INDEX

NAME: S Rakhal STD.: _____ SEC.: CD ROLL NO.: _____ SUB.: A.D.A Lab.

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
	02/05/24	find dissapeared number	1	
	09/05/24	Zig Zag level order traversal	3	
	16-05-24	Increasing order Search tree.	5	
	23-05-24	Topological sort	7	
	30-05-24	Selection and Mergesort time comparison	12	
	06-06-24	Quick sort time complexity analysis.		
	13-06-24	Johnson Trotten method, Brute force string matching, Kth largest number		
	20-06-24	heap sort and floyds algorithm.		4/7/24
	04/07/24	Knapsack using D.P prims Algorithm.		
	11/07/2024	Kruskals algorithm dijkstras algorithm	—	11/7

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks

Leetcode problems

1) Find all Numbers disappeared in an array

code:-

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

void swapArray(int* arr, int first, int second){
    int temp = arr[first];
    arr[first] = arr[second];
    arr[second] = temp;
}

void cyclicSort(int* arr, int size){
    int i=0;
    while(i<size){
        if(arr[i] != arr[arr[i]-1] && arr[i]<=size){
            swapArray(arr, i, arr[i]-1);
        }
        else{
            i++;
        }
    }
}

int* findDisappearedNumbers(int* nums, int numsSize, int* returnSize){
    int* result = (int*)malloc(sizeof(int)*numsSize);
    *returnSize = 0;
    cyclicSort(nums, numsSize);
    for(int i=0; i<numsSize; i++){
        if(nums[i] != i+1){
            result[(*returnSize)++] = i+1;
        }
    }
```

```
    return result;
}
```

9/5/24

Test cases:

case 1:- I/P:-

nums = [4, 3, 2, 7, 8, 2, 3, 1]

O/P:-

[5, 6]

case 2:- I/P:-

nums = [1, 1]

O/P:-

[2]

2nd Lab.

Binary tree Zigzag level order traversal

```
int ** zigzagLevelOrder(struct TreeNode* root, int*
returnSize, int** columnSizes){
        if(root==NULL)
            *returnSize=0;
            *columnSizes=NULL;
             return NULL;
        }
int** res=(int**)malloc(sizeof(int*)*2048);
*columnSizes=(int*)malloc(sizeof(int)*2048);
*returnSize=0;

struct TreeNode** queue=(struct TreeNode**)malloc
(sizeof(struct TreeNode*)*2048);
int front=0, rear=0;

queue[rear++]=root;

bool zigzag=false;
while(front!=rear){
        int size=rear-front;
        res[*returnSize]=(int*)malloc(sizeof(int)*size);
        (*columnSizes)[*returnSize]=size;

    for(int i=0;i<size;i++){
        struct TreeNode* node=queue[front++];
        if(zigzag){
            res[*returnSize][size-i-1]=node->val;
```

```
if (node->left){
    queue[rear++] = node->left;
}
if (node->right){
    queue[rear++] = node->right;
}
}
(*returnSize)++;
zigzag = !zigzag;
}
free(queue);
return res;
}
```

9/5/24

Test cases:-

case 1:- o/p:-

root = [3, 9, 20, null, null, 15, 7]

output:-

[[3], [20, 9], [15, 7]]

case 2:- o/p:-

root = [1]

o/p:-

[[1]]

case 3:- o/p:-

root = []

o/p:-

[]

Increasing order traversal

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

void dfs (struct TreeNode * node, int * list, int * size){
    if (node == NULL){
        return;
    }
    list [*size] = node -> val;
    (*size)++;
    dfs (node -> left, list, size);
    dfs (node -> right, list, size);
}

struct TreeNode * increasingBST (struct TreeNode *root)
{
    int * list = (int *) malloc (sizeof (int) * 1000);
    int size = 0;
    dfs (root, list, &size);

    for(int i=0; i< size-1; i++){
        for(int j=0; j< size-1; j++){
            if( list[j] > list[j+1]){
                int temp = list[j]
                list[j] = list[j+1];
                list[j+1] = temp;
            }
        }
    }

struct Treenode *res = (struct Treenode*) malloc (
sizeof (struct Treenode ));
res -> val = list[0];
res -> left = NULL;
res -> right = NULL;
```

```
struct TreeNode *temp = res;
for(int j = 1; j < size; j++){
        struct TreeNode *node = (struct TreeNode*)
malloc(sizeof(struct TreeNode));
    node->val = list[j]
    node->left = NULL;
    node->right = NULL;
    temp->right = node;
    temp = node;
}
free(list);
return res;
}
```

testcases

16/5/24

case1:

I:P/

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

O/P:-

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

I/P/

root = [5, 1, 7]

O/P:-

[1, null, 5, null, 7]

Topological Sorting using Source removal algorithm.

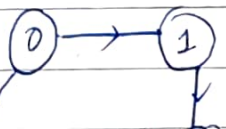
C-code:-

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

#define MAX_VERTICES 100

struct Graph {
    int numVertices;
    int** adjMatrix;
};

struct Graph* createGraph(int numVertices){
    struct Graph* graph = (struct Graph*)
malloc(sizeof(struct Graph));
    graph->numVertices = numVertices;
    graph->adjMatrix = (int**)malloc(numVertices
*sizeof(int*));
    for(i=0; i<numVertices; i++){
        graph->adjMatrix[i] =(int*)calloc(numVertices,
sizeof(int));
    }
    return graph;
}

void addEdge(struct Graph* graph, int src, int dest){
    graph->adjMatrix[src][dest]=1;

}
```

```
void topologicalSort ( struct Graph * graph, int
sortedVertices [] ) {
        int sources [MAX_VERTICES];
        int numSources = 0;

        for (int i = 0; i < graph -> numVertices ; ++i) {
            int isSource = 1;
            for ( int j = 0; j < graph -> numVertices
            {
                if ( graph -> adjMatrix [j][i] == 1 {
                    is source = 0;
                    break;
                }
            }
            if (isSource ) {
                sources [numSources ++] = i;
            }
        }
    int index = 0;
    while (numSources > 0) {
        int Source = sources [--numSources];
        sortedVertices [index++] = source;

    for ( int i = 0; i < graph -> numVertices; ++i)
        if (graph -> adjMatrix [source][i]
            graph -> adjMatrix [ source][i]
            int is Source = 1;
            for (int j = 0; j -> numVertices
            {
                if (graph -> adjMatrix [j][i]
```

```
        if (isSource) {
            sources[numSources++] = i;
        }
    }
}
}

int main() {
    struct Graph* graph = createGraph(4);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 2, 3);

    int sortedVertices[4];
    topologicalSort(graph, sortedVertices);

    printf("Topological Sorting: ");
    for (int i = 0; i < 4; ++i) {
        printf("%d ", sortedVertices[i]);
    }
    printf("\n");
    return 0;
}
```

Output:
Topological sorting:- 0 2 1 3.



2) DFS

C- Implementation

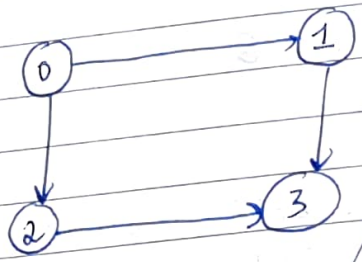
```
#include <stdio.h>
#include <stdlib.h>

#define MAX_VERTICES 100.

struct Graph {
        int numVertices;
        int ** adjMatrix;
};

struct Graph* createGraph(int numVertices){
        struct Graph* graph = (struct Graph*)
malloc (sizeof(struct Graph));
        graph -> numVertices = numVertices;

        graph-> adjMatrix = (int **) malloc (numV
        * sizeof(int *));
        for(int i=0; i<numVertices ; ++i) {
                graph-> adjMatrix[i] = (int *) calloc (num
    -ces , sizeof(int));
        }
        return graph;
}

void addEdge (struct Graph * graph , int src, int
        graph -> adjMatrix [src] [dest ] = 1;
}
```

```
    visited[vertex] = 1;

    for (int i=0; i<graph->numVertices; ++i){
        if (graph->adjMatrix[vertex][i] && !
        visited[i]) {
            DFS(graph, i, visited, index, sortedVertic
        }
    }
    sortedVertices[--(*index)] = vertex;
}

void topologicalSort(struct Graph* graph){
    int visited[MAX_VERTICES] = {0};
    int sortedVertices[MAX_VERTICES];
    int index = graph->numVertices;

    for (int i=0; i<graph->numVertices; ++i){
        if(!visited[i]){
            DFS(graph, i, visited, &index,
        sortedVertices);
        }
    }
    printf("Topological sorting: ");
    for (int i=0; i<graph->numVertices; ++i){
        printf("%d", sortedVertices[i]);
    }
    printf("\n");
}

int main(){
    struct Graph* graph = CreateGraph(4);
```

```
    topologicalSort (graph);
    return 0;
}
```

OUTPUT;

Topological Sorting: 0 2 1 3



6/6/24

Comparing time taken to compile sorting in selection sort and merge sort

1) selection sort.

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

void selsort (int n, int a[]);

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

while(1)

{
 printf("\n1: for manual entry of n values
and array elements");
 printf("\n2: To display time taken for sorting
number of elements in the range of 500 to 14500");
 printf("\n3: To exit);
    printf("\n: Enter your choice:");
    scanf("%d", &ch);
    switch(ch)
    {
     case 1: printf("\n Enter the number of elements");
             scanf("%d", &n);
             printf("\n Enter the number of
             elements");
             for(i=0; i<n; i++){
                 scanf("%d", &a[i]);
             }
             start = clock();
```

```
selsort(n,a);
end = clock();
printf("\n Sorted array is: ");
for(j=0; i<n; i++){
    printf("%d", a[i]);
}
printf("\n Time taken to sort is %f Secs", n,
    (((double)(end-start))/CLOCKS_PER_SECs));
break;
case2:
        n=500;
        while (n<= 14500){
        for (j=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 %f
Secs", n, ((double)(end-start))/CLOCKS_PER_SEC
            n=n+1000;
            }
        break;
    case 3: exit(0);
}
getchar();
 }
}
```

P.T.O

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

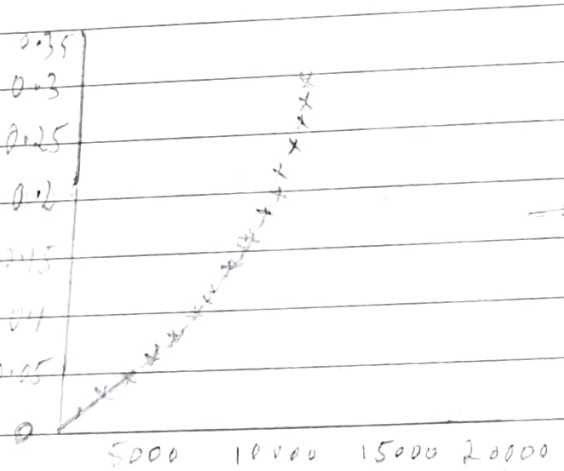
OUTPUT:-

1) For manual entry of N value and array elements
2) To display time taken for sorting number of elements N in the range 10000 to 85000
3) To exit
Enter your choice: 1
Enter the number of array elements: 4
Enter array elements: 44 33 22 11
Sorted array is: 11 22 33 44
Time taken to sort 4 numbers is 0.000001 secs
1) For manual entry of N values and array elements
2) To display time taken for sorting number of elements N in the range 500 to 145000
3) To Exit
Enter your choice: 2

Time taken to sort 500 numbers is 0.003295 Sec
Time taken to sort 1500 numbers is 0.010222 Sec
Time taken to sort 2500 numbers is 0.018430 Sec
Time taken to sort 3500 numbers is 0.029586 Sec
Time taken to sort 4500 numbers is 0.040886 Sec
Time taken to sort 5500 numbers is 0.051843 Sec
Time taken to sort 6500 numbers is 0.063836 Sec
Time taken to sort 7500 numbers is 0.082561 Sec
Time taken to sort 8500 numbers is 0.105688 Sec
Time taken to sort 9500 numbers is 0.131803 Sec
Time taken to sort 10500 numbers is 0.158 110 4 Sec
Time taken to sort 11500 numbers is 0.193703 Sec
Time taken to sort 12500 numbers is 0.227940 Sec
Time taken to sort 13500 numbers is 0.265653 Sec
Time taken to sort 14500 numbers is 0.305670 Sec

Enter your choice: 3

Selection Sort



AIM: Sort a given set of N integer elements using Merge sort technique and compute its time taken.

code :-

```
#include <stdio.h>
#include<time.h>
#include<stdlib.h>
void split(int[], int, int);
void combine(int[], int, int, int);
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
element");
printf("\n2: For displaying time taken for sorting
number of elements N in the range of 500 to 14500");
printf("\n3: To Exit");
    printf("\n Enter your choice:");
    scanf("%d", &ch);
    switch(ch)
    {
      case 1: printf("\n Enter 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 (j=0; i<n; i++)
        printf("%d", a[i]);
printf("\n Time taken to sort %d numbers is %f
Secs ", n, ((double)(end-start))/clocks_PER_SEC);
break;

case 2:
        n=500.
        while (n<=14500){
         for (j=0; i<n; i++)
         {
            a[i]=n-i;
         }
        start = clock();
        split(a,0,n-1);
        //Dummy loop to create delay
        for (j=0; j< 500000 ; j++) { temp= 38/600;}
        end = clock();
printf("\n Time taken to sort %d numbers is %f
Secs ", n, ((double)(end-start))// CLOCKS_PER_SECOND);
        n=n+1000;
        }
      break;

case3: exit(0);
 }
getchar();
}

}
```

```
void dplit ( int a[], low, int high )
{
  int mid;
  if (low < high)
  {
  mid = (low + high)/2;
   dplit (a, low, mid);
   dplit (a, mid+1, high);
    combine (a, low, mid, high);
    }
}

void combine (int a[], int low, int mid, int high)
{
int c [15000], 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];
  }
}
```

OUTPUT:

1) For manual entry of N value and array elements
2) To display time taken for sorting number of elements N
3) the range 500 to 14500.
3) To exit

Enter your choice: 1
Enter the number of elements: 4
Enter the elements: 44 33 22 11
sorted array is: 11 22 33 44
Time taken to sort 4 numbers is 0.000012 secs
1) For manual entry of N value and array elements
2) To display time taken for sorting number of elements

N in the range 500 to 14500

3) To Exit.

Enter your choice: 2

Time taken to sort 500 numbers is 0.002698 secs.

Time taken to sort 1500 numbers is 0.002907 secs

Time taken for sort 2500 number is 0.003055 secs.

Time taken to sort 3500 numbers is 0.003391 secs.

Time taken to sort 4500 numbers is 0.003037 secs

Time taken to sort 5500 numbers is 0.002826 secs

Time taken to sort 6500 numbers is 0.003703 secs

Time taken to sort 7500 numbers is 0.003794 secs

Time taken to sort 8500 numbers is 0.003021 secs

Time taken to sort 9500 numbers is 0.003072 secs

Time taken to sort 10500 numbers is 0.003144 secs

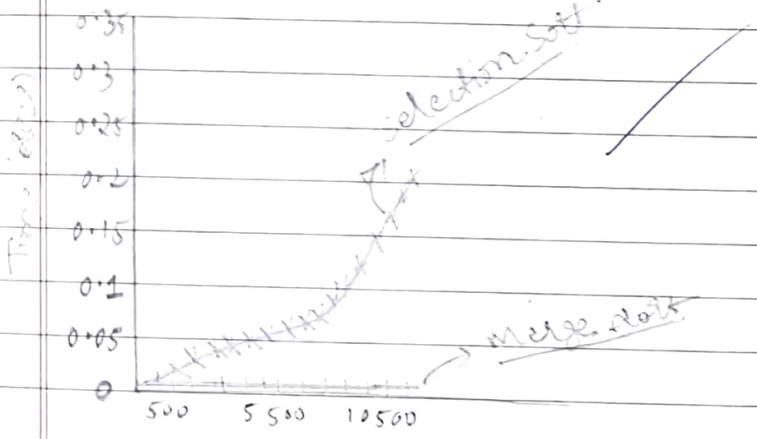
Time taken to sort 11500 numbers is 0.003546 secs

Time taken to sort 12500 numbers is 0.003321 secs

Time taken to sort 13500 numbers is 0.003507 secs

Time taken to sort 14500 numbers is 0.003624 secs

Enter the choice: 3



Quick sort

c-code :-

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

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

int partition (int array[], int low, int high) {

    int pivot = array[high];
    int i = (low - 1);

    for (int j = low; j < high; j++) {
        if (array[j] <= pivot) {
        i++;
        swap (&array[i], &array[j]);
        }
    }
swap (&array[i+1], &array[high]);

return (i+1);
}

void quicksort (int array[], int low, int high) {
    if (low < high) {
    int pi = partition (array, low, high);
    quick sort (array, low, pi - 1);
```

```
        quickSort(array, pi+1, high);
    }
}

void printArray(int array[], int size){
    for (int i=0; i< size; ++i){
        printf("%d", array[i]);
    }
    printf("\n");
}

void main(){

    int a[15000], n, i, ch, j, temp;
    clock_t start, end;

while(1)
    printf("Enter choice");
    scanf("%d", &n);
    printf("\n Enter array elements");
    for(i=0; i<n;i++){
        scanf("%d", &a[i]);
    }
    start = clock();
    quicksort(a, 0, n-1);
    end = clock();
    printf("Sorted array is: ");
    printArray(a, n);
    printf("Time taken to sort %d numbers is
    %f secs", n, ((double)(end-start))/CLOCKS_PE
    -R -SEC));
    break;
case 2:
```

```
n=500;
while(n<=14500){
        for(j=0; i<n; i++){
            a[i]=n-i;
        }
}
start=clock();
quicksort(a,0,n-1);
for(j=0; j<500000; j++){temp=88/600;}
end=clock();
printf("\n Enter the time taken to sort %d.d
is %.f secs", n, ((double)(end-start))/CLOCK.
PER_SEC));
n=n+1000;
}
break;
case 3:
    exit(0);
}
getchar();
}
}
```

O/P:-

enter choice: 1.

Enter number of elements: 11 22 33

the sorted array is: 11 22 33

time taken to sort is 0.000 secs

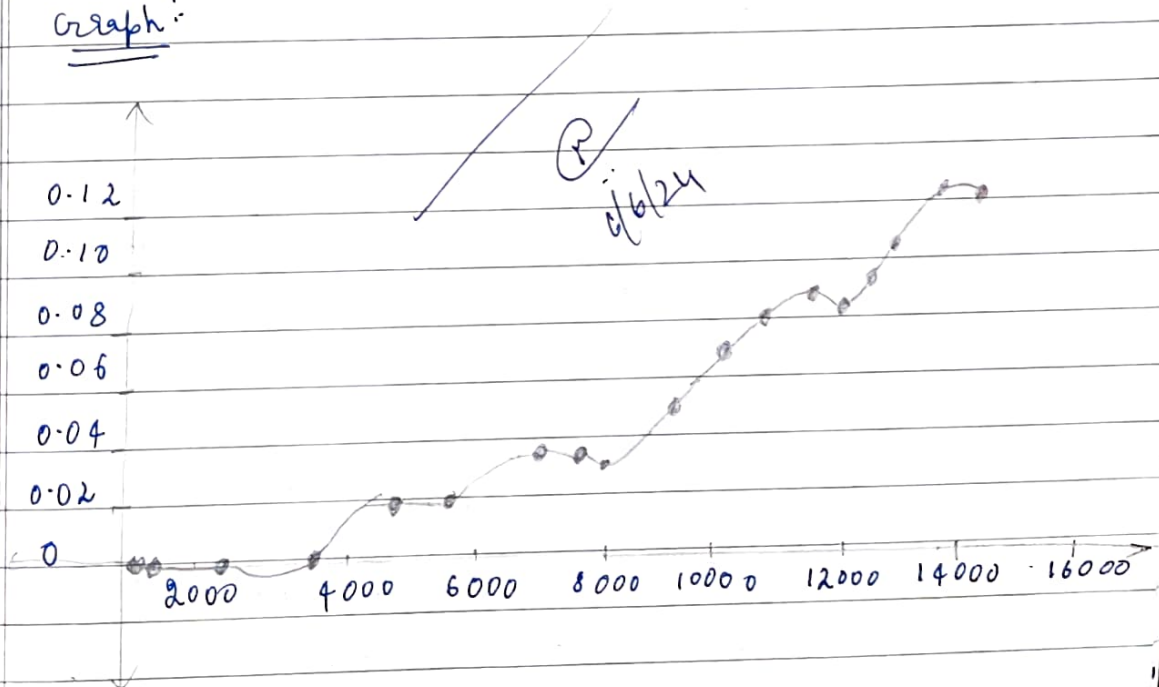
P.T.O

Enter choic: 2

time taken to sort 500 numbers is 0.000 secs
1500 numbers is 0.000 secs
2500 numbers is 0.000 secs
3500 numbers is 0.000 secs
4500 numbers is 0.015 secs
5500 0.016 secs
6500 0.031 secs
7500 0.031 secs
8500 0.031 secs
9500 0.047 secs
10500 0.063 secs
11500 0.078 secs
12500 0.078 secs
13500 0.109 secs
14500 0.110 secs

Enter your choice: 3

Graph:



Johnson Trotter algorithm.

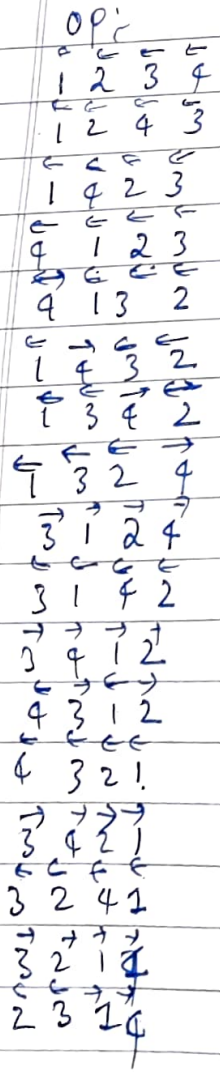
code - c

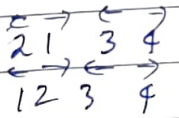
```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

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

void printpermutation(int *a, int *b)
{
  for(int i=0; i<n; i++){
      printf("%d", a[i]);
    }
    printf("\n");
}

int findLargestMobile(int *a, int *dir, int n){
    int largestMobile = -1;
    int largestMobileIndex = -1;

    for(int i=0; i<n; i++){
        if(dir[a[i] - 1] == 0 && i != 0 && a[i] > a[i-1]
&& a[i] > largestMobile){
            largestMobile = a[i];
            largestMobileIndex = i;
        } else if(dir[a[i] - 1] == 1 && i != n-1 &&
a[i] > a[i+1] && a[i] > largestMobile){
            largestMobile = a[i];
            largestMobileIndex = i;
```

```
    }
}
return largestMobileIndex;
}

void johnsonTrotter(int n){
    int* a = (int*)malloc(n * sizeof(int));
    int *dir = (int* )malloc(n* sizeof(int));

    for(int i = 0; i<n; i++){
        a[i] = i+1;
        dir[i] = 0;
    }
    printPermutation(a, n);

    while(1){
        int largestMobileIndex = findlargest
Mobile(a, dir, n);
        if(largestMobileIndex == -1) break;
        int swapIndex = largestMobileIndex +
(dir[a[largestMobileIndex == -1)
        i == 0? -1:1);
        swap(&a[largestMobileIndex], &a
[swapIndex]);

        swap(&dir[a[largestMobileIndex]- &dir]
[swapIndex]], &dir[a[swapIndex]]);

        swap(&dir[a[largestMobileIndex]
& a[

        for(int i=0; i<n; i++)
        {
            if(a[i] > a[swapIndex])
            {
                dir[a[i]] > a[swapIndex]
```

```
cdir[a[i]-1] = !dir[a[i]-1];
        }
    }
    printPermutation(a, n);
}
free(a);
free(dir);
}
int main(){
int n;
printf("Enter the value of n: ");
scanf("%d", &n);
johnsonTrotter(n);
return 0;
}
```

Op:





classmate
Date 13/06/24
Page

Brute force string matching

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

void match(char *text, char *pattern) {
    int n = strlen(text);
    int m = strlen(pattern);
    for (int i = 0; i <= n-m; i++) {
        int j;
        for(j = 0; j < m; j++) {
            if (text[i+j] != pattern[j]) {
                break;
            }
        }
        if (j == m) { Matched
            printf("pattern found");
        }
    }
}

int main() {

    char text[] = "Hello Bye";
    char pattern = "Hell";
    match(text, pattern);
```

① pic

pattern matched

Leetcode

Output

```
int comp(const void *a, const void *b){
    const char *str1 = *(const char **)a;
    const char *str2 = *(const char **)b;

    if(strlen(str1) == strlen(str2)){
        return strcmp(str1, str2);
    }
    return strlen(str1) - strlen(str2);
}
char* kthLargestNumber(char** nums, int numsSize, int k){
    qsort(nums, numsSize, sizeof(char*), comp);
    return nums[numsSize-k];
}
```

Op:-

Test case 1:

Input : nums ["3", "6", "7", "10"]

K = 4

Op: 3

case 2

Input: nums = ["2", "21", "12", "1"]

k = 3

Op: 2.

Heap sort using C:

```
#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--)
        heapify (arr, n, i);

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

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

int main(){

int a[15000], n, i, j, ch, temp;
clock_t start, end

while(1){
printf("\n 1: For manual entry of N value and array elements");
printf("\n 2: For manual display time taken for sorting number of elements N in the range 500 to 14500");
printf("\n 3: Enter exit");
    printf("\n Enter your choice");
    scanf("%d", &n); switch(ch)
{case 1: printf("\n Enter array elements");
    for(i=0; i<n; i++)
    {
     scanf("%d", &a[i]);

    }
    start = clock();
    heapsort(a, n);
    end = clock();
    printf("\n Sorted array is:");
    for(j=0; i<n; i++){
    printf("%d", a[i]);
    printf("\n Time taken do sort %d numbers is %f Secs", n, ((double)(end-start))/CLOCKS_PER_SEC);
        break;
case 2:
```

```
n=500;
while (n<=14500) {
  for (i=0 ; i<n ; i++)
    {
      a[i] = random (100
      a[i] = n-i
    }
    start = clock ();
    heapSort (a, n);
    //Dummy loop to create delay
     for( j=0; j< 500000; j++) { temp = 38/600;}
     end = clock ();
     printf( "\n Time taken to sort %d numbers
is %f Secs ", n, (((double ) (end - start))/CLOCKS_PER
_SEC));
        n = n+1000;
      }
    break;
case 3: exit(0);
 }
getchar ();
}
}
```

Output :-

1) To enter data into the array manually.
2) To display time taken for sorting number of n elements n in the range 500 to 14500
3) To exit.

Enter your choice 1.
Enter the number of elements : 3.
Enter array elements : 33

sorted array is : 11 22 33
Time taken to sort 3 numbers is 0.000002 Secs.

Enter your choice :- 2.

Time taken to sort 500 numbers is 0.000557 secs
Time taken to sort 1500 numbers is 0.000538 secs
Time taken to sort 2500 number is 0.000621 secs
Time taken: to sort 3500 numbers is 0.000784 secs.
Time taken to sort 4500 numbers is 0.000926 secs
Time taken to sort 5500 numbers is 0.001070 secs
Enter your choice: 3

Floyd's algorithm for shortest path.

code :

```
#include <stdio.h>

#define nV 4

#define INF 999

void printMatrix (int matrix [] [nV]);
{
  for (int i=0; i<nV ; i++){
        for (int j=0 ; j<nV ; j++) {
            if (matrix [i][j] ==INF)
                printf ("%-4s" "INF");

                -( " %-4d", matrix [i][j]);

                ");
```


```
raph [] [nV]) {
][nV], i, j, k ;
; i++)
nV ; j++)
][j] = graph [i][j],

k++) {
nV ; i++) {
for (j=0 ; j<nV ; j++ ) {
  if (matrix [i][k ] + matrix [k][j]<
  matrix [i][j]
  matrix [i][j] = matrix [i][k] + matrix[k][j];
} } }
```

```
    printMatrix(matrix);
}

int main(){
    int graph[nV][nV] = {{0, 3, INF, 5},
                         {2, 0, INF, 4},
                         {INF, 1, 0, INF},
                         {INF, INF, 2, 0}};
    floydWarshall(graph);
}
```

OUTPUT:-

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

20/6/24

classmate
Date 04/07/2024
Page

Knapsack - problem using Dynamic programming

code:

```
#include <stdio.h>

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

void knapsack(int W, int wt[], int val[], int n){
    int i, ww;
    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[wt[i-1]<=w]
                dp[i][w]=max[val[i-1]+dp[i-1][
                [w-wt[i-1]], dp[i-1][w]);
            else
                dp[i][w]=dp[i-1][w];
        }
    }

int res = dp[n][W];
printf("Maximum profit %d\n", res);

w=W;
printf("Items included in the knapsack \n");
for(i=n; i>0 && res>0; i--){
    if(res==dp[i-1][w])
        continue;
```

```
        else {
            printf("items %d (weight %d profit %d)\n",
i,      wt[i-1], val[i-1]);
            res = res - val[i-1];
            w = w - wt[i-1];
        }
    }

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

int main() {
    int n = 4;
    int val[] = {3, 4, 5, 6};
    int wt[] = {2, 3, 4, 5};
    int W = 8;
    knapsack(W, wt, val, n);
    return 0;
}
```

OUTPUT:-

Maximum profit : 10
Items included in the knapsack:
Item 4 (weight: 5, profit: 6)
Item 2 (weight: 3, profit 4)

DP table:

0	0	0	0	0	0	0	0	0
0	0	3	3	3	3	3	3	3
0	0	3	4	4	7	7	7	7
0	0	3	4	5	7	8	9	9
0	0	3	4	5	7	8	9	10.

classmate
Date 04/02/2023
Page

PRIMS algorithm.

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>
#define V 5

int minKey (int key[], bool mstSet[]){
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;
    return min_index;
}

void printMST(int parent[], int graph[V][V])
{
    printf("Edge \t Weight \n");
    for (int j=0 1; i<V; i++){
        int parent[V];
        int 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] == false && graph[u][v] < key[v])
    parent[v] = u, key[v] = graph[u][v];
}
  printMST(parent, graph);
}

int main(){

  int graph[V][V] = {
        {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}
      };
      printMST(graph);
      return 0;
}
```

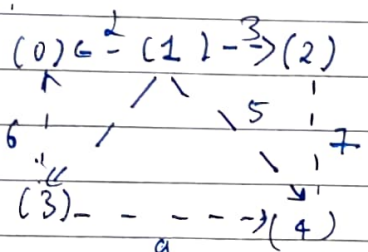
Op:-

Edge	Weight
0 - 1	2
1 - 2	3
0 - 3	6
1 - 4	5

4/7/22

16 is the total weigh of MST.

Input:-



Kruskal's algorithm

Code:-

```
#include <stdio.h>
#include <stdlib.h>
#define V 5

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

typedef struct {
    int parent, rank;
} Subset;

int compare (const void* a, const void* b){
    Edge* a1 = (Edge*) a;
    Edge* b1 = (Edge*) b;
    return a1->weight > b1->weight;
}

int find (Subset subsets[], int x, int y) {
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);
    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;
    else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}
```

```
void KruskalMST(Edge edges[], int E) {
    Edge result[V];
    int e = 0;
    int i = 0;

    qsort(edges, E, sizeof(edges[0]), compare);

    Subset *subsets = (Subset*) malloc(V * sizeof(Subset));
    for (int v=0; v<V; ++v) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    while (e < V - 1 && i < E) {
        Edge next_edge = edges[i++];
        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);

        if (x != y) {
            result[e++] = next_edge;
            Union(subsets, x, y);
        }
    }
    printf("Following are the edges in the constructed MST\n");
    for (j=0; j<e; ++j)
        printf("%d -- %d == %d\n", result[j].src, result[j].
dest, result[j].weight);
    return;
}

int main() {
    int graph[V][V] = { { 0, 2, 0, 6, 0},
                        { 2, 0, 3, 8, 5},
```

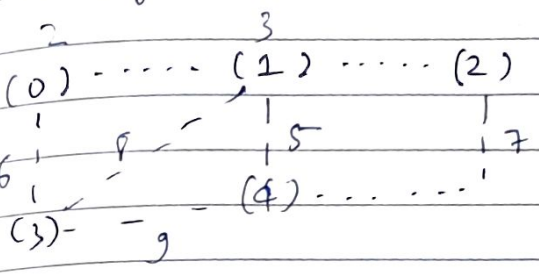
```
{0, 3, 0, 0, 7},
{6, 8, 0, 0, 9},
{0, 5, 7, 9, 0}};

Edge edges[9];
int k = 0;
for (int i = 0; i < V; i++) {
    for (int j = i+1; j < V; j++) {
        if (graph[i][j]) {
            edges[k].src = i;
            edges[k].dest = j;
            edges[k].weight = graph[i][j];
            k++;
        }
    }
}
KruskalMST(edges, k);
return 0;
}
```

OUTPUT

~~Vertex~~	~~Distance from source~~
~~0~~	~~0~~
~~1~~	~~10~~
~~2~~	~~20~~
~~3~~	~~22~~
~~4~~	~~20~~

0 - - 1 = = 2
1 - - 2 = = 3
1 - - 4 = = 5
0 - - 3 = = 6

Input graph



Dijkstras algorithm

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

#define V 5

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 \t\t Distance from source\n");
    for(int i=0; i<V; j++)
        printf("%d \t\t %d\n", 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[v] > graph[u][v] + graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
    }
    printSolution(dist);
}

int main() {
    int graph[V][V] = {{ 0, 10, 20, 0, 0},
                       {10, 0, 0, 50, 10},
                       {20, 0, 0, 20, 33},
                       { 0, 50, 20, 0, 2},
                       { 0, 10, 33, 2, 0}};

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

OUTPUT:

Vertex	Dist from source.
0	0
1	10
2	20
3	22
4	20

11/7

Induced graph