step process:\n");
```

```
    for (int i = 0; i <= textLen - patternLen; i++) {
        printf("Index %d-%d, Text Window: ", i, i + patternLen - 1);
        for (int j = i; j < i + patternLen; j++) {
            printf("%d ", text[j]);
        }
        printf("\n");
```

```
    printf("Current Hash: %d\n", currentHash);
```

```
    if (currentHash == patternHash) {
        int match = 1;
        for (int j = 0; j < patternLen; j++) {
            if (text[i + j] != pattern[j]) {
                match = 0;
                break;
            }
        }
    }
}
```



```
        if (match) {
            printf("Valid match found at index %d\n", i);
        } else {
            printf("Hash matched, but window doesn't match pattern.\n");
        }
    }

    if (i < textLen - patternLen) {
        currentHash = (currentHash * 10 - text[i] * 10000 + text[i + patternLen]) %
modulus;
        if (currentHash < 0) {
            currentHash += modulus;
        }
    }
}

int main() {
    int text[] = {9, 2, 7, 2, 1, 8, 3, 0, 5, 7, 1, 2, 1, 2, 1, 9, 3, 6, 2, 3, 9, 7};
    int pattern[] = {2, 1, 9, 3, 6};
    int textLen = sizeof(text) / sizeof(text[0]);
    int patternLen = sizeof(pattern) / sizeof(pattern[0]);
    int modulus = 21;

    rabinKarp(text, textLen, pattern, patternLen, modulus);

    return 0;
}
```

- 2) 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.

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'

Source code:

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

char* transform_message(const char* s) {
    int len = strlen(s);
    char* result = (char*)malloc(len + 1);
    int result_index = 0;

    for (int i = 0; i < len; i++) {
        if (isdigit(s[i])) {
            int shift_value = s[i] - '0';
            char previous_char = result[result_index - 1];
            char shifted_char = previous_char + shift_value;
            result[result_index++] = shifted_char;
        } else {
            result[result_index++] = s[i];
        }
    }
    result[result_index] = '\0';
    return result;
}

int main() {
    const char* input_str = "a1b2c3d4e";
    char* output_str = transform_message(input_str);
    printf("Output: %s\n", output_str);
    free(output_str);
}
```

```
return 0;  
}
```

<u>Comments of the Evaluators (if Any)</u>	<u>Evaluator's Observation</u>
	Marks Secured: _____ out of [50]. Signature of the Evaluator Date of Evaluation:

Date of the Session: ____/____/____

Time of the Session: ____ to ____

EX – 3 Implementing Programs on Sorting**Prerequisites:**

- Basics of Data Structures and C Programming.
- Basic knowledge about Sorting Techniques.

Pre-Lab:

- 1) Write a Divide and Conquer algorithm to find the minimum distance between the pair of the points given in an array. Find the time complexity of the algorithm.

P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}}

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

```
#include <math.h>
```

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

```
typedef struct {
```

```
    int x, y;
```

```
} Point;
```

```
double dist(Point p1, Point p2) {
```

```
    return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
```

```
}
```

```
double min(double a, double b) {
```

```
    return (a < b) ? a : b;
```

```
}
```

```
int compareX(const void *a, const void *b) {  
    return ((Point*)a)->x - ((Point*)b)->x;  
}
```

```
int compareY(const void *a, const void *b) {  
    return ((Point*)a)->y - ((Point*)b)->y;  
}
```

```
double stripClosest(Point strip[], int size, double d) {  
    double min_dist = d;  
    qsort(strip, size, sizeof(Point), compareY);  
    for (int i = 0; i < size; i++) {  
        for (int j = i + 1; j < size && (strip[j].y - strip[i].y) < min_dist; j++) {  
            min_dist = min(min_dist, dist(strip[i], strip[j]));  
        }  
    }  
    return min_dist;  
}
```

```
double closestUtil(Point Px[], Point Py[], int n) {  
    if (n <= 3) {  
        double min_dist = INFINITY;  
        for (int i = 0; i < n; i++) {  
            for (int j = i + 1; j < n; j++) {  
                min_dist = min(min_dist, dist(Px[i], Px[j]));  
            }  
        }  
    }
```

```
    return min_dist;
}

int mid = n / 2;
Point midPoint = Px[mid];

Point Qx[mid], Rx[n - mid];
Point Qy[n], Ry[n];

int li = 0, ri = 0, liq = 0, riq = 0;
for (int i = 0; i < n; i++) {
    if (Px[i].x <= midPoint.x) {
        Qx[li++] = Px[i];
    } else {
        Rx[ri++] = Px[i];
    }

    if (Px[i].x <= midPoint.x) {
        Qy[liq++] = Py[i];
    } else {
        Ry[riq++] = Py[i];
    }
}

double dl = closestUtil(Qx, Qy, li);
double dr = closestUtil(Rx, Ry, ri);
double d = min(dl, dr);
```

```
Point strip[n];

int j = 0;

for (int i = 0; i < n; i++) {
    if (abs(Py[i].x - midPoint.x) < d) {
        strip[j++] = Py[i];
    }
}

return min(d, stripClosest(strip, j, d));
}

double closestPair(Point P[], int n) {
    Point Px[n], Py[n];

    for (int i = 0; i < n; i++) {
        Px[i] = P[i];
        Py[i] = P[i];
    }

    qsort(Px, n, sizeof(Point), compareX);
    qsort(Py, n, sizeof(Point), compareY);

    return closestUtil(Px, Py, n);
}
```

```
int main() {
```

```
Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};  
  
int n = sizeof(P) / sizeof(P[0]);  
  
printf("The minimum distance is: %lf\n", closestPair(P, n));  
  
return 0;  
}
```

Algorithm

Algorithm:

1. Sort the points by x and y coordinates.
2. Divide the points into two halves based on x-coordinates.
3. Recursively find the minimum distance in both halves.
4. Merge step: Check if there's a closer pair across the dividing line.

Time Complexity

Time Complexity:

- Sorting: $O(n \log n)$
- Recursion: $O(n \log n)$
- Final complexity: $O(n \log n)$

- 2) Write a divide and conquer algorithm for finding the maximum and minimum in the sequence of numbers. Find the time complexity.

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

```
void find_max_min(int arr[], int low, int high, int *max, int *min) {  
    if (low == high) {  
        *max = arr[low];  
        *min = arr[low];  
    } else if (high == low + 1) {  
        if (arr[low] > arr[high]) {  
            *max = arr[low];  
            *min = arr[high];  
        } else {  
            *max = arr[high];  
            *min = arr[low];  
        }  
    } else {  
        int mid = (low + high) / 2;  
        int max1, min1, max2, min2;  
  
        find_max_min(arr, low, mid, &max1, &min1);  
        find_max_min(arr, mid + 1, high, &max2, &min2);  
  
        *max = (max1 > max2) ? max1 : max2;  
        *min = (min1 < min2) ? min1 : min2;  
    }  
}
```

```
int main() {  
    int arr[] = {12, 5, 7, 9, 15, 3, 11, 10};  
    int n = sizeof(arr) / sizeof(arr[0]);  
    int max, min;  
  
    find_max_min(arr, 0, n - 1, &max, &min);  
  
    printf("Maximum: %d\n", max);  
    printf("Minimum: %d\n", min);  
}
```

```
return 0;  
}
```

Algorithm

Algorithm:

1. If the sequence has one element, return it as both the maximum and minimum.
2. If two elements, return the larger as the maximum and the smaller as the minimum.
3. Otherwise, divide the sequence, find the max and min recursively in both halves, and then compare the results.

Time Complexity

Time Complexity:

$O(n)$, where n is the number of elements in the sequence.

- 3) Trace out the output of the following using Merge sort. 10, 49, 32, 67, 45, 4, 7, 2, 1, 51, 78, 34, 89, 87, 36, 29, 3, 9, 11.

Merge Sort Steps:

1. Split the array into two halves until each subarray has only one element.
2. Merge the subarrays by comparing elements and combining them back into sorted arrays.

Initial Array:

[10, 49, 32, 67, 45, 4, 7, 2, 1, 51, 78, 34, 89, 87, 36, 29, 3, 9, 11]

Step 1: Splitting

- First split:
[10, 49, 32, 67, 45, 4, 7, 2, 1] and [51, 78, 34, 89, 87, 36, 29, 3, 9, 11]
- Continue splitting recursively:
 - Left side: [10, 49, 32, 67, 45, 4, 7, 2, 1]
 - [10, 49, 32] and [67, 45, 4]
 - [10, 49] and [32], then merge to [10, 32, 49]
 - [67, 45] and [4], then merge to [4, 45, 67]
 - Merge to: [4, 10, 32, 45, 49, 67]
 - Right side: [51, 78, 34, 89, 87, 36, 29, 3, 9, 11]
 - [51, 78, 34] and [89, 87, 36]
 - [51, 78] and [34], then merge to [34, 51, 78]
 - [89, 87] and [36], then merge to [36, 87, 89]
 - Merge to: [34, 36, 51, 78, 87, 89]
 - Now merge these two parts: [4, 10, 32, 45, 49, 67] and [34, 36, 51, 78, 87, 89]



Step 2: Merging

- Merge the left and right subarrays:
 1. Compare 4 and 34 : 4 is smaller, so add 4 to the result.
 2. Compare 10 and 34 : 10 is smaller, so add 10 .
 3. Compare 32 and 34 : 32 is smaller, so add 32 .
 4. Compare 45 and 34 : 34 is smaller, so add 34 .
 5. Compare 45 and 36 : 36 is smaller, so add 36 .
 6. Compare 45 and 51 : 45 is smaller, so add 45 .
 7. Compare 49 and 51 : 49 is smaller, so add 49 .
 8. Compare 67 and 51 : 51 is smaller, so add 51 .
 9. Compare 67 and 78 : 67 is smaller, so add 67 .
 10. Compare 78 and 89 : 78 is smaller, so add 78 .
 11. Compare 87 and 89 : 87 is smaller, so add 87 .
 12. Finally, add the remaining element 89 .

Final Sorted Array:

[1, 2, 3, 4, 7, 9, 10, 11, 29, 32, 34, 36, 45, 49, 51, 67, 78, 87, 89]

So, the output after applying Merge Sort to the given array is: [1, 2, 3, 4, 7, 9, 10, 11, 29, 32, 34, 36, 45, 49, 51, 67, 78, 87, 89]

In-Lab:

- 1) Harry's Aunt and family treat him badly and make him work all the time. Dudley, his cousin got homework from school and he as usual handed it over to Harry but Harry has a lot of work and his own homework to do.

The homework is to solve the problems which are numbered in numerical he tries to solve random question after solving random questions he did not put those questions in order Dudley will return in a time of $n \cdot \log n$ Harry has to arrange them as soon as possible. Help Harry to solve this problem so that he can go on and do his own homework.

Example**Input**

9

15,5,24,8,1,3,16,10,20

Output

1, 3, 5, 8, 10, 15, 16, 20, 24

Source code:

```
#include <stdio.h>

void quicksort(int arr[], int low, int high) {
    if (low < high) {
        int pivot = arr[high];
        int i = (low - 1);

        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        quicksort(arr, low, i);
        quicksort(arr, i + 2, high);
    }
}

int main() {
    int arr[] = {15, 5, 24, 8, 1, 3, 16, 10, 20};
    int n = sizeof(arr) / sizeof(arr[0]);

    quicksort(arr, 0, n - 1);

    for (int i = 0; i < n; i++) {
        if (i != 0) {
            printf(", ");
        }
    }
}
```

```
    }  
    printf("%d", arr[i]);  
}  
printf("\n");  
return 0;  
}
```

- 2) A group of 9 friends are playing a game, rules of the game are as follows: Each member will be assigned with a number and the sequence goes like e.g.: 7,6,10,5,9,2,1,15,7. Now they will be sorted in ascending order in such a way that tallest one will be sorted first. Now your task is to find the order of indices based on initial position of the given sequence and print the order of indices at the end of the iteration.

Source code:

```
#include <stdio.h>

#include <stdlib.h>

typedef struct {
    int value, index;
} Element;

int compare(const void *a, const void *b) {
    return ((Element*)b)->value - ((Element*)a)->value;
}

int main() {
    int sequence[] = {7, 6, 10, 5, 9, 2, 1, 15, 7};
    int n = sizeof(sequence) / sizeof(sequence[0]);
    Element elements[n];

    for (int i = 0; i < n; i++) {
        elements[i].value = sequence[i];
        elements[i].index = i;
    }

    qsort(elements, n, sizeof(Element), compare);
```

```
for (int i = 0; i < n; i++) {  
    printf("%d ", elements[i].index);  
}  
  
return 0;  
}
```

OUTPUT

7 2 4 0 8 1 3 5 6

Post-Lab:

- 1) Suppose a merge sort algorithm, for input size 64, takes 30 secs in the worst case. What is the maximum input size that can be calculated in 6 minutes (approximately)?

Source code:

Step 1: Time Complexity Relationship

The time complexity for merge sort is $T(n) = k \cdot n \log n$, where k is a constant. We're given that for an input size of $n = 64$, the time is $T(64) = 30$ seconds. We can use this to find k .

Step 2: Solve for k

From the given data:

$$30 = k \cdot 64 \log(64)$$

Since $\log(64) = 6$ (because $64 = 2^6$), we have:

$$30 = k \cdot 64 \cdot 6$$

$$30 = k \cdot 384$$

$$k = \frac{30}{384} = 0.078125$$

Step 3: Use k to Find Maximum n for 360 seconds

Now that we have $k = 0.078125$, we want to find the maximum input size n that can be processed in 360 seconds. We use the formula:

$$360 = 0.078125 \cdot n \log n$$

This equation doesn't have a simple algebraic solution, so we increment n starting from $n = 64$ until the time exceeds 360 seconds.

Step 4: Iterative Calculation

By testing values of n , we find that the largest n that fits within 360 seconds is $n = 512$.

So, the maximum input size that can be processed in 6 minutes is approximately $n = 512$.

- 2) Chris and Scarlett were playing a block sorting game where Scarlett challenged Chris that he has to sort the blocks which arranged in random order. And Scarlett puts a restriction that he should not use reference of first, median and last blocks to sort, and after sorting one block with reference to other block, for next iteration he must choose another block as the reference not the same block (random pivot).

Now, Chris wants help from you to sort the blocks. He wanted to sort them in a least time. Help him with the least time complexity sorting algorithm.

Input format

First line of input contains the number of test cases.

Next t lines of input contain

The number of blocks provided by Scarlett.

The array of blocks.

Source code:

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

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

int partition(int arr[], int low, int high) {
    int pivotIndex = low + rand() % (high - low + 1);
    int pivot = arr[pivotIndex];
    swap(&arr[pivotIndex], &arr[high]);
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}

void randomizedQuickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivotIndex = partition(arr, low, high);
        randomizedQuickSort(arr, low, pivotIndex - 1);
        randomizedQuickSort(arr, pivotIndex + 1, high);
    }
}

int main() {
    srand(time(NULL));
    int t;
    scanf("%d", &t);
```

```

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

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

    randomizedQuickSort(arr, 0, n - 1);

    for (int j = 0; j < n; j++) {
        printf("%d ", arr[j]);
    }
    printf("\n");
}

return 0;
}

```

Time Complexity

- **Average Case: $O(n \log n)$** – occurs when the pivot divides the array into roughly equal sub-arrays.
- **Worst Case: $O(n^2)$** – occurs with very unbalanced partitions, but is rare with random pivots.
- **Best Case: $O(n \log n)$** – occurs when the pivot always divides the array evenly.

<u>Comments of the Evaluators (if Any)</u>	<u>Evaluator's Observation</u>
	Marks Secured: _____ out of [50]. Signature of the Evaluator Date of Evaluation:

Date of the Session: ____ / ____ / ____

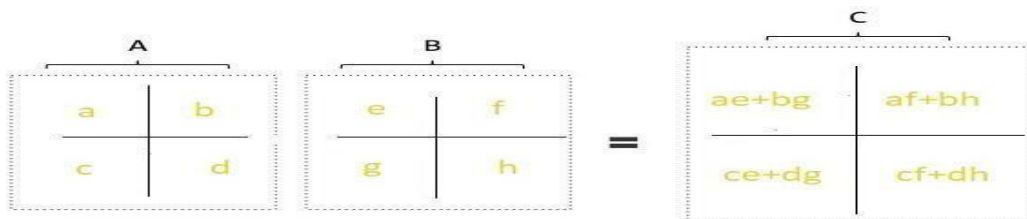
Time of the Session: ____ to ____

EX – 4 Application of Strassen's Matrix Multiplication and Convex Hull**Prerequisites:**

- Basics of Data Structures and C Programming.
- Basic knowledge about Matrix Multiplication.

Pre-Lab:

- 1) Trace the output of the following matrix multiplication using Strassen's Multiplication Method



A, B and C are the Matrices of Size $N \times N$

a, b, c and d are the sub-Matrices of A of size $\frac{N}{2} \times \frac{N}{2}$

e, f, g and h are the sub-Matrices of B of size $\frac{N}{2} \times \frac{N}{2}$

Step 1: Seven Products

Given the two matrices A and B :

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

We calculate the seven products as follows:

1. M_1 :

$$M_1 = (a + d)(e + h) = ae + ah + de + dh$$

2. M_2 :

$$M_2 = (c + d)e = ce + de$$

3. M_3 :

$$M_3 = a(f - h) = af - ah$$

4. M_4 :

$$M_4 = d(g - e) = dg - de$$

5. M_5 :

$$M_5 = (a + b)h = ah + bh$$

6. M_6 :

$$M_6 = (c - a)(e + f) = ce + cf - ae - af$$

7. M_7 :

$$M_7 = (b - d)(g + h) = bg + bh - dg - dh$$

Step 2: Calculate the Resulting Matrix C

Matrix C is:

$$C = \begin{bmatrix} p & q \\ r & s \end{bmatrix}$$

Now we use the seven products to compute each element of matrix C :

- p :

$$p = M_1 + M_4 - M_5 + M_7$$

Substituting the expressions for M_1 , M_4 , M_5 , and M_7 :

$$p = (ae + ah + de + dh) + (dg - de) - (ah + bh) + (bg + bh)$$

Simplifying:

$$p = ae + ah + de + dh + dg - de - ah - bh + bg + bh$$

After canceling out terms:

$$p = ae + dg + bg$$

- q :

$$q = M_3 + M_5$$

Substituting for M_3 and M_5 :

$$q = (af - ah) + (ah + bh) = af + bh$$

- r :

$$r = M_2 + M_4$$

Substituting for M_2 and M_4 :

$$r = (ce + de) + (dg - de) = ce + dg$$

- s :

$$s = M_1 - M_2 + M_3 + M_6$$

Substituting for M_1 , M_2 , M_3 , and M_6 :

$$s = (ae + ah + de + dh) - (ce + de) + (af - ah) + (ce + cf - ae - af)$$

Simplifying:

$$s = ae + ah + de + dh - ce - de + af - ah + ce + cf - ae - af$$

After canceling out terms:

$$s = dh + cf$$

Final Result

So the resulting matrix C is:

$$C = \begin{bmatrix} p & q \\ r & s \end{bmatrix} = \begin{bmatrix} ae + dg + bg & af + bh \\ ce + dg & dh + cf \end{bmatrix}$$

- 2) China had recently banned cryptocurrency. Due to this cryptocurrency cost has fallen drastically. Mr. Lee has lost a lot so he could not afford a stock advisor. He came to you (his friend) for help. He has the prices of stock on i'th day. Help him to find the maximum profit he can achieve. You may complete as many transactions as you like.

Input

7

[1,2,3,4,5,6,7]

Output

6

Explanation:

Buy the stock on 1st day at cost 1Rupee and sell the stock on 7th day at cost 7 Rupee and get profit of 6 rupees.

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

```
int maxProfit(int prices[], int n) {  
    int profit = 0;  
  
    for (int i = 1; i < n; i++) {  
        if (prices[i] > prices[i - 1]) {  
            profit += prices[i] - prices[i - 1];  
        }  
    }  
  
    return profit;  
}  
  
int main() {  
    int prices[] = {1, 2, 3, 4, 5, 6, 7};  
    int n = sizeof(prices) / sizeof(prices[0]);  
  
    int profit = maxProfit(prices, n);  
    printf("Maximum Profit: %d\n", profit);  
  
    return 0;  
}
```


In-Lab:

- 1) You are given 2 matrices of any size $N \times N$. Write a program to find the product of the two matrixes (use Strassen's Matrix Multiplication).

Source code:

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

void add(int n, int A[n][n], int B[n][n], int result[n][n]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result[i][j] = A[i][j] + B[i][j];
        }
    }
}

void subtract(int n, int A[n][n], int B[n][n], int result[n][n]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result[i][j] = A[i][j] - B[i][j];
        }
    }
}

void strassen(int n, int A[n][n], int B[n][n], int result[n][n]) {
    if (n == 1) {
        result[0][0] = A[0][0] * B[0][0];
        return;
    }

    int mid = n / 2;

    int A11[mid][mid], A12[mid][mid], A21[mid][mid], A22[mid][mid];
    int B11[mid][mid], B12[mid][mid], B21[mid][mid], B22[mid][mid];
    int C11[mid][mid], C12[mid][mid], C21[mid][mid], C22[mid][mid];
    int M1[mid][mid], M2[mid][mid], M3[mid][mid], M4[mid][mid], M5[mid][mid],
    M6[mid][mid], M7[mid][mid];
    int temp1[mid][mid], temp2[mid][mid];

    for (int i = 0; i < mid; i++) {
        for (int j = 0; j < mid; j++) {
```

```
A11[i][j] = A[i][j];
A12[i][j] = A[i][j + mid];
A21[i][j] = A[i + mid][j];
A22[i][j] = A[i + mid][j + mid];

B11[i][j] = B[i][j];
B12[i][j] = B[i][j + mid];
B21[i][j] = B[i + mid][j];
B22[i][j] = B[i + mid][j + mid];
    }
}

add(mid, A11, A22, temp1);
add(mid, B11, B22, temp2);
strassen(mid, temp1, temp2, M1);

add(mid, A21, A22, temp1);
strassen(mid, temp1, B11, M2);

subtract(mid, B12, B22, temp2);
strassen(mid, A11, temp2, M3);

subtract(mid, B21, B11, temp2);
strassen(mid, A22, temp2, M4);

add(mid, A11, A12, temp1);
strassen(mid, temp1, B22, M5);

subtract(mid, A21, A11, temp1);
add(mid, B11, B12, temp2);
strassen(mid, temp1, temp2, M6);

subtract(mid, A12, A22, temp1);
add(mid, B21, B22, temp2);
strassen(mid, temp1, temp2, M7);

add(mid, M1, M4, temp1);
subtract(mid, M7, M5, temp2);
add(mid, temp1, temp2, C11);

add(mid, M3, M5, C12);

add(mid, M2, M4, C21);
```

```
add(mid, M1, M3, temp1);
subtract(mid, M6, M2, temp2);
add(mid, temp1, temp2, C22);

for (int i = 0; i < mid; i++) {
    for (int j = 0; j < mid; j++) {
        result[i][j] = C11[i][j];
        result[i][j + mid] = C12[i][j];
        result[i + mid][j] = C21[i][j];
        result[i + mid][j + mid] = C22[i][j];
    }
}

void printMatrix(int n, int matrix[n][n]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int n;
    printf("Enter the size of the matrix (N x N): ");
    scanf("%d", &n);

    int A[n][n], B[n][n], result[n][n];

    printf("Enter elements of matrix A:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &A[i][j]);
        }
    }

    printf("Enter elements of matrix B:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &B[i][j]);
        }
    }
```

```
}  
  
strassen(n, A, B, result);  
  
printf("Product of matrix A and B is:\n");  
printMatrix(n, result);  
  
return 0;  
}
```

- 2) Mr. Lee is working in a construction where they had to fencing around trees in a field. The owner has asked a rough estimate to do fencing in that field such all the trees lie inside that region. Consider yourself as a cost estimator who works under Mr. Lee. You are given location of all the trees, and you need to find the points that include this fencing. You need to output the trees that are included in the fencing.

Input:

points = [[1,1], [2,2], [2,0], [2,4], [3,3], [4,2]]

Output:

[[1,1], [2,0], [3,3], [2,4], [4,2]]

Source code:

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

typedef struct {
    int x;
    int y;
} Point;

int compare(const void *a, const void *b) {
    Point *p1 = (Point *)a;
    Point *p2 = (Point *)b;
    if (p1->x == p2->x) {
        return p1->y - p2->y;
    }
    return p1->x - p2->x;
}

int cross(Point o, Point a, Point b) {
    return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x);
}

void convexHull(Point points[], int n) {
    qsort(points, n, sizeof(Point), compare);

    Point *lower = (Point *)malloc(n * sizeof(Point));
    int lower_size = 0;
    for (int i = 0; i < n; i++) {
        while (lower_size >= 2 && cross(lower[lower_size - 2], lower[lower_size - 1],
points[i]) < 0) {
            lower_size--;
        }
    }
}
```

```

    lower[lower_size++] = points[i];
}

Point *upper = (Point *)malloc(n * sizeof(Point));
int upper_size = 0;
for (int i = n - 1; i >= 0; i--) {
    while (upper_size >= 2 && cross(upper[upper_size - 2], upper[upper_size - 1],
points[i]) < 0) {
        upper_size--;
    }
    upper[upper_size++] = points[i];
}

int total_size = lower_size + upper_size - 2;
Point *hull = (Point *)malloc(total_size * sizeof(Point));
int idx = 0;
for (int i = 0; i < lower_size - 1; i++) {
    hull[idx++] = lower[i];
}
for (int i = 0; i < upper_size - 1; i++) {
    hull[idx++] = upper[i];
}

printf("Convex Hull: \n");
for (int i = 0; i < total_size; i++) {
    printf("[%d, %d]\n", hull[i].x, hull[i].y);
}

free(lower);
free(upper);
free(hull);
}

int main() {
    Point points[] = {{1, 1}, {2, 2}, {2, 0}, {2, 4}, {3, 3}, {4, 2}};
    int n = sizeof(points) / sizeof(points[0]);

    convexHull(points, n);

    return 0;
}

```

- 3) In a school there was conducted a contest among two groups. As part of the contest each group must re-arrange the cards that had given to the members in ascending order. Consider yourself as a part of the team and find the best viable way to win that round.

Input

3

3

[[2,5,6], [4,8,7], [9,3,1]]

Output

[1,2,3,4,5,6,7,8,9]

Source code:

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

int compare(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}

int main() {
    int arr1[] = {2, 5, 6};
    int arr2[] = {4, 8, 7};
    int arr3[] = {9, 3, 1};

    int len1 = sizeof(arr1) / sizeof(arr1[0]);
    int len2 = sizeof(arr2) / sizeof(arr2[0]);
    int len3 = sizeof(arr3) / sizeof(arr3[0]);

    int total_len = len1 + len2 + len3;

    int merged[total_len];

    int index = 0;
    for (int i = 0; i < len1; i++) merged[index++] = arr1[i];
    for (int i = 0; i < len2; i++) merged[index++] = arr2[i];
    for (int i = 0; i < len3; i++) merged[index++] = arr3[i];

    qsort(merged, total_len, sizeof(int), compare);

    for (int i = 0; i < total_len; i++) {
        printf("%d ", merged[i]);
    }
    printf("\n");

    return 0;
}
```

Post-Lab:

- 1) Mr. Hari Kumar owns a fruit market. In the market there are many sellers who are selling many kinds of fruits. More than one fruits seller can sell same kind of fruit. Mr. Hari Kumar wants to arrange their information in the sorted order based on their names of the sellers and id of the fruits. You must arrange the same type of fruits in the same order as original order.

0-mangoes 1-apples

[Hint: Use counting sort algorithm]

Input

4

[[0, c], [1, b], [0, a], [1, d]]

Output

[[0, a], [0, c], [1, b], [1, d]]

Source code:

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

struct FruitSeller {
    int fruit_id;
    char seller_name[50];
};

void counting_sort(struct FruitSeller arr[], int n) {
    struct FruitSeller output[n];
    int count[26] = {0};

    for (int i = 0; i < n; i++) {
        count[arr[i].seller_name[0] - 'a']++;
    }

    for (int i = 1; i < 26; i++) {
        count[i] += count[i - 1];
    }

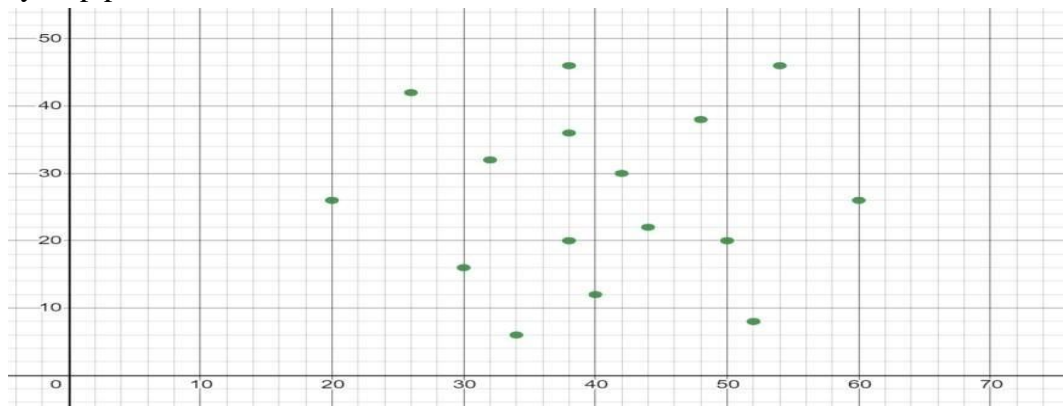
    for (int i = n - 1; i >= 0; i--) {
        int index = arr[i].seller_name[0] - 'a';
        output[count[index] - 1] = arr[i];
        count[index]--;
    }

    for (int i = 0; i < n; i++) {
        arr[i] = output[i];
    }
}
```



```
    }  
}  
  
void print_result(struct FruitSeller arr[], int n) {  
    for (int i = 0; i < n; i++) {  
        printf("[%d, %s] ", arr[i].fruit_id, arr[i].seller_name);  
    }  
    printf("\n");  
}  
  
int main() {  
    int n = 4;  
    struct FruitSeller sellers[] = {{0, "c"}, {1, "b"}, {0, "a"}, {1, "d"}};  
  
    counting_sort(sellers, n);  
  
    print_result(sellers, n);  
  
    return 0;  
}
```

- 2) Given a set of points in the plane, apply convex hull algorithm to the given points and explain it step by step process.



Source code:

The convex hull algorithm identifies the smallest convex boundary that can enclose a set of points. Let me explain the step-by-step process to apply the convex hull algorithm to the points in the given image.

Step-by-Step Process: Convex Hull Algorithm

1. Identify Extreme Points:

- Begin by finding the point with the lowest y-coordinate (if there's a tie, use the x-coordinate). This point serves as the starting reference.

2. Sort Points by Polar Angle:

- Sort all the points relative to the starting point by the angle they make with the x-axis. If two points form the same angle, the closer one is considered first.

3. Build the Hull Using Orientation Check:

- Traverse through the sorted list of points, maintaining a stack to track the vertices of the convex hull.
- For each point:
 - Check the orientation (left turn, right turn, or collinear) formed by the last two points in the stack and the current point.
 - If it's a right turn, pop the last point from the stack.
 - Push the current point onto the stack.

4. Close the Loop:

- Continue until all points have been processed, and the stack contains the convex hull vertices in order.

5. Output the Hull:

- The stack represents the convex hull in counterclockwise order.

<u>Comments of the Evaluators (if Any)</u>	<u>Evaluator's Observation</u>
	Marks Secured: _____ out of [50]. Signature of the Evaluator Date of Evaluation:

Date of the Session: ____/____/____

Time of the Session: ____ to ____

EX – 5 Working with Greedy Method**Prerequisites:**

- Basics of Data Structures and C Programming.
- Basic knowledge about Arrays.

Pre-Lab:

- 1) Explain why 0-1 Knapsack problems cannot be solved using greedy method unlike fractional knapsack.

- **0-1 Knapsack:** Greedy method doesn't work because you can't break items into fractions; choosing the highest value-to-weight ratio may not lead to the optimal solution.
- **Fractional Knapsack:** Works with the greedy method because you can take fractions of items, ensuring optimality.

- 2) Categorize the Following as single source or multiple source shortest path algorithms.

Floyd-Warshall algorithm –

Dijkstra's algorithm –

Bellman-Ford algorithm –

- **Floyd-Warshall:** Multiple-source
- **Dijkstra's:** Single-source
- **Bellman-Ford:** Single-source

3) List down various shortest path greedy algorithms.

- **Dijkstra's Algorithm**
- **Prim's Algorithm**
- **Kruskal's Algorithm**

In-Lab:

- 1) Given an array of size n that has the following specifications:
 - a. Each element in the array contains either a police officer or a thief.
 - b. Each police officer can catch only one thief.
 - c. A police officer cannot catch a thief who is more than K units away from the police officer.

We need to find the maximum number of thieves that can be caught.

Input

arr [] = {'P', 'T', 'T', 'P', 'T'},
k = 1.

Output

2

Here maximum 2 thieves can be caught; first police officer catches first thief and second police officer can catch either second or third thief.

Source code:

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

#define MAX_SIZE 100

int maxThievesCaught(char arr[], int n, int k) {
    int police[MAX_SIZE], thieves[MAX_SIZE];
    int police_count = 0, thieves_count = 0, caught = 0;

    for (int i = 0; i < n; i++) {
        if (arr[i] == 'P') {
            police[police_count++] = i;
        } else if (arr[i] == 'T') {
            thieves[thieves_count++] = i;
        }
    }

    int i = 0, j = 0;
    while (i < police_count && j < thieves_count) {
        if (abs(police[i] - thieves[j]) <= k) {
            caught++;
            i++;
            j++;
        } else if (police[i] < thieves[j]) {
            i++;
        } else {
            j++;
        }
    }
}
```

```
}

return caught;
}

int main() {
    char arr[] = {'P', 'T', 'T', 'P', 'T'};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 1;

    int result = maxThievesCaught(arr, n, k);
    printf("Maximum number of thieves caught: %d\n", result);

    return 0;
}
```

- 2) Given n non-negative integers a_1, a_2, \dots, a_n , where each represents a point at coordinate (i, a_i) . n vertical lines are drawn such that the two endpoints of the line i is at (i, a_i) and $(i, 0)$. Find two lines, which, together with the x-axis forms a container, such that the container contains the most water. Notice that you may not slant the container.

Input

height = [1,8,6,2,5,4,8,3,7]

Output

49

Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container

Source code:

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

```
int maxArea(int height[], int n) {
    int left = 0;
    int right = n - 1;
    int max_area = 0;

    while (left < right) {
        int width = right - left;
        int h = (height[left] < height[right]) ? height[left] : height[right];
        int area = h * width;

        if (area > max_area) {
            max_area = area;
        }

        if (height[left] < height[right]) {
            left++;
        } else {
            right--;
        }
    }

    return max_area;
}

int main() {
    int height[] = {1, 8, 6, 2, 5, 4, 8, 3, 7};
```



```
int n = sizeof(height) / sizeof(height[0]);  
  
int result = maxArea(height, n);  
printf("Maximum area: %d\n", result);  
  
return 0;  
}
```

Post-Lab:

- 1) Given an array of jobs where every job has a deadline and associated profit if the job is finished before the deadline. It is also given that every job takes a single unit of time, so the minimum possible deadline for any job is 1. How to maximize total profit if only one job can be scheduled at a time.

Input

4

Job ID Deadline Profit

a 4 20

b 1 10

c 1 40

d 1 30

Output

60

profit sequence of jobs is c, a

Source code:

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

typedef struct {
    char job_id;
    int deadline;
    int profit;
} Job;

int compare(const void* a, const void* b) {
    return ((Job*)b)->profit - ((Job*)a)->profit;
}

void jobScheduling(Job jobs[], int n) {
    qsort(jobs, n, sizeof(Job), compare);

    int max_deadline = 0;
    for (int i = 0; i < n; i++) {
        if (jobs[i].deadline > max_deadline) {
            max_deadline = jobs[i].deadline;
        }
    }

    int slots[max_deadline];
```

```
memset(slots, -1, sizeof(slots));

char job_sequence[max_deadline];
int total_profit = 0;

for (int i = 0; i < n; i++) {
    for (int j = jobs[i].deadline - 1; j >= 0; j--) {
        if (slots[j] == -1) {
            slots[j] = i;
            job_sequence[j] = jobs[i].job_id;
            total_profit += jobs[i].profit;
            break;
        }
    }
}

printf("Maximum Profit: %d\n", total_profit);
printf("Job Sequence: ");
for (int i = 0; i < max_deadline; i++) {
    if (slots[i] != -1) {
        printf("%c ", job_sequence[i]);
    }
}
printf("\n");

int main() {
    Job jobs[] = {
        {'a', 4, 20},
        {'b', 1, 10},
        {'c', 1, 40},
        {'d', 1, 30}
    };
    int n = sizeof(jobs) / sizeof(jobs[0]);
    jobScheduling(jobs, n);
    return 0;
}
```

- 2) There are N Mice and N holes are placed in a straight line. Each hole can accommodate only 1 mouse. A mouse can stay at his position, move one step right from x to $x + 1$, or move one step left from x to $x - 1$. Any of these moves consumes 1 minute. Assign mice to holes so that the time when the last mouse gets inside a hole is minimized.

Example: positions of mice are: 4 -4 2

Positions of holes are: 4 0 5

Input

A: list of positions of mice

B: list of positions of holes

Output

single integer value

Source code:

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

int compare(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}

int assignMiceToHoles(int mice[], int holes[], int n) {
    qsort(mice, n, sizeof(int), compare);
    qsort(holes, n, sizeof(int), compare);

    int max_time = 0;
    for (int i = 0; i < n; i++) {
        int distance = abs(mice[i] - holes[i]);
        if (distance > max_time) {
            max_time = distance;
        }
    }

    return max_time;
}

int main() {
    int mice[] = {4, -4, 2};
    int holes[] = {4, 0, 5};
    int n = sizeof(mice) / sizeof(mice[0]);

    int result = assignMiceToHoles(mice, holes, n);
    printf("Minimum time required: %d\n", result);
    return 0;
}
```

Comments of the Evaluators (if Any)Evaluator's Observation

Marks Secured: _____ out of [50].

Signature of the Evaluator
Date of Evaluation:

Date of the Session: ____ / ____ / ____

Time of the Session: ____ to ____

EX – 6 Working with Job Sequencing, Huffman Coding, Single Source Shortest Path and Minimum Cost Spanning Trees.

Prerequisites:

- Basics of Data Structures and C Programming.
- Basic knowledge about arrays, trees and graphs.

Pre-Lab:

- 1) Our TASC has the information that a scientist has gone crazy and was planning a gas leakage in Delhi. We do not know the exact location where he was planning to do this attack. When Srikanth Tiwari tried to catch him, The Scientist tried to kill himself and has gone into coma in this process. Now no one knows the location of the attack except he has kept all the information in a file in his private server, but it is encoded using Huffman coding. So, help NIA decode the information. Given the root of the graph and the encoded information write a function to decode it. The function will have input parameters of root node and encoded string.

Input

Encoded String-1001011

Output

ABACA

Source code:

```
#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

    char data;

    struct Node *left, *right;

} Node;
```

```
void decodeHuffman(Node *root, const char *encoded) {  
  
    Node *curr = root;  
  
    while (*encoded) {  
  
        curr = (*encoded == '0') ? curr->left : curr->right;  
  
        if (!curr->left && !curr->right) {  
  
            printf("%c", curr->data);  
  
            curr = root;  
  
        }  
  
        encoded++;  
  
    }  
  
    printf("\n");  
  
}
```

```
Node *newNode(char data) {  
  
    Node *node = (Node *)malloc(sizeof(Node));  
  
    node->data = data;  
  
    node->left = node->right = NULL;  
  
    return node;  
  
}
```

```
int main() {  
  
    Node *root = newNode('\0');  
  
    root->left = newNode('A');  
  
    root->right = newNode('\0');
```

```
root->right->left = newNode('B');  
  
root->right->right = newNode('C');  
  
char encoded[] = "1001011";  
  
decodeHuffman(root, encoded);  
  
return 0;  
  
}
```


In-Lab:

- 1) You are a Human resource manager working in a Startup. You are tasked with to utilize the best of the working professionals to get the maximum profit for different jobs of a Project. An array of jobs is given where every job has a deadline and associated profit if the job is finished before the deadline. It is also given that every job takes single unit of time, so the minimum possible deadline for any job is 1. How to maximize total profit if only one job can be scheduled at a time. Write a code for the following problem.

Input

```
7
a b c d e f g
3 4 4 2 3 1 2
35 30 25 20 15 12 5
```

Output

```
110
d c a b
```

Source code:

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

typedef struct {
    char id;
    int deadline;
    int profit;
} Job;

int compare(const void *a, const void *b) {
    return ((Job *)b)->profit - ((Job *)a)->profit;
}

void job_scheduling(char ids[], int deadlines[], int profits[], int n) {
    Job jobs[n];
    for (int i = 0; i < n; i++) {
        jobs[i].id = ids[i];
        jobs[i].deadline = deadlines[i];
        jobs[i].profit = profits[i];
    }

    qsort(jobs, n, sizeof(Job), compare);

    int max_deadline = 0;
    for (int i = 0; i < n; i++) {
        if (jobs[i].deadline > max_deadline) {
            max_deadline = jobs[i].deadline;
        }
    }
}
```

```

    }
}

char slots[max_deadline];
memset(slots, '-', sizeof(slots));
int total_profit = 0;

for (int i = 0; i < n; i++) {
    for (int j = jobs[i].deadline - 1; j >= 0; j--) {
        if (slots[j] == '-') {
            slots[j] = jobs[i].id;
            total_profit += jobs[i].profit;
            break;
        }
    }
}

printf("%d\n", total_profit);
for (int i = 0; i < max_deadline; i++) {
    if (slots[i] != '-') {
        printf("%c ", slots[i]);
    }
}
printf("\n");
}

int main() {
    char ids[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g'};
    int deadlines[] = {3, 4, 4, 2, 3, 1, 2};
    int profits[] = {35, 30, 25, 20, 15, 12, 5};
    int n = sizeof(ids) / sizeof(ids[0]);

    job_scheduling(ids, deadlines, profits, n);

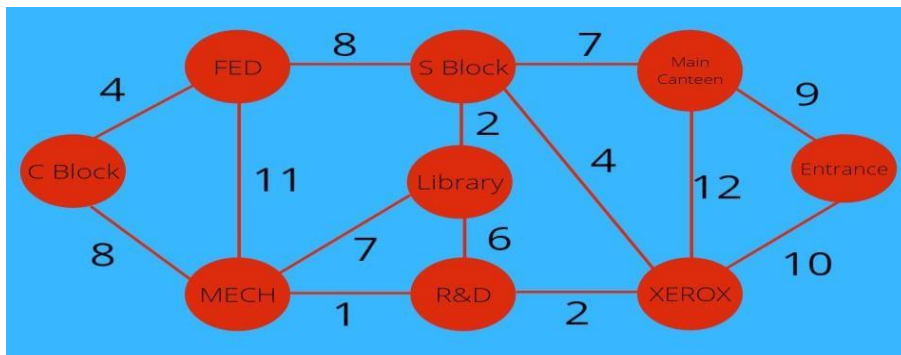
    return 0;
}

```

- 2) Surya is a student at KL University, He is currently standing in the C-Block. He has 8 friends who are situated at different Blocks (Places) inside the university and he wishes to talk to each of them in person. The distances are represented on the undirected graph which is provided below. He wishes to take the shortest distance for each place from his location. Help him in meeting every one of his friends by developing a program that can determine the shortest distance between the C-Block and every other place on the graph. Remember that the path is not required.

Hint: Use Dijkstra's algorithm to calculate the distances between the C-Block and every other place

Output for the same is provided along with the question.



Output:

Vertex	Distance from Source
C Block	0
FED Block	4
S Block	12
Main Canteen	19
Entrance	21
Xerox	11
R&D Block	9
Mech Block	8
Library	14

Source code:

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

#define V 9

int minDistance(int dist[], int sptSet[]) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++) {
        if (sptSet[v] == 0 && dist[v] <= min) {
            min = dist[v], min_index = v;
        }
    }
}
```

```

    return min_index;
}

void printSolution(int dist[]) {
    printf("Vertex\tDistance from Source\n");
    const char* vertices[V] = {"C Block", "FED Block", "S Block", "Main Canteen", "Entrance",
    "Xerox", "R&D Block", "Mech Block", "Library"};
    for (int i = 0; i < V; i++) {
        printf("%s\t%d\n", vertices[i], dist[i]);
    }
}

void dijkstra(int graph[V][V], int src) {
    int dist[V];
    int sptSet[V];

    for (int i = 0; i < V; i++) {
        dist[i] = INT_MAX;
        sptSet[i] = 0;
    }

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet);

        sptSet[u] = 1;

        for (int v = 0; v < V; v++) {
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

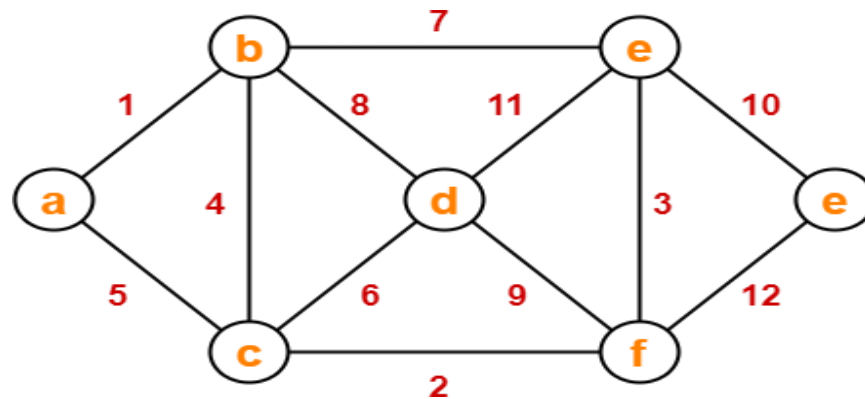
    printSolution(dist);
}

```

```
int main() {  
    int graph[V][V] = {  
        {0, 4, 0, 0, 0, 0, 0, 8, 0},  
        {4, 0, 8, 0, 0, 0, 0, 11, 0},  
        {0, 8, 0, 7, 0, 4, 0, 0, 2},  
        {0, 0, 7, 0, 9, 14, 0, 0, 0},  
        {0, 0, 0, 9, 0, 10, 0, 0, 0},  
        {0, 0, 4, 14, 10, 0, 2, 0, 0},  
        {0, 0, 0, 0, 0, 2, 0, 1, 6},  
        {8, 11, 0, 0, 0, 0, 1, 0, 7},  
        {0, 0, 2, 0, 0, 0, 6, 7, 0}  
    };  
  
    dijkstra(graph, 0);  
  
    return 0;  
}
```

Post-Lab:

- 1) Mr. Tripathi is a network cable operator. He now shifted to a new city where he needs to work on designing the cable lines to each street/colony. Given the graph and each node represents a street, help him to design an optimal cable line using prim's algorithm. Write a program to solve this.



Source code:

```

#include <stdio.h>
#include <limits.h>
#include <stdbool.h>

#define V 6

int minKey(int key[], bool mstSet[]) {
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++) {
        if (!mstSet[v] && key[v] < min) {
            min = key[v], min_index = v;
        }
    }
    return min_index;
}

void printMST(int parent[], int graph[V][V]) {
    printf("Minimum Spanning Tree:\n");
    for (int i = 1; i < V; i++) {
        printf("%c - %c : %d\n", 'a' + parent[i], 'a' + i, graph[i][parent[i]]);
    }
}

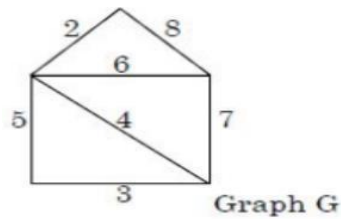
void primMST(int graph[V][V]) {
    int parent[V], key[V];
    bool mstSet[V];
    for (int i = 0; i < V; i++) {

```

```
    key[i] = INT_MAX, mstSet[i] = false;
}
key[0] = 0;
parent[0] = -1;
for (int count = 0; count < V - 1; count++) {
    int u = minKey(key, mstSet);
    mstSet[u] = true;
    for (int v = 0; v < V; v++) {
        if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {
            parent[v] = u, key[v] = graph[u][v];
        }
    }
}
printMST(parent, graph);
}

int main() {
    int graph[V][V] = {
        {0, 1, 5, 0, 0, 0},
        {1, 0, 4, 8, 7, 0},
        {5, 4, 0, 6, 0, 2},
        {0, 8, 6, 0, 11, 9},
        {0, 7, 0, 11, 0, 3},
        {0, 0, 2, 9, 3, 0}
    };
    primMST(graph);
    return 0;
}
```

- 2) Mr.Tripathi done with designing the cable lines but now there comes a new task, that is working on street lines. Help him again to find weight of the graph using Kruskal algorithm. Write a program to solve this.



Source code:

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

#define VERTICES 5
#define EDGES 7

typedef struct {
    int u, v, weight;
} Edge;

typedef struct {
    int parent[VERTICES];
    int rank[VERTICES];
} DisjointSet;

void initializeSet(DisjointSet *ds) {
    for (int i = 0; i < VERTICES; i++) {
        ds->parent[i] = i;
        ds->rank[i] = 0;
    }
}

int find(DisjointSet *ds, int u) {
    if (ds->parent[u] != u) {
        ds->parent[u] = find(ds, ds->parent[u]);
    }
    return ds->parent[u];
}

void unionSet(DisjointSet *ds, int u, int v) {
```



```

int rootU = find(ds, u);
int rootV = find(ds, v);

if (rootU != rootV) {
    if (ds->rank[rootU] > ds->rank[rootV]) {
        ds->parent[rootV] = rootU;
    } else if (ds->rank[rootU] < ds->rank[rootV]) {
        ds->parent[rootU] = rootV;
    } else {
        ds->parent[rootV] = rootU;
        ds->rank[rootU]++;
    }
}
}

int compareEdges(const void *a, const void *b) {
    return ((Edge*)a)->weight - ((Edge*)b)->weight;
}

int kruskalMST(Edge edges[]) {
    qsort(edges, EDGES, sizeof(Edge), compareEdges);

    DisjointSet ds;
    initializeSet(&ds);

    int mst_weight = 0, count = 0;

    for (int i = 0; i < EDGES; i++) {
        if (find(&ds, edges[i].u) != find(&ds, edges[i].v)) {
            unionSet(&ds, edges[i].u, edges[i].v);
            mst_weight += edges[i].weight;
            count++;
            if (count == VERTICES - 1) break;
        }
    }
    return mst_weight;
}

```

```
int main() {  
    Edge edges[EDGES] = {  
        {0, 1, 2}, {1, 2, 8}, {1, 3, 6}, {2, 3, 4},  
        {2, 4, 5}, {3, 4, 3}, {0, 2, 7}  
    };  
  
    printf("Weight of MST: %d\n", kruskalMST(edges));  
    return 0;  
}
```

<u>Comments of the Evaluators (if Any)</u>	<u>Evaluator's Observation</u>
	Marks Secured: _____ out of [50]. Signature of the Evaluator Date of Evaluation:

Date of the Session: ____/____/____

Time of the Session: ____to____

EX – 7 Working with Dynamic Programming**Prerequisites:**

- Basics of Data Structures and C Programming.
- Basic knowledge about Arrays and Binary Search Tree.

Pre-Lab:

- 1) A student named Satish is eagerly waiting to attend tomorrow's class. As he searched the concepts of tomorrow's lecture in the course handout and started solving the problems, in the middle he was stoked in the concept of strings because he was poor in this concept so help him so solve the problem, given two strings str1 and str2 and below operations that can be performed on str1. Find minimum number of edits (operations) required to convert 'str1' into 'str2'.

1. Insert
2. Remove
3. Replace

Input

str1 = "cat", str2 = "cut"

Output

1

We can convert str1 into str2 by replacing 'a' with 'u'.

Input

str1 = "sunday", str2 = "saturday"

Output

3

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

```
int min(int x, int y, int z) {
    if (x <= y && x <= z)
        return x;
    if (y <= x && y <= z)
        return y;
    return z;
}
```

```
int minEditDistance(char *str1, char *str2) {
    int m = strlen(str1);
    int n = strlen(str2);

    int dp[m + 1][n + 1];

    for (int i = 0; i <= m; i++)
        dp[i][0] = i;
```

```
for (int j = 0; j <= n; j++)
    dp[0][j] = j;

for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++) {
        if (str1[i - 1] == str2[j - 1]) {
            dp[i][j] = dp[i - 1][j - 1];
        } else {
            dp[i][j] = 1 + min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]);
        }
    }
}

return dp[m][n];
}

int main() {
    char str1[100], str2[100];
    printf("Enter the first string: ");
    scanf("%s", str1);
    printf("Enter the second string: ");
    scanf("%s", str2);
    printf("The minimum edit distance is: %d\n", minEditDistance(str1, str2));
    return 0;
}
```

- 2) Given N numbers n_1, n_2, \dots, n_N and Q queries q_1, q_2, \dots, q_Q . Your task is to print Q ($Q < j < \text{numbers}$) f_1, f_2, \dots, f_Q , corresponding to query q_j $\max(n_1=f_j, n_2, \dots, n_Q)$ using dynamic programming.

Input

5 3
5 4 8 6 9
2 3 5

Output

5 8 9

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

```
int main() {
    int N, Q;
    scanf("%d %d", &N, &Q);

    int nums[N], queries[Q], max_up_to[N];

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

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

    max_up_to[0] = nums[0];
    for (int i = 1; i < N; i++) {
        max_up_to[i] = (nums[i] > max_up_to[i - 1]) ? nums[i] : max_up_to[i - 1];
    }

    for (int i = 0; i < Q; i++) {
        int query = queries[i] - 1;
        printf("%d ", max_up_to[query]);
    }
    printf("\n");

    return 0;
}
```

In-Lab:

- 1) Bhanu is a student at KL University who likes playing with strings, after reading a question from their lab workbook for the DAA Course she found what is meant by a subsequence.

(A subsequence is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements)

So, she created 2 strings out of which one she considered as a master string and the other one as a slave string. She challenged her friend Teju to find out whether the slave string is a subsequence of the master string or not, As Teju is undergoing her CRT classes she decided to code the logic for this question. Help her in building the logic and write a code using Dynamic programming concept.

Source code:

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

bool is_subsequence(char *master, char *slave) {
    int m = strlen(master), n = strlen(slave);
    bool dp[m + 1][n + 1];

    for (int i = 0; i <= m; i++)
        dp[i][0] = true;
    for (int j = 1; j <= n; j++)
        dp[0][j] = false;

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (master[i - 1] == slave[j - 1])
                dp[i][j] = dp[i - 1][j - 1];
            else
                dp[i][j] = dp[i - 1][j];
        }
    }

    return dp[m][n];
}

int main() {
    char master[] = "abcdef";
    char slave[] = "ace";

    if (is_subsequence(master, slave))
```

```
    printf("True\n");  
else  
    printf("False\n");  
  
return 0;  
}
```

- 2) Raju has very less Marks in Dynamic programming test conducted by his teacher. So, the teacher had given a problem about optimal Binary Search Tree which should be solved using Dynamic Programming by the next class. Since, he is very weak in Dynamic programming you must help him to solve the problem. The problem can be described as follows:

An unsorted data of keys and their frequencies are given to you and some queries where each query contains two integers which describe the range of the indices (index range) for which you must print the root of that Optimal Binary Search Tree.

Input

```
4
12 10 20 21
8 34 50 1
2
0 3
0 1
```

Output

```
Cost of Optimal BST is 144
20
1
```

Source code:

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

int sum(int freq[], int i, int j) {
    int total = 0;
    for (int k = i; k <= j; k++) {
        total += freq[k];
    }
    return total;
}

void optimal_bst(int keys[], int freq[], int n) {
    int dp[n][n];
    int root[n][n];

    for (int i = 0; i < n; i++) {
        dp[i][i] = freq[i];
        root[i][i] = i;
    }

    for (int len = 2; len <= n; len++) {
        for (int i = 0; i <= n - len; i++) {
            int j = i + len - 1;
            dp[i][j] = INT_MAX;
```



```
int total_freq = sum(freq, i, j);

for (int r = i; r <= j; r++) {
    int cost = (r > i ? dp[i][r - 1] : 0) + (r < j ? dp[r + 1][j] : 0) + total_freq;

    if (cost < dp[i][j]) {
        dp[i][j] = cost;
        root[i][j] = r;
    }
}

printf("Cost of Optimal BST is %d %d\n", dp[0][n - 1], keys[root[0][n - 1]]);
printf("%d\n", root[0][n - 1]);
}

int main() {
    int keys[] = {12, 10, 20, 21};
    int freq[] = {8, 34, 50, 1};
    int n = sizeof(keys) / sizeof(keys[0]);

    optimal_bst(keys, freq, n);

    return 0;
}
```

Post-Lab:

- 1) Bhanu and Teju are playing dice game where there are N dice with M faces and the dice are numbered from 1 to M. A person wins the game if the sum of the faces of dice adds up to a value X. you are playing on Bhanu's team, and It is Teju's turn now.

You are supposed to find number of ways your opponent can win the game where N, M and X are provided as input. Use Dynamic programming to solve the problem.

Using DP (Dynamic programming) to find the number of ways to get sum X.

Input

M = 2

N = 2

X = 3

Output

2

Source code:

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

int countWays(int M, int N, int X) {
    int dp[N + 1][X + 1];
    memset(dp, 0, sizeof(dp));

    dp[0][0] = 1;

    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= X; j++) {
            dp[i][j] = 0;
            for (int k = 1; k <= M; k++) {
                if (j - k >= 0) {
                    dp[i][j] += dp[i - 1][j - k];
                }
            }
        }
    }

    return dp[N][X];
}

int main() {
    int M = 2;
    int N = 2;
    int X = 3;

    printf("%d\n", countWays(M, N, X));
    return 0;
}
```

Comments of the Evaluators (if Any)Evaluator's Observation

Marks Secured: _____ out of [50].

Signature of the Evaluator

Date of Evaluation:

Date of the Session: ____ / ____ / ____

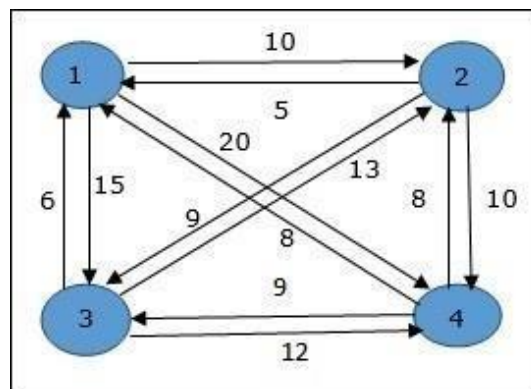
Time of the Session: ____ to ____

EX – 8 Working with TSP problem, Ford Fulkerson and Graph Traversal Techniques**Prerequisites:**

- Basics of Data Structures and C Programming.
- Basic knowledge about Graphs.

Pre-lab:

- 1) Your father brought you a ticket to world tour. You have a choice to go to three places, your father knows the places you wanted to travel so he made a graph with the measuring distances from home. Now you start from your place (1: Source) to other places as shown in the graph below apply TSP to find shortest path to visit all places and return to home. (Ex: 2: London, 3: Paris, 4: Singapore)



Step 1: Extract Given Distances

The graph is represented as follows:

- $1 \rightarrow 2 = 10$
- $1 \rightarrow 3 = 15$
- $1 \rightarrow 4 = 20$
- $2 \rightarrow 1 = 5$
- $2 \rightarrow 3 = 9$
- $2 \rightarrow 4 = 10$
- $3 \rightarrow 1 = 6$
- $3 \rightarrow 2 = 13$
- $3 \rightarrow 4 = 12$
- $4 \rightarrow 1 = 8$
- $4 \rightarrow 2 = 8$
- $4 \rightarrow 3 = 9$

Step 2: Evaluate All Possible Routes

We need to visit all cities (2, 3, 4) and return to 1 with minimal cost.

Possible Routes and Their Costs

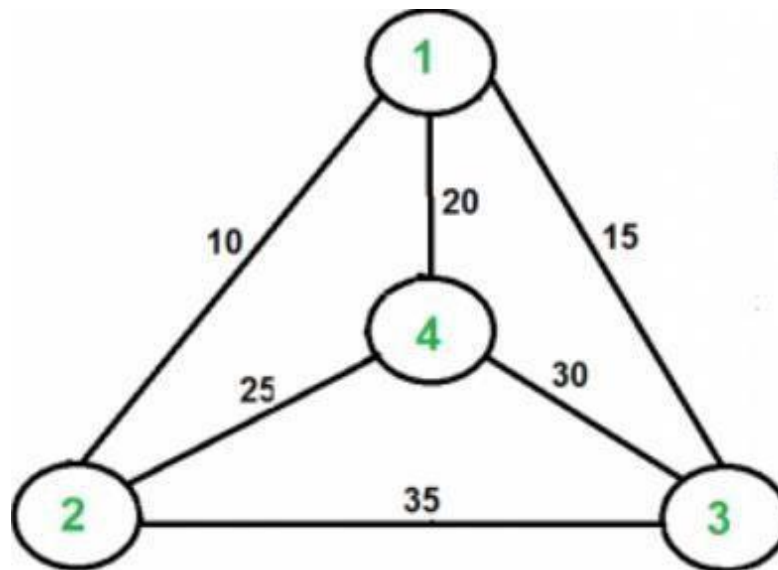
1. $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$
 - $10 + 9 + 12 + 8 = 39$
2. $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$
 - $10 + 10 + 9 + 6 = 35$ ✓
3. $1 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 1$
 - $15 + 13 + 10 + 8 = 46$
4. $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$
 - $15 + 12 + 8 + 5 = 40$
5. $1 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 1$
 - $20 + 8 + 9 + 6 = 43$
6. $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$
 - $20 + 9 + 13 + 5 = 47$

Step 3: Optimal Solution

The shortest path for TSP is:

$1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$ with a total cost of 35.

- 2) You are given a map(graph) with the distances between the places. Here you will consider the 1st node as the starting point and ending point, Since the route is cyclic you can take any node as starting point and ending point. Find the minimum cost route and remember the Hamiltonian cycle.



Graph Representation

Vertices: {1, 2, 3, 4}

Edges with weights:

- $(1,2) = 10$
- $(1,3) = 15$
- $(1,4) = 20$
- $(2,3) = 35$
- $(2,4) = 25$
- $(3,4) = 30$

Solving for Minimum Cost Hamiltonian Cycle

A **Hamiltonian cycle** visits all nodes exactly once and returns to the starting node. We compute the total weight for all possible cycles and choose the minimum.

Possible Hamiltonian Cycles:

1. $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$

$$\text{Cost} = 10 + 35 + 30 + 20 = 95$$

2. $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$

$$\text{Cost} = 10 + 25 + 30 + 15 = 80$$

3. $1 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 1$

$$\text{Cost} = 15 + 35 + 25 + 20 = 95$$

4. $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$

$$\text{Cost} = 15 + 30 + 25 + 10 = 80$$

5. $1 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 1$

$$\text{Cost} = 20 + 25 + 35 + 15 = 95$$

6. $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$

$$\text{Cost} = 20 + 30 + 35 + 10 = 95$$

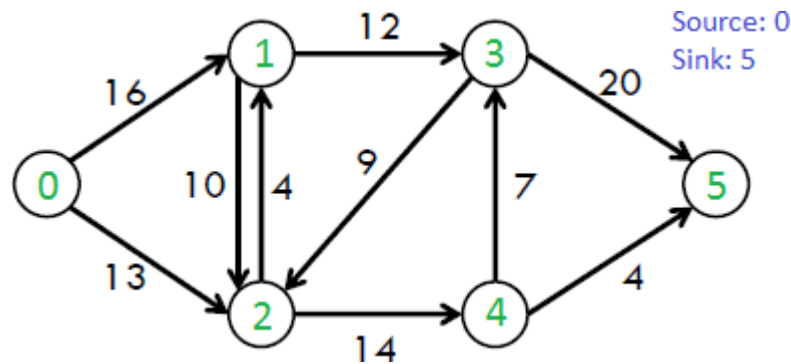
Minimum Cost Hamiltonian Cycle

The **optimal route** with minimum cost is:

- $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$ OR $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$
- Minimum cost = 80

In-Lab:

- 1) Emma has a graph which represents a flow network where every edge has a capacity. Also given two vertices source s and sink t in the graph, find the maximum possible flow from s to t with following constraints:
 - a. Flow on an edge does not exceed the given capacity of the edge.
 - b. Incoming flow is equal to outgoing flow for every vertex except s and t .



Find the Maximum flow using the graph and by implementing the code.

Source code:

```

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

struct Edge {
    int flow, capacity, u, v;
};

struct Vertex {
    int h, e_flow;
};

struct Graph {
    int V;
    struct Vertex *ver;
    struct Edge *edge;
    int edge_count;
};

struct Graph* createGraph(int V) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->V = V;
    graph->ver = (struct Vertex*)malloc(V * sizeof(struct Vertex));
    graph->edge = (struct Edge*)malloc(100 * sizeof(struct Edge)); // Adjust size as needed
  
```

```

graph->edge_count = 0;
for (int i = 0; i < V; i++)
    graph->ver[i] = (struct Vertex){0, 0};
return graph;
}

void addEdge(struct Graph* graph, int u, int v, int capacity) {
    graph->edge[graph->edge_count++] = (struct Edge){0, capacity, u, v};
}

void preflow(struct Graph* graph, int s) {
    graph->ver[s].h = graph->V;
    for (int i = 0; i < graph->edge_count; i++) {
        if (graph->edge[i].u == s) {
            graph->edge[i].flow = graph->edge[i].capacity;
            graph->ver[graph->edge[i].v].e_flow += graph->edge[i].flow;
            graph->edge[graph->edge_count++] = (struct Edge){-graph->edge[i].flow, 0, graph->edge[i].v, s};
        }
    }
}

int overFlowVertex(struct Graph* graph) {
    for (int i = 1; i < graph->V - 1; i++)
        if (graph->ver[i].e_flow > 0)
            return i;
    return -1;
}

void updateReverseEdgeFlow(struct Graph* graph, int i, int flow) {
    int u = graph->edge[i].v, v = graph->edge[i].u;
    for (int j = 0; j < graph->edge_count; j++) {
        if (graph->edge[j].v == v && graph->edge[j].u == u) {
            graph->edge[j].flow -= flow;
            return;
        }
    }
    graph->edge[graph->edge_count++] = (struct Edge){0, flow, u, v};
}

int push(struct Graph* graph, int u) {
    for (int i = 0; i < graph->edge_count; i++) {
        if (graph->edge[i].u == u) {

```

```

    if (graph->edge[i].flow == graph->edge[i].capacity)
        continue;
    if (graph->ver[u].h > graph->ver[graph->edge[i].v].h) {
        int flow = (graph->edge[i].capacity - graph->edge[i].flow < graph->ver[u].e_flow) ?
            graph->edge[i].capacity - graph->edge[i].flow : graph->ver[u].e_flow;
        graph->ver[u].e_flow -= flow;
        graph->ver[graph->edge[i].v].e_flow += flow;
        graph->edge[i].flow += flow;
        updateReverseEdgeFlow(graph, i, flow);
        return 1;
    }
}
}
return 0;
}

void relabel(struct Graph* graph, int u) {
    int mh = INT_MAX;
    for (int i = 0; i < graph->edge_count; i++) {
        if (graph->edge[i].u == u) {
            if (graph->edge[i].flow == graph->edge[i].capacity)
                continue;
            if (graph->ver[graph->edge[i].v].h < mh) {
                mh = graph->ver[graph->edge[i].v].h;
                graph->ver[u].h = mh + 1;
            }
        }
    }
}

int getMaxFlow(struct Graph* graph, int s, int t) {
    preflow(graph, s);
    while (overFlowVertex(graph) != -1) {
        int u = overFlowVertex(graph);
        if (!push(graph, u))
            relabel(graph, u);
    }
    return graph->ver[t].e_flow;
}

int main() {
    int V = 6;
    struct Graph* g = createGraph(V);

```

```
addEdge(g, 0, 1, 16);
addEdge(g, 0, 2, 13);
addEdge(g, 1, 2, 10);
addEdge(g, 2, 1, 4);
addEdge(g, 1, 3, 12);
addEdge(g, 2, 4, 14);
addEdge(g, 3, 2, 9);
addEdge(g, 3, 5, 20);
addEdge(g, 4, 3, 7);
addEdge(g, 4, 5, 4);
int s = 0, t = 5;
printf("Maximum flow is %d\n", getMaxFlow(g, s, t));
free(g->ver);
free(g->edge);
free(g);
return 0;
}
```

- 2) For the above given graph Emma wants an s-t cut that requires the source s and the sink t to be in different subsets, and it consists of edges going from the source's side to the sink's side. So, she wants you to find the minimum capacity s-t cut of the given network. Expected output is all the edges of minimum cut.

Source code:

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

#define N 6
#define MIN(a, b) ((a) < (b) ? (a) : (b))

typedef struct {
    int u, v, cap, flow;
} Edge;

Edge edges[20];
int height[N], excess[N], edgeCount, visited[N];

void addEdge(int u, int v, int cap) {
    edges[edgeCount++] = (Edge){u, v, cap, 0};
    edges[edgeCount++] = (Edge){v, u, 0, 0};
}

void push(int i) {
    int flow = MIN(excess[edges[i].u], edges[i].cap - edges[i].flow);
    edges[i].flow += flow;
    edges[i^1].flow -= flow;
    excess[edges[i].u] -= flow;
    excess[edges[i].v] += flow;
}

void relabel(int u) {
    int minH = INT_MAX;
    for (int i = 0; i < edgeCount; i++)
        if (edges[i].u == u && edges[i].flow < edges[i].cap)
            minH = MIN(minH, height[edges[i].v]);
    height[u] = minH + 1;
}

void dfs(int u) {
```

```

visited[u] = 1;
for (int i = 0; i < edgeCount; i++)
    if (edges[i].u == u && edges[i].flow < edges[i].cap && !visited[edges[i].v])
        dfs(edges[i].v);
}

void minCut(int s, int t) {
    for (int i = 0; i < N; i++) visited[i] = 0;
    dfs(s);
    printf("Minimum s-t cut edges:\n");
    for (int i = 0; i < edgeCount; i++)
        if (visited[edges[i].u] && !visited[edges[i].v])
            printf("%d -> %d\n", edges[i].u, edges[i].v);
}

int maxFlow(int s, int t) {
    height[s] = N;
    for (int i = 0; i < edgeCount; i++)
        if (edges[i].u == s) {
            edges[i].flow = edges[i].cap;
            excess[edges[i].v] += edges[i].cap;
            excess[s] -= edges[i].cap;
        }
    while (1) {
        int pushed = 0;
        for (int i = 0; i < edgeCount; i++)
            if (excess[edges[i].u] > 0 && height[edges[i].u] > height[edges[i].v]) {
                push(i);
                pushed = 1;
            }
        if (!pushed) break;
        for (int u = 1; u < N - 1; u++)
            if (excess[u] > 0) relabel(u);
    }
    return excess[t];
}

```

```
int main() {  
    addEdge(0, 1, 16); addEdge(0, 2, 13);  
    addEdge(1, 2, 10); addEdge(2, 1, 4);  
    addEdge(1, 3, 12); addEdge(2, 4, 14);  
    addEdge(3, 2, 9); addEdge(3, 5, 20);  
    addEdge(4, 3, 7); addEdge(4, 5, 4);  
  
    printf("Maximum flow is %d\n", maxFlow(0, 5));  
    minCut(0, 5);  
    return 0;  
}
```

Post-Lab:

- 1) Hogwarts has yet again declared The Triwizard tournament. Harry must pass this round to get to the next one. Each participant will be given a graph with vertices as shown below. Each vertex is a dungeon, and a golden egg is placed in the root dungeon i.e., the root vertex. Help Harry find the dungeon with the golden egg using traversing or searching tree or graph data structure. (P.S: To pass this round Harry must write a code).

Source code:

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

#define MAX_NODES 100

typedef struct {
    int adj[MAX_NODES][MAX_NODES];
    int vertices;
} Graph;

void init_graph(Graph *g, int vertices) {
    g->vertices = vertices;
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            g->adj[i][j] = 0;
        }
    }
}

void add_edge(Graph *g, int u, int v) {
    g->adj[u][v] = 1;
}

void bfs(Graph *g, int start) {
    int queue[MAX_NODES], front = 0, rear = 0;
    int visited[MAX_NODES] = {0};

    queue[rear++] = start;
    visited[start] = 1;

    while (front < rear) {
        int node = queue[front++];
        printf("Dungeon %d checked\n", node);
        for (int i = 0; i < g->vertices; i++) {
```



```
        if (g->adj[node][i] && !visited[i]) {
            queue[rear++] = i;
            visited[i] = 1;
        }
    }
}
printf("Golden egg found at dungeon: %d\n", start);
}

int main() {
    Graph g;
    init_graph(&g, 8);
    add_edge(&g, 1, 2);
    add_edge(&g, 1, 3);
    add_edge(&g, 2, 4);
    add_edge(&g, 2, 5);
    add_edge(&g, 3, 6);
    add_edge(&g, 3, 7);

    int root_dungeon = 1;
    bfs(&g, root_dungeon);
    return 0;
}
```

- 2) KL University is starting next week. There are S subjects in total, and you need to choose K of them to attend each day, you required number of credits to pass the semester. There are $N+1$ buildings. Your hostel is in building number 0. Subject j is taught in building B_j . After each subject, you have a break, during which you go back to your hostel. There are M bidirectional paths of length 1 which connects building b_1 to building b_2 . Find the minimum total distance you need to travel each day if you choose your subjects correctly.

Input

2 3 2 2
0 1
1 2
2 0
1 2

Output

4

Source code:

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

#define MAX_BUILDINGS 100

int graph[MAX_BUILDINGS][MAX_BUILDINGS];
int distances[MAX_BUILDINGS], visited[MAX_BUILDINGS];

void bfs(int start, int n) {
    for (int i = 0; i <= n; i++) distances[i] = INT_MAX, visited[i] = 0;
    distances[start] = 0;

    for (int i = 0; i <= n; i++) {
        int minDist = INT_MAX, u = -1;
        for (int j = 0; j <= n; j++) {
            if (!visited[j] && distances[j] < minDist) {
                minDist = distances[j];
                u = j;
            }
        }
        if (u == -1) break;
        visited[u] = 1;
        for (int v = 0; v <= n; v++) {
            if (graph[u][v] && distances[u] + 1 < distances[v]) {
                distances[v] = distances[u] + 1;
            }
        }
    }
}
```

```

}

int compare(const void* a, const void* b) {
    return (*(int*)a - *(int*)b);
}

int main() {
    int N = 2, S = 3, K = 2, M = 2;
    int edges[2][2] = {{0, 1}, {1, 2}};
    int subjects[] = {1, 2};

    for (int i = 0; i <= N; i++)
        for (int j = 0; j <= N; j++)
            graph[i][j] = 0;

    for (int i = 0; i < M; i++) {
        int u = edges[i][0], v = edges[i][1];
        graph[u][v] = graph[v][u] = 1;
    }

    bfs(0, N);

    int subjectDistances[S];
    for (int i = 0; i < S; i++) subjectDistances[i] = distances[subjects[i]];
    qsort(subjectDistances, S, sizeof(int), compare);

    int totalDistance = 0;
    for (int i = 0; i < K; i++) totalDistance += 2 * subjectDistances[i];

    printf("%d\n", totalDistance);
    return 0;
}

```

<u>Comments of the Evaluators (if Any)</u>	<u>Evaluator's Observation</u>
	Marks Secured: _____ out of [50].
	Signature of the Evaluator Date of Evaluation:

Date of the Session: ____/____/____

Time of the Session: ____ to ____

EX – 9 Solving NQueens and Graph Coloring Problems using Back tracking methodology**Prerequisites:**

- Basics of Data Structures and C Programming.
- Basic knowledge about Arrays and Graphs.

Pre-Lab:

1) What is Back Tracking methodology and when can we use Back Tracking, explain with an example?

Backtracking

A recursive method to explore all possibilities by undoing invalid choices.

Usage

- Constraint satisfaction problems (Sudoku, N-Queens).
- Finding all solutions efficiently.

Example: N-Queens

Place queens row by row, backtrack if attacked, continue until solved.

- 2) Mr. Anumula went to the Ice-Cream Parlor. He will be with certain amount of money to buy ice-creams. When he goes to the counter for ordering the Ice-cream, there will be displayed with the items and its cost, respectively. He will be checking which items he can buy according to the money. Print "YES" if he can buy the Ice-Creams without leaving one rupee also otherwise print "NO".

Input

costs= [100, 70, 50, 180, 120, 200, 150]

Total Money=300

Output

YES

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

```
int can_buy_ice_creams(int costs[], int n, int total_money) {
    int dp[total_money + 1];
    for (int i = 0; i <= total_money; i++) {
        dp[i] = 0;
    }
    dp[0] = 1;

    for (int i = 0; i < n; i++) {
        for (int j = total_money; j >= costs[i]; j--) {
            if (dp[j - costs[i]] == 1) {
                dp[j] = 1;
            }
        }
    }

    return dp[total_money];
}
```

```
int main() {
    int costs[] = {100, 70, 50, 180, 120, 200, 150};
    int total_money = 300;
    int n = sizeof(costs) / sizeof(costs[0]);

    if (can_buy_ice_creams(costs, n, total_money)) {
        printf("YES\n");
    } else {
        printf("NO\n");
    }

    return 0;
}
```

In-Lab:

- 1) Mani is poor at playing chess, he was asked to arrange “N” Queens on the chess board in such a way that no two queens are allowed to kill each other. Since he is poor at chess you were asked to arrange them on behalf of him. (You must use Backtracking Approach)

Source code:

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

#define N 8

void printSolution(int board[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d ", board[i][j]);
        }
        printf("\n");
    }
}

bool isSafe(int board[N][N], int row, int col) {
    for (int i = 0; i < col; i++)
        if (board[row][i])
            return false;

    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;

    for (int i = row, j = col; i < N && j >= 0; i++, j--)
        if (board[i][j])
            return false;

    return true;
}

bool solveNQueens(int board[N][N], int col) {
    if (col >= N)
        return true;
}
```

```
for (int i = 0; i < N; i++) {
    if (isSafe(board, i, col)) {
        board[i][col] = 1;
        if (solveNQueens(board, col + 1))
            return true;
        board[i][col] = 0;
    }
}
return false;
}

void solve() {
    int board[N][N] = {0};
    if (!solveNQueens(board, 0)) {
        printf("Solution does not exist\n");
        return;
    }
    printSolution(board);
}

int main() {
    solve();
    return 0;
}
```

- 2) 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

Source code:

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

#define MAX_VERTICES 100

bool isSafe(int graph[MAX_VERTICES][MAX_VERTICES], int color[], int vertex, int c, int N) {
    for (int i = 0; i < N; i++) {
        if (graph[vertex][i] == 1 && color[i] == c) {
            return false;
        }
    }
    return true;
}

bool graphColoring(int graph[MAX_VERTICES][MAX_VERTICES], int m, int color[], int vertex,
int N) {
    if (vertex == N) {
        return true;
    }

    for (int c = 1; c <= m; c++) {
        if (isSafe(graph, color, vertex, c, N)) {
            color[vertex] = c;
            if (graphColoring(graph, m, color, vertex + 1, N)) {
                return true;
            }
            color[vertex] = 0;
        }
    }
}
```



```
    return false;
}

void canColorGraph(int N, int edges[][2], int E, int m) {
    int graph[MAX_VERTICES][MAX_VERTICES] = {0};
    int color[MAX_VERTICES] = {0};

    for (int i = 0; i < E; i++) {
        int u = edges[i][0];
        int v = edges[i][1];
        graph[u][v] = 1;
        graph[v][u] = 1;
    }

    if (graphColoring(graph, m, color, 0, N)) {
        printf("Possible\n");
    } else {
        printf("Not Possible\n");
    }
}

int main() {
    int N = 3;
    int m = 2;
    int edges[][2] = {{0, 2}, {1, 2}};
    int E = sizeof(edges) / sizeof(edges[0]);

    canColorGraph(N, edges, E, m);

    return 0;
}
```

Post-Lab:

- 1) John is a very obedient boy and helping others is a habit of him. He will do any work with dedication his teacher assigns him a work to generate all permutations of given word. Since he is busy helping others, you are asked to help him to complete his work on behalf of him. (Use Backtracking Concept)

Source code:

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

void swap(char *x, char *y) {
    char temp;
    temp = *x;
    *x = *y;
    *y = temp;
}

void permute(char *str, int l, int r) {
    if (l == r) {
        printf("%s\n", str);
    } else {
        for (int i = l; i <= r; i++) {
            swap((str + l), (str + i));
            permute(str, l + 1, r);
            swap((str + l), (str + i));
        }
    }
}

int main() {
    char str[] = "ABC";
    int n = strlen(str);
    permute(str, 0, n - 1);
    return 0;
}
```

- 2) Mr. Sai joined for as an assistant at a school where he was given a job to arrange schedules for the subject n1, n2, n3, n4, n5, n6, n7, n8. Help him schedule the timetable from the given information.

Let this be the schedule:

Here for everyone hour there are many subjects competing. Use backtracking and create a schedule where each subject is assigned to a specific hour in a day. In the schedule '1' represents that the subject is competing for that hour.

	Subjects		1		2		3		4		5		6		7		8	
Hours		N1	N2	N3	N4	N5	N6	N7	N8									
	1	0	1	0	1	1	1	0	1									
	2	1	0	0	0	1	1	0	0									
	3	0	0	0	0	0	1	1	1									
	4	1	0	0	0	0	0	0	1									
	5	1	1	0	0	0	1	0	0									
	6	1	1	1	0	1	0	1	0									
	7	0	0	1	0	0	1	0	0									
	8	1	0	1	1	0	0	0	0									

Source code:

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

#define SUBJECTS 8
#define HOURS 8

bool is_safe(int subject, int hour, int schedule[], int conflicts[SUBJECTS][SUBJECTS]) {
    for (int i = 0; i < SUBJECTS; i++) {
        if (schedule[i] == hour && conflicts[subject][i]) {
            return false;
        }
    }
    return true;
}

bool backtrack(int schedule[], int subjects[], int hours[], int conflicts[SUBJECTS][SUBJECTS],
int index) {
    if (index == SUBJECTS) return true;

    for (int i = 0; i < HOURS; i++) {
        if (is_safe(subjects[index], hours[i], schedule, conflicts)) {
            schedule[subjects[index]] = hours[i];
            if (backtrack(schedule, subjects, hours, conflicts, index + 1)) return true;
            schedule[subjects[index]] = -1;
        }
    }
}
```

```

    }
    return false;
}

void schedule_subjects(int conflict_matrix[SUBJECTS][SUBJECTS]) {
    int subjects[SUBJECTS] = {0, 1, 2, 3, 4, 5, 6, 7};
    int hours[HOURS] = {1, 2, 3, 4, 5, 6, 7, 8};
    int schedule[SUBJECTS];
    for (int i = 0; i < SUBJECTS; i++) schedule[i] = -1;

    if (backtrack(schedule, subjects, hours, conflict_matrix, 0)) {
        for (int i = 0; i < SUBJECTS; i++) {
            printf("N%d -> Hour %d\n", i + 1, schedule[i]);
        }
    } else {
        printf("No valid schedule found\n");
    }
}

int main() {
    int conflict_matrix[SUBJECTS][SUBJECTS] = {
        {0, 1, 0, 1, 1, 1, 0, 1},
        {1, 0, 0, 0, 1, 1, 0, 0},
        {0, 0, 0, 0, 0, 1, 1, 1},
        {1, 0, 0, 0, 0, 0, 0, 1},
        {1, 1, 0, 0, 0, 1, 0, 0},
        {1, 1, 1, 0, 1, 0, 1, 0},
        {0, 0, 1, 0, 0, 1, 0, 0},
        {1, 0, 1, 1, 0, 0, 0, 0}
    };

    schedule_subjects(conflict_matrix);
    return 0;
}

```

Comments of the Evaluators (if Any)	Evaluator's Observation
	Marks Secured: _____ out of [50].
	Signature of the Evaluator Date of Evaluation:

Date of the Session: ____/____/____

Time of the Session: ____ to ____

EX – 10 Sum of Subsets Problem using Back tracking**Prerequisites:**

- Basics of Data Structures and C Programming.
- Basic knowledge about arrays.

Pre-Lab:

- 1) Dark Peace is volunteer in VISION 2K17. ChiefCO gave him a task to complete it before the CODEGEEKS which is on 11th march. ChiefCo asks DarkPeace to do the work as soon as possible. So, to complete the task as soon as possible Dark Peace asks you to help him. You will be given two set of length N and M. Your task is to find whether the subset is present in the first set whose sum of elements is equal to the member of the set. You must check for every member of the second set. Print Yes if subset is present and if not present print No and if the member exceeds the maximum sum of

Input

3 3

1 7 4

10 14 4

Output

No -1 Yes

Explanation

For first member no subset gives sum to 10 so we print No. For second since maximum sum is 12 and 14 is greater than 12 so we print -1. For last subset {4} exists whose sum is equal to 4 and hence we print Yes.

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

```
#include <stdbool.h>
```

```
bool subset_sum_possible(int arr[], int n, int target_sum) {
```

```
    if (target_sum == 0) return true;
```

```
    bool dp[target_sum + 1];
```

```
    dp[0] = true;
```

```
    int i, j;
```

```
    for (i = 1; i <= target_sum; i++) {
```

```
        dp[i] = false;
    }

    for (i = 0; i < n; i++) {
        for (j = target_sum; j >= arr[i]; j--) {
            if (dp[j - arr[i]]) {
                dp[j] = true;
            }
        }
    }

    return dp[target_sum];
}

int main() {
    int first_set[] = {1, 7, 4};
    int second_set[] = {10, 14, 4};
    int N = sizeof(first_set) / sizeof(first_set[0]);
    int M = sizeof(second_set) / sizeof(second_set[0]);

    int i, target;

    int max_sum = 0;
    for (i = 0; i < N; i++) {
        max_sum += first_set[i];
    }

    for (i = 0; i < M; i++) {
        target = second_set[i];

        if (target > max_sum) {
            printf("-1 ");
        } else {
```

```
    if (subset_sum_possible(first_set, N, target)) {  
        printf("Yes ");  
    } else {  
        printf("No ");  
    }  
}  
}  
}  
  
return 0;  
}
```

In-Lab:

- 1) Geeta is working in a company, and she has n different projects to work on, where every project is scheduled to be done from `startTime[i]` to `endTime[i]`, obtaining a profit of `profit[i]`. You are given the `startTime`, `endTime` and `profit` arrays, return the maximum profit you can take such that there are no two projects that she is working on in that given subset with overlapping time range. If she chooses a project that ends at time a then she will be able to start another project that starts at time b.

Input

`startTime` = [1,2,3,4,6], `endTime` = [3,5,10,6,9], `profit` = [20,20,100,70,60]

Output

150

Explanation: The subset chosen is the first, fourth and fifth project.

Profit obtained $150 = 20 + 70 + 60$.

Source code:

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

typedef struct {
    int start, end, profit;
} Project;

int compare(const void *a, const void *b) {
    return ((Project *)a)->end - ((Project *)b)->end;
}

int findLastNonOverlapping(Project projects[], int i) {
    int low = 0, high = i - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (projects[mid].end <= projects[i].start) {
            if (projects[mid + 1].end <= projects[i].start) {
                low = mid + 1;
            } else {
                return mid;
            }
        } else {
            high = mid - 1;
        }
    }
    return -1;
}
```



```

int findMaxProfit(int startTime[], int endTime[], int profit[], int n) {
    Project projects[n];
    for (int i = 0; i < n; i++) {
        projects[i].start = startTime[i];
        projects[i].end = endTime[i];
        projects[i].profit = profit[i];
    }
    qsort(projects, n, sizeof(Project), compare);

    int dp[n];
    dp[0] = projects[0].profit;

    for (int i = 1; i < n; i++) {
        dp[i] = dp[i - 1];
        int lastNonOverlap = findLastNonOverlapping(projects, i);
        if (lastNonOverlap != -1) {
            dp[i] = (dp[i] > projects[i].profit + dp[lastNonOverlap]) ? dp[i] : (projects[i].profit + dp[lastNonOverlap]);
        } else {
            dp[i] = (dp[i] > projects[i].profit) ? dp[i] : projects[i].profit;
        }
    }

    return dp[n - 1];
}

int main() {
    int startTime[] = {1, 2, 3, 4, 6};
    int endTime[] = {3, 5, 10, 6, 9};
    int profit[] = {20, 20, 100, 70, 60};
    int n = sizeof(startTime) / sizeof(startTime[0]);

    printf("Maximum Profit: %d\n", findMaxProfit(startTime, endTime, profit, n));

    return 0;
}

```

- 2) The subset sum problem is an important problem in computer science. Below we will provide a simple algorithm for solving this problem. The challenge is to determine if there is some subset of numbers in an array that can sum up to some number S. These algorithms can both be implemented to solve Array Addition I and Array Addition.

Simple (Naive) solution

The algorithm for the exponential time, naive solution, is as follows: First we will generate every possible set (the power set), and then check if the sum of any of these sets equals the sum S. For example: arr = [1, 2, 3] sum = 5

All possible sets: [] [1] [2] [3] [1, 2] [1, 3] [2, 3] [1, 2, 3]

We can see that we can get a sum of 5 by adding the elements in the set [2, 3]. To generate all possible sets of an array, we will implement the following algorithm:

- (1) The initial power set only contains the empty set: [[]]
- (2) We loop through each element in the array and add it to every element in the powerset.
- (3) Then we take the union of these two sets.
- (4) Once we get the power set, we check to see if the sum equals our goal S.

Example

arr = [1, 2, 3] sum = 5 sets = [[]]

Step 1: Add 1 to the power set [[], [1]]

Step 2: Add 2 to the power set [[], [1], [1, 2], [2]]

Step 3: Add 3 to the power set [[], [1], [1, 2], [2], [1, 3], [2, 3], [1, 2, 3], [3]]

Source code:

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

void findSubsets(int arr[], int n, int sum) {
    int totalSubsets = 1 << n;
    for (int i = 0; i < totalSubsets; i++) {
        int currentSum = 0;
        printf("[");
        for (int j = 0; j < n; j++) {
            if (i & (1 << j)) {
                currentSum += arr[j];
                printf("%d ", arr[j]);
            }
        }
        printf("\n");
        if (currentSum == sum) {
            printf("Found a subset with the required sum: %d\n", sum);
        }
    }
}
```

```
int main() {  
    int arr[] = {1, 2, 3};  
    int sum = 5;  
    int n = sizeof(arr) / sizeof(arr[0]);  
    findSubsets(arr, n, sum);  
    return 0;  
}
```

Post-Lab:

- 1) 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

333

1

3

Output

6

Source code:

```
#include <stdio.h>
int count_subsets(int student_ids[], int n, int x_numbers[], int m) {
    int count[10] = {0};

    for (int i = 0; i < n; i++) {
        count[student_ids[i]]++;
    }

    for (int i = 0; i < m; i++) {
        if (count[x_numbers[i]] == 0) {
            return 0;
        }
    }

    int total_subsets = 1;
    for (int i = 0; i < m; i++) {
        total_subsets *= count[x_numbers[i]];
    }

    return total_subsets;
}

int main() {
    int student_ids[] = {3, 3, 3};
    int x_numbers[] = {1, 3};
    int n = sizeof(student_ids) / sizeof(student_ids[0]);
    int m = sizeof(x_numbers) / sizeof(x_numbers[0]);

    int result = count_subsets(student_ids, n, x_numbers, m);

    printf("%d\n", result);
    return 0;
}
```

- 2) You are offered N pay packages by various companies, Congratulations! You can only select 2 unique groups of packages each with a total of at least K. Select the 2 groups such that you choose minimum number of companies adding up to the total of K.

Input

arr[] = {2, 4, 5, 6, 7, 8}, K = 16

Output

6

Explanation:

The subsets {2, 6, 8} and {4, 5, 7} are the two smallest subsets with sum K (= 16).

Therefore, the sum of the lengths of both these subsets = 3 + 3 = 6.4

Source code:

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

#define MAX 100

int minCompanies(int arr[], int n, int K) {
    int dp[MAX] = {0};
    for (int i = 1; i <= K; i++) {
        dp[i] = MAX;
    }
    dp[0] = 0;

    for (int i = 0; i < n; i++) {
        for (int j = K; j >= arr[i]; j--) {
            if (dp[j - arr[i]] + 1 < dp[j]) {
                dp[j] = dp[j - arr[i]] + 1;
            }
        }
    }

    return dp[K];
}

int main() {
    int arr[] = {2, 4, 5, 6, 7, 8};
    int K = 16;
    int minCount = minCompanies(arr, sizeof(arr) / sizeof(arr[0]), K);

    printf("%d\n", minCount * 2);
    return 0;
}
```

<u>Comments of the Evaluators (if Any)</u>	<u>Evaluator's Observation</u>
	Marks Secured: _____ out of [50].
	Signature of the Evaluator Date of Evaluation:

Date of the Session: ____ / ____ / ____

Time of the Session: ____ to ____

EX – 11 Solving Problems using Branch and Bound technique**Prerequisites:**

- Basics of Data Structures and C Programming.
- Basic knowledge about Arrays.

Pre-Lab:

- 1) Match the data structure used to generate branch and bound strategy.

Queue Data Structure	LIFO Branch and Bound Strategy
Stack Data Structure	Best First branch and bound strategy
Priority Queue	FIFO Branch and Bound Strategy

1. Queue Data Structure → FIFO Branch and Bound Strategy
2. Stack Data Structure → LIFO Branch and Bound Strategy
3. Priority Queue → Best First Branch and Bound Strategy


- 2) You are given a knapsack that can carry a maximum weight of 60. There are 4 items with weights {20, 30, 40, 70} and values {70, 80, 90, 200}. What is the maximum value of the items you can carry using the knapsack?

Given:

- Knapsack capacity = 60
- Items (weight, value):
 - (20, 70)
 - (30, 80)
 - (40, 90)
 - (70, 200)

Solution:

We check combinations of items that fit within 60:

1. (20, 70) + (30, 80) = 50 weight, 150 value
2. (20, 70) + (40, 90) = 60 weight, 160 value  (Best choice)
3. (30, 80) + (40, 90) = 70 weight (exceeds capacity, not possible)
4. Single item choices:
 - (20, 70) → 70 value
 - (30, 80) → 80 value
 - (40, 90) → 90 value

Maximum Value = 160 (Items: 20 & 40)

In-Lab :

- 1) A Professor is teaching about Binary Numbers. He wants to know all the possible binary numbers of a given length. Help the Professor to generate all the set of binary strings of length N in ascending order using Branch and Bound Technique.

Input

N = 3

Output

000

001

010

011

100

101

110

111

Explanation:

Numbers with 3 binary digits are

0, 1, 2, 3, 4, 5, 6, 7

Source code:

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

```
void generateBinary(int n, char *binary, int index) {
    if (index == n) {
        binary[index] = '\0';
        printf("%s\n", binary);
        return;
    }
```

```
    binary[index] = '0';
    generateBinary(n, binary, index + 1);
```

```
    binary[index] = '1';
    generateBinary(n, binary, index + 1);
}
```

```
int main() {
    int N = 3;
    char binary[N + 1];
    generateBinary(N, binary, 0);
    return 0;
}
```


- 2) You are in a supermarket shopping for fruits with huge discounts. There are N varieties of fruits available each with a weight C and discount P. Now, select K fruits in a way that maximizes the discount. You only have a bag which can carry a weight of W.

Input

N = 5

P [] = {2, 7, 1, 5, 3} C [] = {2, 5, 2, 3, 4}, W = 8, K = 2.

Output

12

Explanation:

Here, the maximum possible profit is when we take 2 items: item2 (P[1] = 7 and C[1] = 5) and item4 (P[3] = 5 and C[3] = 3).

Hence, maximum profit = 7 + 5 = 12

Source code:

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

```
int maxDiscount(int N, int P[], int C[], int W, int K) {
    int dp[K + 1][W + 1];
    for (int i = 0; i <= K; i++)
        for (int j = 0; j <= W; j++)
            dp[i][j] = 0;

    for (int i = 0; i < N; i++) {
        for (int k = K; k > 0; k--) {
            for (int w = W; w >= C[i]; w--) {
                dp[k][w] = (dp[k][w] > dp[k - 1][w - C[i]] + P[i]) ? dp[k][w] : (dp[k - 1][w - C[i]] + P[i]);
            }
        }
    }
    return dp[K][W];
}
```

```
int main() {
    int N = 5;
    int P[] = {2, 7, 1, 5, 3};
    int C[] = {2, 5, 2, 3, 4};
    int W = 8;
    int K = 2;

    int result = maxDiscount(N, P, C, W, K);
    printf("%d\n", result);
    return 0;
}
```

Post-Lab:

- 1) Given an array of positive elements, you must flip the sign of some of its elements such that the resultant sum of the elements of array should be minimum non-negative (as close to zero as possible). Return the minimum no. of elements whose sign needs to be flipped such that the resultant sum is minimum non-negative. Note that the sum of all the array elements will not exceed 104.

Input

arr [] = {15, 10, 6}

Output

1

Here, we will flip the sign of 15
and the resultant sum will be 1.

Source code:

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

#define MAX_SUM 104

int minimum_flips(int arr[], int n) {
    int total_sum = 0;
    for (int i = 0; i < n; i++) {
        total_sum += arr[i];
    }

    int target = total_sum / 2;
    int dp[target + 1];
    for (int i = 0; i <= target; i++) {
        dp[i] = 0;
    }
    dp[0] = 1;

    for (int i = 0; i < n; i++) {
        for (int j = target; j >= arr[i]; j--) {
            if (dp[j - arr[i]]) {
                dp[j] = 1;
            }
        }
    }

    int closest_sum = 0;
    for (int i = target; i >= 0; i--) {
        if (dp[i]) {
```

closest_sum = i;

```
        break;
    }
}

int closest_sum_to_zero = total_sum - 2 * closest_sum;
int flips = 0;
int remaining_sum = closest_sum;

int sorted_arr[n];
for (int i = 0; i < n; i++) {
    sorted_arr[i] = arr[i];
}
for (int i = 0; i < n - 1; i++) {
    for (int j = i + 1; j < n; j++) {
        if (sorted_arr[i] < sorted_arr[j]) {
            int temp = sorted_arr[i];
            sorted_arr[i] = sorted_arr[j];
            sorted_arr[j] = temp;
        }
    }
}
for (int i = 0; i < n; i++) {
    if (remaining_sum >= sorted_arr[i]) {
        remaining_sum -= sorted_arr[i];
        flips++;
    }
    if (remaining_sum == 0) {
        break;
    }
}

return flips;
}

int main() {
    int arr[] = {15, 10, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    int result = minimum_flips(arr, n);
    printf("Minimum flips needed: %d\n", result);

    return 0;
}
```

- 2) There are some processes that need to be executed. The amount of load that process causes a server that runs it, is being represented by a single integer. The total load caused on a server is the sum of the loads of all the processes that run on that server. You have at your disposal two servers, on which the mentioned processes can be run. Your goal is to distribute given processes between those two servers in a way that, the absolute difference of their loads will be minimized.

Given an array of A[] of N integers, which represents loads caused by successive processes, the task is to print the minimum absolute difference of server loads.

Input

A[] = {1, 2, 3, 4, 5}

Output

1

Explanation:

Distribute the processes with loads {1, 2, 4} on the first server and {3, 5} on the second server, so that their total loads will be 7 and 8, respectively.

The difference of their loads will be equal to 1.

Source code:

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

int minAbsDifference(int A[], int N) {
    int totalLoad = 0;
    for (int i = 0; i < N; i++) {
        totalLoad += A[i];
    }

    int halfLoad = totalLoad / 2;
    int dp[halfLoad + 1];
    for (int i = 0; i <= halfLoad; i++) {
        dp[i] = 0;
    }

    for (int i = 0; i < N; i++) {
        for (int j = halfLoad; j >= A[i]; j--) {
            dp[j] = dp[j] > dp[j - A[i]] + A[i] ? dp[j] : dp[j - A[i]] + A[i];
        }
    }

    return totalLoad - 2 * dp[halfLoad];
}
```

```
int main() {  
    int A[] = {1, 2, 3, 4, 5};  
    int N = sizeof(A) / sizeof(A[0]);  
    int result = minAbsDifference(A, N);  
    printf("%d\n", result);  
    return 0;  
}
```

<u>Comments of the Evaluators (if Any)</u>	<u>Evaluator's Observation</u>
	Marks Secured: _____ out of [50]. Signature of the Evaluator Date of Evaluation:

Date of the Session: ____ / ____ / ____

Time of the Session: ____ to ____

EX – 12 Working with NP Hard and NP Complete Problems.**Prerequisites:**

- Basics of Data Structures and C Programming.
- Basic knowledge about arrays and graphs.

Pre-Lab:

1) Read the following conversation

Jaya: Travelling salesman problem is a NP hard problem.

Hema: I do not think so

Jaya: No, I am so sure that Travelling Salesman problem is a NP hard problem.

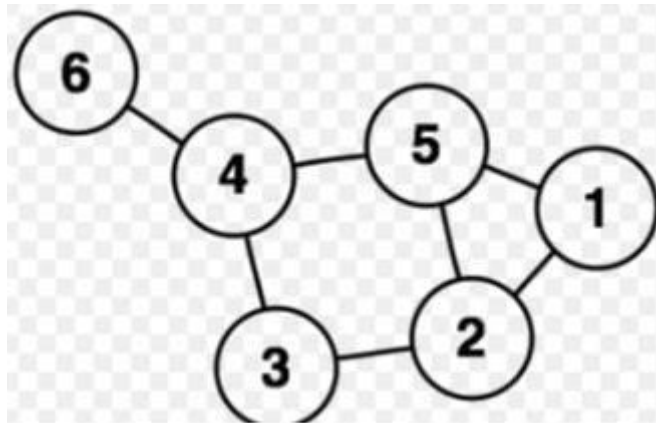
Hema: ...!!

You are Jaya's friend. Help her prove her statement.

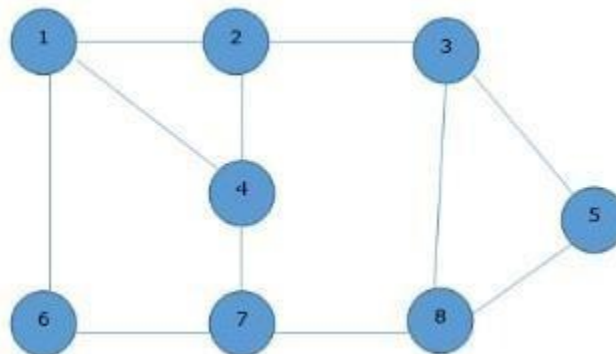
- **Definition:** A problem is **NP-hard** if solving it efficiently would allow us to solve all problems in NP efficiently.
- **Reduction:** The **Hamiltonian Cycle Problem** (which is NP-complete) can be reduced to the **Travelling Salesman Problem (TSP)** in polynomial time.
- **Implication:** Since Hamiltonian Cycle is NP-hard, and TSP is at least as hard as it, TSP is also **NP-hard**.

- 2) Consider the two graphs and find out the node cover for the each of the graphs and specify the maximum node cover.

a)



b)



Graph 1 Analysis:

The first graph consists of 6 nodes. To find a **minimum vertex cover**, we aim to cover all edges with the fewest nodes.

- One possible minimum node cover: {2, 3, 4, 5}
- Maximum node cover: All 6 nodes.

Graph 2 Analysis:

The second graph consists of 8 nodes and has a more structured layout.

- One possible minimum node cover: {2, 3, 4, 5, 6, 8}
- Maximum node cover: All 8 nodes.

In both graphs, the **maximum node cover** is the entire set of nodes.

In-Lab :

- 1) Raju prepares for the examination, but he got stuck into a concept called "NP-HARD AND "NP-COMPLETE PROBLEMS" on Nondeterministic Algorithms. So, help Raju to score good marks. Help him to define the Nondeterministic algorithms by sorting an array.

Source code:

A **Nondeterministic Algorithm** consists of two phases:

1. **Guessing (Nondeterministic Phase):** Generates a potential solution.
2. **Verification (Deterministic Phase):** Checks if the guessed solution is correct in polynomial time.

Sorting an Array using a Nondeterministic Algorithm:

1. **Guess:** Randomly generate a permutation of the array.
2. **Verify:** Check if the guessed permutation is sorted in $O(n)$ time.

- 2) Karthik and Bavya are trying to implement a program that helps a transportation company use its container to maximum efficiency, as part of their CS project.
- Few objects are more valuable than others. Each object has a certain weight. The transportation company wants to fill the container so that the value of the container (sum of value of objects in the container) is maximum, while total weight of the objects does not exceed the container's capacity.
- As the outcome is not fixed, this is a non-deterministic problem. This is a knapsack problem as we must maximize value within the given weight limit.
- So, to understand the problem well and implement it, help them in finding non-deterministic knapsack algorithm.

Source code:

```
#include <stdio.h>

int max(int a, int b) {
    return (a > b) ? a : b;
}

int knapsack(int i, int weight, int value, int W, int weights[], int values[], int n) {
    if (weight > W) return 0;
    if (i == n) return value;

    int include = knapsack(i + 1, weight + weights[i], value + values[i], W, weights, values, n);
    int exclude = knapsack(i + 1, weight, value, W, weights, values, n);

    return max(include, exclude);
}

int main() {
    int weights[] = {2, 3, 4, 5};
    int values[] = {3, 4, 5, 6};
    int W = 5;
    int n = sizeof(weights) / sizeof(weights[0]);

    printf("Maximum Value: %d\n", knapsack(0, 0, 0, W, weights, values, n));
    return 0;
}
```

Post-Lab:

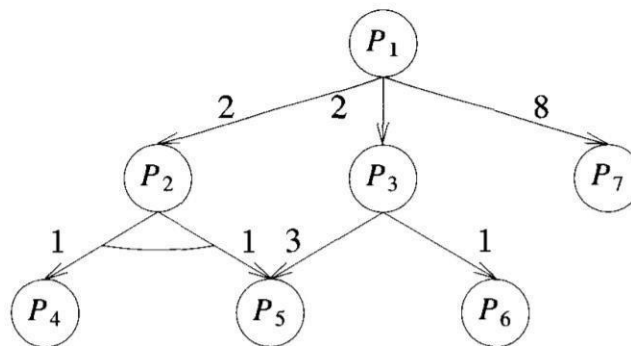
- 1) Hema: Hamiltonian Path is NP-Complete.
Jaya: Well, prove that!
Hema: I will prove and let you know.
Help Hema to try and prove that Hamilton Path is NP-Complete

Source code:

Proof that Hamiltonian Path is NP-Complete

1. **In NP:** A given path can be verified in polynomial time.
2. **NP-Hard:** Reduce from **Hamiltonian Cycle** (HC, known NP-Complete).
 - Given a graph G with HC, pick a vertex v , split it into v_1, v_2 .
 - If G has HC, the modified graph has a Hamiltonian Path from v_1 to v_2 .
3. Since Hamiltonian Path is in NP and NP-Hard, it is NP-Complete.

- 2) Hemanth was unable to answer the following question in exam. Here is the question, help Hemanth to find P1 solution. Find the total cost to find the P1 solution



Source code:

Paths and Costs:

- $P_4 \rightarrow P_2 \rightarrow P_1 = 1 + 2 = 3$
- $P_5 \rightarrow P_3 \rightarrow P_1 = 3 + 2 = 5$
- $P_6 \rightarrow P_3 \rightarrow P_1 = 1 + 2 = 3$
- $P_7 \rightarrow P_1 = 8$

Minimum Cost Path:

The shortest path to P_1 is from $P_4 \rightarrow P_2 \rightarrow P_1$ or $P_6 \rightarrow P_3 \rightarrow P_1$, both having a total cost of 3.

Answer:

The minimum total cost to reach P_1 is 3.

Comments of the Evaluators (if Any)

Evaluator's Observation

Marks Secured: _____ out of [50].

Signature of the Evaluator

Date of Evaluation: