URBAN EDGE

# INDEX

Name R. Abhinav Subject ADA Lab

Standard Section 4D Roll No. School / College

| S. No. | Date | Title | Page No. | Teacher's Sign |
|---|---|---|---|---|
| 1 | 2/05/24 | Leetcode: Repeated substring pattern | 2/5/24 | |
| 2) | 9/05/24 | Leetcode: Kth largest sum in a binary tree | 9/5/24 | |
| 3) | 16/05/24 | Leetcode: Increasing Order Search Tree | | |
| 4 | 23/05/24 | Topological sort using source removal and DFS | | |
| 5) | 30/05/24 | Sorting Techniques: i) Selection sort ii) Merge sort | 13/6/24 | |
| 6) | 6/06/24 | Sorting Technique: i) Quick sort | | |
| 7 | 13/06/24 | i) Johnson trotter ii) Pattern Matching (Brute Force) | | |
| 8 | 20/06/24 | i) Heap sort ii) Quick sort | 20/6/24 | |

| S. No. | Date | Title | Page No. | Teacher's Sign |
|---|---|---|---|---|
| 9 | 4/7/24 | i) Knapsack problem<br>ii) Prim's algorithm. | | |
| 10) | 11/7/24 | Greedy knapsack<br>Dijkstra and Kruskal algorithm | | |

Lab-1 URBAN EDGE 2.05.24

# Repeated Substring Pattern

1)
```
char *substring(char *s, int start, int end){

    char *res=(char*)malloc((end-start)+1)*
                                sizeof(char));

    int j=0;
    for(int i=start; i<end; i++)
        res[j++]=s[i];
    res[j]='\0';
    return res;
}

bool repeatedSubstringPattern(char* s){

    int len=strlen(s);

    for(int i=1; i<len; i++){
        if(len%i==0){
            int fac=0;
            char *s1=substring(s,0,i);
            char *s2;
            int flag;
            do{
                s2=substring(s, fac, fac+i);
                fac+=i;
                flag=strcmp(s1,s2);
            }while(flag==0 || fac+i<=len);
            free(s1);
            free(s2);
            if(fac==len && flag==0)
                return true;
        }
    }
}
```

Evaluated 24.

→ Output:

i) abab → true

ii) aba → false

iii) abcabc → false

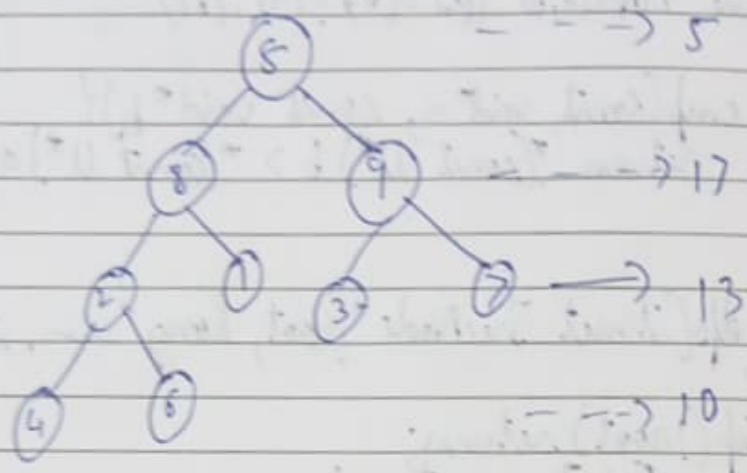
Lab-2 29.05.2024

# $K^{th}$ Largest sum in a Binary Tree

```
#define ll long long
#define MAX(a,b) (b>a)?(a):(b)

int comp(const void* a, const void* b){
    return *(const ll*) b > *(const ll*) a;
}

void dfs(struct TreeNode* root, long* sum, int idx, int d){

    if (!root) return;
    sum[idx] += root->val;
    dfs(root->left, sum, idx+1, d);
    dfs(root->right, sum, idx+1, d);
    (*d) = MAX((*d), idx);
}

long long kthLargestLevelSum(struct TreeNode* root, int k)
{
    int d=0;
    ll* sum = (ll*) calloc(100000, sizeof(ll));
    dfs(root, sum, 0, &d);
    qsort(sum, d+1, sizeof(ll), comp);
    ll ans = (k-1 < d+1)? sum[k-1] : -1;
    free(sum);
    return ans;
}
```

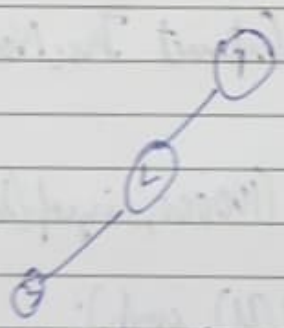
Output:

i) root = [5, 8, 9, 2, 1, 3, 7, 4, 6] , K = 2




Output: 13

ii) root = [1, 2, NULL, 3] , K = 1




Output 3

URBAN EDGE 16.05.2024

# Lab-3

## Increasing Order Search Tree

1) //Program to convert tree to increasing order

```
struct TreeNode* increasingBST(struct TreeNode* root)
{
    if(!root)
        return NULL;

    struct TreeNode* left = increasingBST(root->left);
    struct TreeNode* right = increasingBST(root->right);

    root->left = NULL;
    root->right = right;

    if(!left)
        return root;

    ~~while~~
    ~~struct TreeNode* iter = left;~~
    ~~while (iter && iter->right)~~
        ~~iter = iter->right;~~

    iter->right = root;
    return left;
}
```

URBAN EDGE 16.05.2024

# Lab-3

## Increasing Order Search Tree

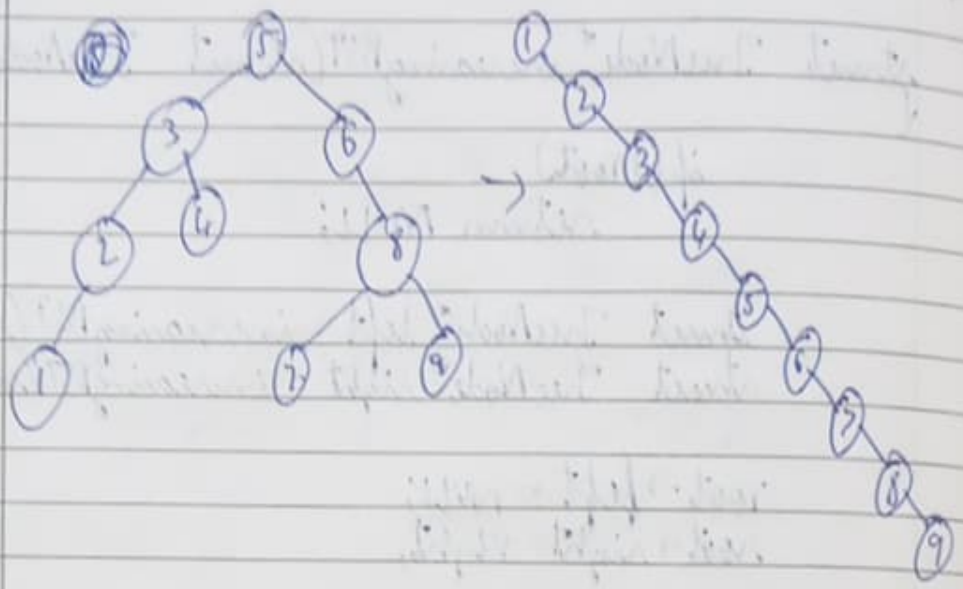
1) //Program to convert tree to increasing order

```
struct TreeNode* increasingBST(struct TreeNode* root)
{
    if(!root)
        return NULL;

    struct TreeNode* left = increasingBST(root->left);
    struct TreeNode* right = increasingBST(root->right);

    root->left = NULL;
    root->right = right;

    if(!left)
        return root;

    ~~while~~
    ~~struct TreeNode* iter = left;~~
    ~~while (iter && iter->right)~~
        ~~iter = iter->right;~~

    iter->right = root;
    return left;
}
```

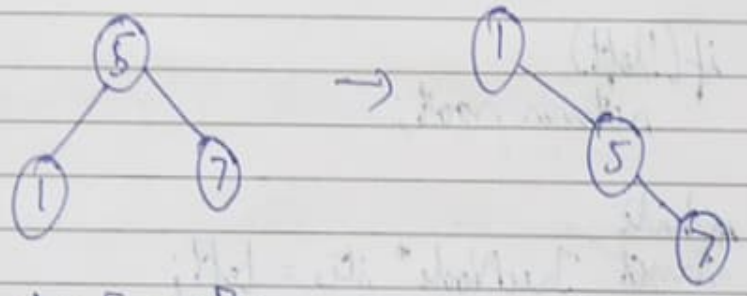
Output:

i) root

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



ii)



root = [5, 1, 7]

URBAN EDGE 23.05.24

# Lab-4

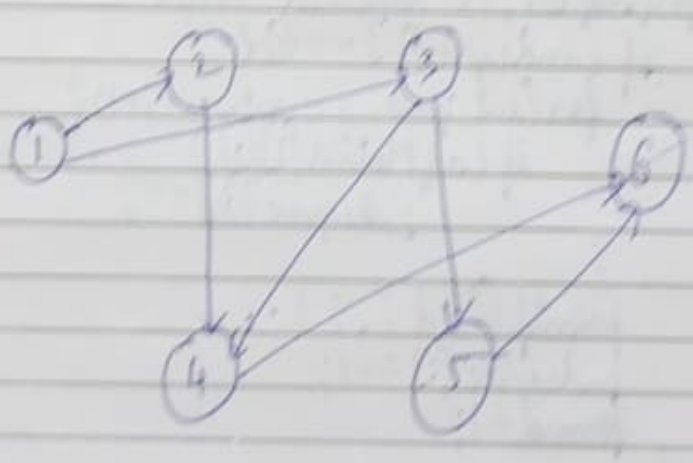
## Topological Sort Algorithm

```
#include <stdio.h>

#define MAX 100

void topologicalSort(int vertices, int adjMatrix[MAX][MAX])
{
    int indegree[MAX] = {0};

    for(int i=0; i<vertices; i++){
        for(int j=0; j<vertices; j++)
            indegree[j] += adjMatrix[i][j];
    }

    for(int count=0; count<vertices; count++) {
        int found=0;
        for(int i=0; i<vertices; i++){
            if (indegree[i]==0){
                for(int j=0; j<vertices; j++){
                    if(adjMatrix[i][j])
                        indegree[j]--;
                }
                printf("%d ", i);
                indegree[i]=-1;
                found=1;
            }
        }
    }
    if(!found){
        printf("\nCycle detected. Topological sort not
                possible.\n");
    }
    return;
}
```

```
int main()
    int vertices = 6;
    int adjMatrix[MAX][MAX] = {
                {0, 1, 1, 0, 0, 0},
                {0, 0, 0, 1, 0, 0},
                {0, 0, 0, 1, 1, 0},
                {0, 0, 0, 0, 0, 1},
                {0, 0, 0, 0, 0, 1},
                {0, 0, 0, 0, 0, 0},
    }
    printf("Topological Sort: ");
    topologicalSort(vertices, adjMatrix);
    return;
}
```

Output:

Topological Sort: ~~[illegible]~~ 1 2 3 4 5 6



ii) DFS

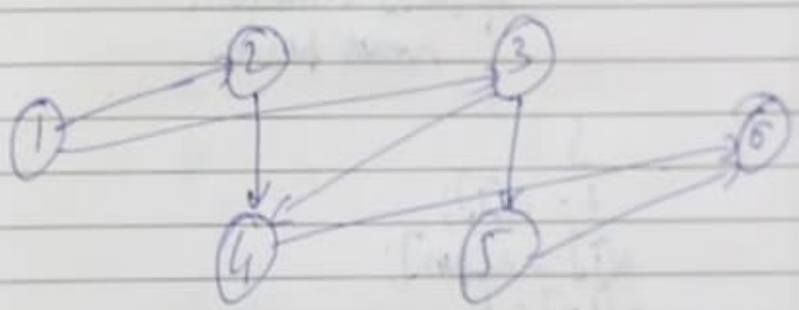
```
#include<stdio.h>

#define MAX 100

int graph[MAX][MAX];
int stack[MAX];
int top =-1;

void initialize(){
    int i,j;

    for(i=0; i<MAX; i++){
        visited[i] = false;
        stack[i] = -1;
        for(j=0; j<MAX; j++)
            graph[i][j]=0;
    }

}

void dfs(int node){
    visited[Node]=true;
    int i,
    for(i=0; i<numNodes; i++){
        if( graph[node][i] && !visited[i])
            dfs(i);
    }
    stack[++top]=node;
}
```

```
void topologicalSort(){
    int i;
    for(i=0; i<numNodes; i++){
        if(!visited[i])
            dfs(i);
    }

    printf("Topological sort:");
}

int main(){
    int i,j;
    printf("Enter number of nodes:");
    scanf("%d",&numNodes);

    printf("Enter adjacency matrix:\n");

    for(i=0; i<numNodes; i++)
        scanf("%d",&graph[i][j]);
    }

    initializing();
    topologicalSort();

    return 0;
}
```

Enter number of Nodes: 6
Enter adjacency matrix:

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 |

Topological order: 5 4 3 2 1 0



Lab-5

1) Selection Sort

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

void selsort(int n, int a[]){
        int i, j, t, small, pos;

        for(i=0; i<n-1; i++){
                pos=i;
                small= a[i];
                for(j=i+1; j<n; j++){
                        if (a[j] < small){
                                pos=j;
                        }
                }
                t= a[i];
                a[i] = a[pos];
                a[pos] = t;
        }
}

void main(){
        int a[15000], n, i, j, ch, temp;
        clock_t start, end;

        while(1){
                printf("\n1: For manual entry of N value
                        and array elements");
                printf("\n2: To display time taken for
                        sorting number of elements N in
                        range 500 to 14500");
```

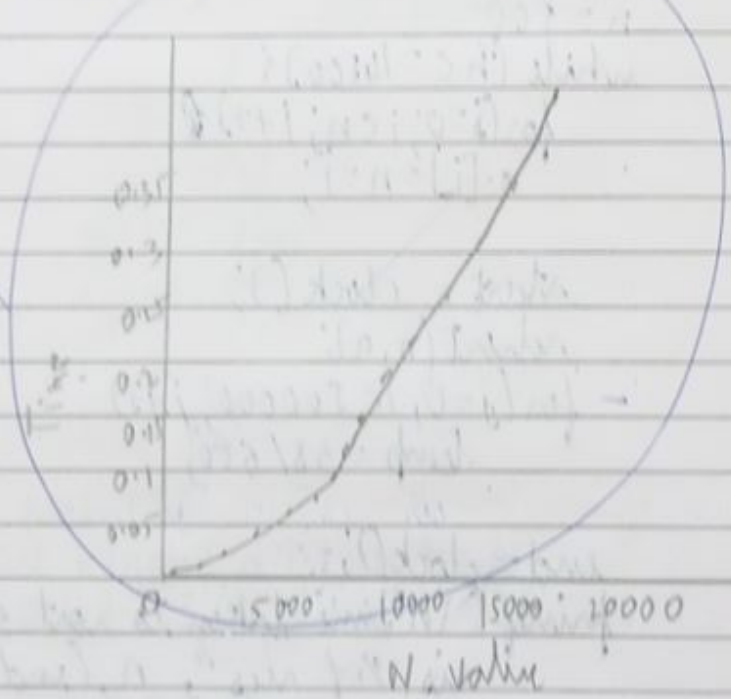
```
switch(ch){
    case 1:
    printf("\nEnter the number of element: ");
    scanf("%d", &n);
    printf("\n Enter array elements:");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);

    start = clock ();
    selsort (n,a);
    end = clock ();
    break;

  case 2:
    n=500;
    while (n<=14500){
        for(i=0;i<n;i++)
          a[i]=n-i;

        start = clock();
        selsort(n,a);
        for(j=0;j<500000;j++)
            temp=38/600;

        end=clock();
        printf("\n Time taken to sort %d numbers
                is %f secs", n,(end-start));
        n=n+1000;
    }
    break;
  case 3:
      exit(0);
}
getchar
```

Output:

1. For manual entry of N value and array elements
2. To display time taken for sorting number of N in rang 100 to 14500
3. Exit

Enter choice: 1
Enter number of elements: 8
Enter array numbers: 4 3 8 6 7 1 2 5
sorted array: 1 2 3 4 5 6 7 8
time taken to sort is 0.000000 secs.

Enter choice: 2




30/5/25

URBAN EDGE

```
#include <bits/stdc++.h>
#include <iostream.h>
#include <iostream.h>

void split(int [], int low, int high){

    int mid;
    if (low < high){
        mid = (low + high)/2;
        split(a, low, mid);
        split(a, mid+1, high);
        combine(a, low, mid, high);
    }
}


void combine(int a[], int low, int mid, int high){
    int c[10000], i, j, k;
    i = k = low;
    j = mid+1;

    while(i <= mid && j <= high){
        if (a[i] < a[j]){
            c[k] = a[i];
            ++k;
            ++i;
        }else {
            c[k] = a[j];
            ++k;
            ++j;
        }
    }
}
```

```
        if(i> mid){
            while (j<=high){
                c[k]= a[j];
                ++k;
                ++j;
            }
        }
        if (j > high){
            while (i<=mid){
                c[k]= a[i];
                ++k;
                ++i;
            }
        }
        for(i=low; i<=high;i++){
            a[i]= c[i];
        }

void main(){
    int a[15000], n,i,j, ch, temp;
    clock_t start, end;

    while(1){
        printf("\n1: For manual entry of N elements value
                and array elements");
        printf("\n2: To display time taken for
                sorting numbers");
        scanf("%d",&ch);
```

```
switch(ch){
    case 1;
        printf("\nEnter the number of elements:
        scanf("%d, &n);
        printf("\n Enter array elements:");

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

        start = clock();
        split(a, 0, n-1);
        end = clock();
        printf("\n Sorted array is:");

        for(i=0; i<n; i++)
            printf("%d\t", a[i]);

        break;

    case 2:
        n=500;
        while(n<=14500){
            for(i=0; i<n; i++){
                a[i]=n-i;

        start = clock();
        split(a, 0, n-1);

        for(j=0; j<50000; j++){
            temp = 38/500;

        n=n+1000;
    }
    break;
```

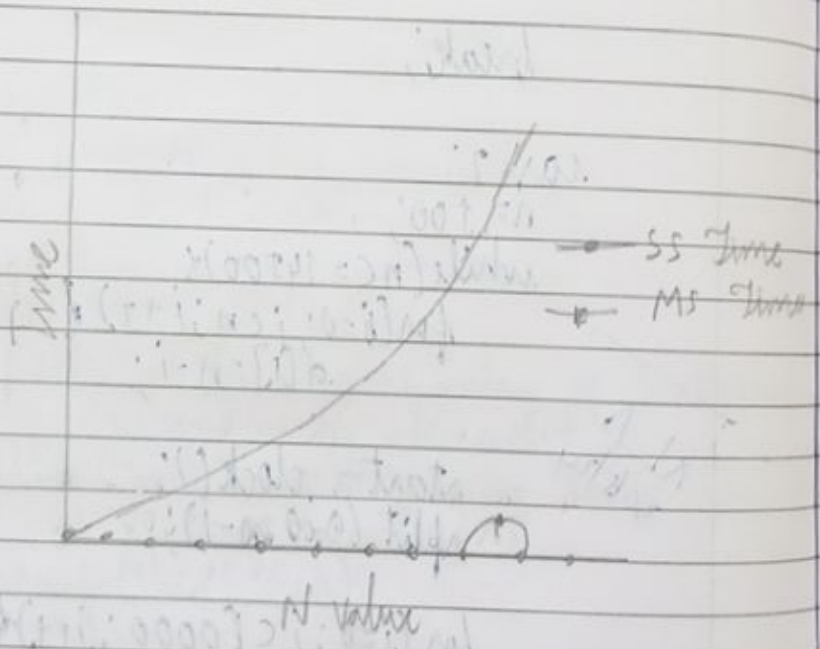
NLD
30/8/24

```
case 3:
    exit(0);
}
```

Output:

1: For manual entry of N value and array elements
2: To display Time taken for sorting from n=500 to 14500
3: To exit

Choice: 1
Enter the number of elements: 6
Enter array elements: 3 2 4 1 5 6
sorted array is: 1 2 3 4 5 6
Time taken to sort 6 numbers is 0.00000 sec



# Lab-6

URBAN EDGE 06.06.24

## Quick Sort

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

void swap(int* p1, int* p2){
        int temp;
        temp = *p1;
        *p1 = *p2;
        *p2 = temp;
}

int partion(int arr[], int low, int high){
        int pivot = 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);
}
```

N 6/6/24

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

int main(){
    int choice=0;

    while (choice <3){
        printf("\n1: For manual entry");
        printf("\n2: To display time taken");

        switch(choice){
        case 1:
            int n;
            printf("Enter n:");
            scanf("%d", &n);
            int arr[n];

            printf("Enter the elements of
                array");

            quickSort (arr, 0, n-1);

            printf("Sorted array: ");
            for (int i = 0; i<n; i++)
                printf("%d", arr[i]);

            break
```

case 2:

```
int n=500;
clock_t start,end;

while (n<=14500){
    int arr[n];
    for (int i=0;i<n;i++){
        arr[i]=n-1;
    }

    start=clock();
    quicksort(arr, 0,n-1);

    for(int j=0;j<500000;j++){
        temp=38/600;

    end=clock();
    printf("\n%d\t%f secs",n);
}
break;
```

Output:

1: For manual entry

2: For displaying time from 500 to 16500

choice: 1

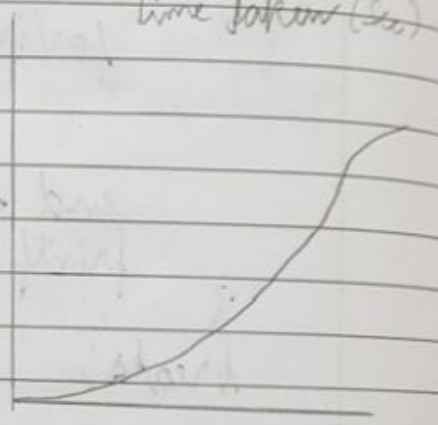
Enter n: 5

Enter the elements of array: 4 1 5 3 2

Sorted array: 1 2 3 4 5

choice: 2

| N | Time |
|---|---|
| 500 | 0.001959 |
| 1500 | 0.008572 |
| 2500 | 0.022145 |
| 3500 | 0.042460 |
| 4500 | 0.069289 |
| 5500 | 0.102797 |
| 6500 | 0.136150 |
| 7500 | 0.137964 |
| 8500 | 0.176664 |
| 9500 | 0.220885 |
| 10500 | 0.281747 |
| 11500 | 0.326804 |
| 12500 | 0.391993 |
| 13500 | 0.552570 |
| 14500 | 0.639614 |



ii) Binary Search:

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

void main(){
    int n, arr[10];
    int key;

    printf("Enter n:");

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

    printf("Enter the key to find:");
    scanf("%d", &key);

    while(low<=high){
            mid=(low-high)/2;

            if(arr[mid]==key)
                    printf("key found at %d", mid

            else if(arr[mid]>key)
                    high ~~mid~~ = ~~high~~ mid-1;

            else
                    low = mid+1;
```

# Lab-7

URBAN EDGE 13.06.24

1) Johnson Trotter:

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

int flag=0;

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

int search(int arr[], int num, int mobile){
        int g;
        for(g=0; g<num; g++){
                if(arr[g] == mobile)
                        return g+1;
                else
                        flag=1;
        }
        return -1;

int find_Mobile(int arr[], int d[], int num)
{
        int mobile=0;
        int mobile_p=0;
        int i;

        for(i=0; i<num; i++){
```

```
        if((d[arr[i
                if(arr
                {

                }else
                        fl
        }else if(
                if(ar

                }else
                        fl
        for(i=0; i
                prin
        }
}

int factorial(int
        int f=1;
        int i=0;
        for(i=1; i<
                f=f
        return f;
}
```

N 13/6/24

```
if((d[arr[i]-1]==0) && i!=0){
    if(arr[i]>arr[i-1] && arr[i]>mobilep)
    {
        mobile=arr[i];
        mobilep=mobile;
    }else{
        flag++;

}else if((d[arr[i]-1]==1) && i!=num-1){
    if(arr[i]>a[i+1] && arr[i]>mobilep){
        mobile=arr[i];
        mobilep=mobile;
    }else
        flag++;

for(i=0;i<num;i++)
    printf("%d", arr[i]);
}
}

int factorial(int k){
    int f=1;
    int i=0;
    for(i=1;i<k+1;i++)
        f=f*i;
    return f;
}
```

12/6/24

```
int main(){
    int num=0,i,j,z=0;
    scanf("%d",&num);

    int arr[num],d[num];

    for(i=0;i<num;i++){
            d[i]=0;
            arr[i]=i+1;
            printf("%d",arr[i]);
    }
    printf("\n");

    for(j=1;j<z;j++){
        permutations(arr,d,num);
        print("\n");
    }
    return 0;
}
```

Output:

```
Johnson Trotter algorithm:
Enter the Number:3
total permutations=6
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3
```

Trotter

ii) Pattern Matching (Brute Force)

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

int main(){
    char str[20], substr[10];

    printf("Enter string:");
    scanf("%s", str);

    printf("Enter substring:");
    scanf("%s", substr);

    int mlen=strlen(str), slen=strlen(substr);
    int m_ind=0, flag=0;

    while(m_ind<mlen){
        if(str[m_ind]==substr[0]){
            for(int i=0;i<slen;i++){
                if(str[m_ind+i]!=substr[i]){
                    flag=1;
                    break;
                }
            }
            if(flag==0){
                printf("Match found at position
                    %d", m_ind);
                exit(0);
            }
        }
        m_ind++;
    }
```

URBAN EDGE

Output:

Enter the main string : ~~ab~~ abcd
Enter the substring : cd
Match found at position 2.

URBAN EDGE 20.06.24

# Lab-8

## i) Heap Sort:

```
#include<stdio.h>

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

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

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

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

    if (largest != i){
        swap(&arr[i], &arr[largest]);
        heapify(arr, N, largest);

}
```

```
void heapsort (int arr[], int N){
        for(int i=n/2-1; i>=0;i--)
                heapif(arr, n, i);

        for (int i=n-1;  i>=0;i--){
                swap(&arr[0], &arr[i]);
                heapify(arr, i, 0);
        }

int main(){
        int arr[]={12,11,13,5,6,7};
        int N= sizeof(arr)/sizeof(arr[0]);
        heapsort(arr, N);
        printf("Sorted Array is :\n");
        printArray(arr, N);
}
```

Output

Sorted array is : 3 8 9 16 32 46

golden

ii) Floyd's Algorithm

```
#include <stdio.h>
#define V 4
#define INF 99999

void printSolution(int dist[][V]);

void floydWarshal(int dist[][V])
{
    int i,j,k;

    for(k=0; k<V; k++){
        for(i=0; i<V; i++){
            for(j=0; j<V; j++){

                if(dist[i][k] + dist[k][j] <dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j]
            }
        }
    }
    printSolution(dist);
}

int main()
{
    int graph[V][V]={{0, 3, INF, 1},
                     {INF, 1, 6, 2},
                     {INF, INF, 0, 1},
                     {INF, INF, INF, 0}};
    floydWarshall(graph);
    return 0;
}
```

Output:
The shortest distance is :

| | | | |
|---|---|---|---|
| 0 | 3 | 9 | 1 |
| INF | 0 | 6 | 1 |
| INF | INF | 0 | 1 |
| INF | INF | INF | 0 |

30/10/24

URBAN EDGE 4.7.24

# Lab-9

1) Knapsack:

```
#include <stdio.h>

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

void knapsack(int n, int W, int weights[], int values[])
{
    int i, w;
    int dp[n+1].[W+1];

    for (i=0; i<=n; i++){
        for(w=0; w<=W; w++){
            if (i==0||w==0){
                dp[i][w]=0;
            } else if (weights[i-1]<=w){
                dp[i][w]= max(dp[i-1][w], dp[i-1]
                                [w-weights[i-1]]
                                + values[i-1]);
            } else {
                dp[i][w]=dp[i-1][w];
            }
        }
    }

    printf("DP table:\n");
    for(i=0; i<=n; i++){
        for(w=0; w<=W; w++){
            printf("%d\t",dp[i][w]);
        }
        printf("\n");
    }
```

4/7/24

```
printf("Selected items: ");
int res = dp[n][W];
w = W;

for(i=n; i>0 && res>0; i--){
        if(res==dp[i-1][w])
            continue;
        else{
            printf("%d ", i);
            res = res - values[i-1];
            w = w-weights[i-1];
        }
}

printf("\nMaximum profit: %d\n", dp[n][W]);

int main(){
    int n=6, weights[]={2,3,4,5};
    int values[]={3,4,5,8};
    int W=5;

    knapsack(n, W, weights, values);
    return 0;
}
```

Output:

DP table:

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 3 | 3 | 3 | 3 |
| 0 | 0 | 3 | 4 | 4 | 7 |
| 0 | 0 | 3 | 4 | 4 | 7 |
| 0 | 0 | 3 | 4 | 5 | 7 |
| 0 | 0 | 3 | 4 | 5 | 8 |

selected items: 6

Maximum profit: 8

ii) Prim's Algorithm:

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

#define max 5

int minkey(int n, int key[],int mst[set[]){

    int min = INT_MAX, min_index;

    for(int v=0; v<n ; v++){
        if (mstset[v]==0 && key [v] < min){
            min = key[v];
            min_index = v;
        }
    }

    return min_index;
}

void printMST(int n, int parent [], int cost[n][max])
{
    int num=0;

    printf ("Edges in MST:\n");
    for(int i =1; i <n; i++){
        printf("%d-%d\n", parent [i], i);
        num + = cost [i][parent [i]];
    }
    printf ("Cost of MST is :%d\n", num);
}
```

NL 4/7/24

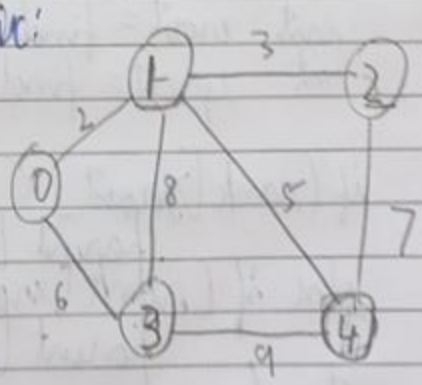
```
void primeMST(int n, int cost[max][max]){
    int parent[max];
    int key[max];
    int mstSet[max];

    for(int i=0; i<n; i++){
        key[i]=INT_MAX;
        mstSet[i]=0;
    }

    key[0]=0;
    parent[0]=-1;

    for(int count=0; count<n-1; count++){

        int v= minKey(n, key, mstSet);
        mstSet[v]=1;

        for(int v=0; v<n; v++){
            if (cost[v][v] && mstSet[v]==0 &&
                cost[v][v] < key[v]){
                    parent[v] = v;
                    key[v] = cost[v][v];
            }
        }
    }
    printMST(n, parent, cost);
}
```

```
int main(){
    int n = max;
    int cost[max][max] = {
        {0, 2, 0, 6, 0},
        {2, 0, 3, 8, 5},
        {0, 3, 0, 0, 7},
        {6, 8, 0, 0, 9},
        {0, 5, 7, 9, 0}
    };

    printf("Cost adjacency matrix:\n");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            printf("%d ", cost[i][j]);
        }
        printf("\n");
    }

    primeMST(n, cost);
    return 0;
}
```

Output:

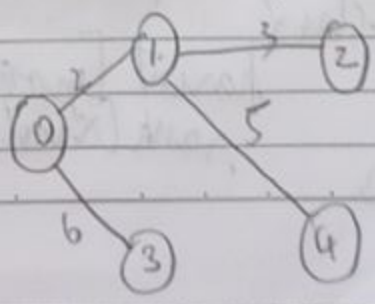
Cost adjacency matrix:

```
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0
```




MST:

```
0 - 1
1 - 2
0 - 3
1 - 4
```



# Lab 10

1) Dijkstra's and Kruskal algorithm:

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

#define MAX 100
#define INF 9999

typedef struct edge{
        int src, dest, weight;
}edge;

int find(int parent[], int i){
        if(parent[i]==-1)
                return i;
        return find(parent, parent[i]);
}

void union(int parent[], int rank[], int x, int y)
{
        int xroot= find(parent, x);
        int yroot= find(parent, y);

        if(rank[xroot] < rank[yroot])
                parent[xroot] = yroot;
        else if(rank[xroot] > rank[yroot])
                parent[yroot] = xroot;
        else{
                parent[yroot]= xroot;
                rank[xroot]++;
        }
}
```

URBAN EDGE

```
int compareEdges(void *a, void *b){
    Edge *edgeA=(Edge*)a;
    Edge *edgeB=(Edge*)b;
    return edgeA->weight - edgeB->weight;
}

void kruskalMST(int graph[MAX][MAX], int numVertices)
{
    int E=0;
    Edge edges[MAX*MAX];

    for(int i=0; i<numVertices; i++){
      for(int j=i+1; j<numVertices; j++){
        if(graph[i][j]!=0 && graph[i][j]!=INF)
        {
          edge[E].src=i;
          edges[E].dest=j;
          edges[E].weight=graph[i][j];
          E++;
        }
      }
    }

    qsort(edges, E, sizeof(edges[0]), compareEdges);
    int parent[numVertices];
    int rank[numVertices];
    for(int v=0; v<numVertices; v++){
            parent[v]=v;
            rank[v]=0;
    }
    Edge result[numVertices];
    int e=0, i=0;
}
```

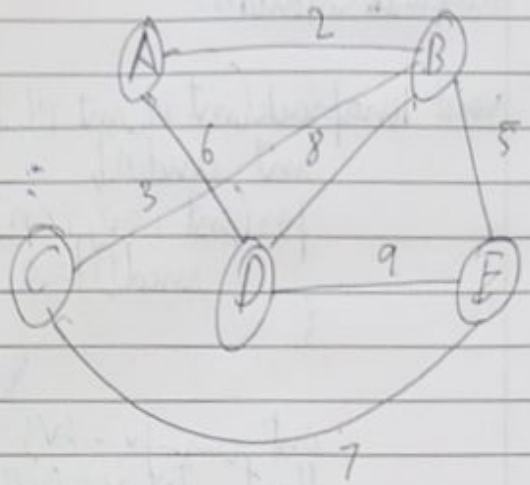
```
void dijkstra(int graph[MAX][MAX], int src, int numVertices)
        int dist[numVertices];
        bool sptSet[numVertices];

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


int main(){
    int numVertices=5;
    int graph[MAX][MAX]={
            {0, 2, INF, 6, INF},
            {2, 0, 3, 8, 5},
            {INF, 3, 0, INF, 7},
            {6, 8, INF, 0, 9},
            {INF, 5, 7, 9, 0}
        };

    kruskalMST(graph, numVertices);
    dijkstra(graph, 0, numVertices);
    return 0;
}
```

Output:

Kruskal's MST:

Edge in MST

0 - - 1 = 2
1 - - 2 = 3
1 - - 4 = 4
0 - - 3 = 6




Dijkstra's shortest path

| Vertex | Distance from source |
|---|---|
| 0 | 0 |
| 1 | 2 |
| 2 | 5 |
| 3 | 8 |
| 4 | 7 |

10/9/24

ii) Greedy Knapsack using greedy algorithm:

```
#include <stdio.h>

void knapsack(int n, int p[], int w[], int W){
        int used[n];
        for(int i=0; i<n; ++i){
                used[i]=0;
        }


        int cur_w = W;
        float tot_v = 0.0;
        int i, maxi;

        while(cur_w >0){
                maxi = -1;
                for(i=0; i<n; ++i){
                        if((used[i] ==0)&&((maxi==-1)||((float)p[i]/w[i]
                        {

                                maxi = i;
                        }
                }
                if(maxi==-1)break;

                used[maxi] = 1;

                if(w[maxi] <= cur_w){
                        cur_w -= w[maxi];
                        tot_v += p[maxi]
                }
                else{
                        int taken = cur_w;
                        cur_w = 0;
                        tot_v += (float)taken / w[maxi] * p[maxi];
                }
        }
}
```

```
int main(){
    int n, W;

    printf("Enter the number of objects:");
    scanf("%d", &n);

    int p[n], w[n];

    printf("Enter the profits of the objects:");
    for(int i=0; i<n; i++){
        scanf("%d", &p[i]);

    knapsack(n, p, w, W);

    return 0;
}
```

16/12/24

Output:

```
Enter the number of obj: 4
Enter the profits of obj: 20 30 60 40
Enter the weights of obj: 10 20 30 40
Enter the maximum weight of the bag: 50

Added object 3 completely in the bag. Space left
20.
Added object 1 completely in the bag. Space left
```