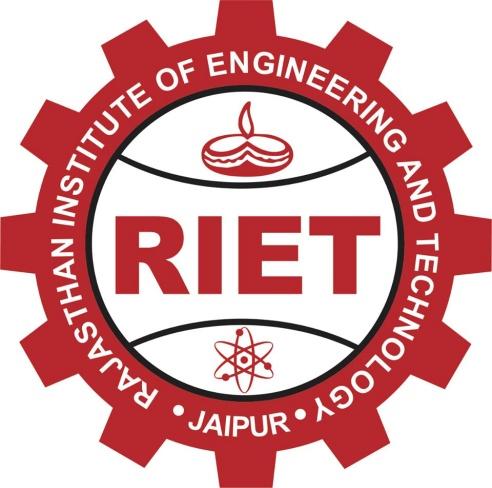
****

**MCA SEM-III**

**ADA LAB MANUAL(MCA-351)**

**SESSION 2021-23**



Submitted to: Submitted by

Mr. Anil Dhankhar Ajit kumar kushwaha

(Hod of MCA Department) Roll No- 21CRIXX601

**Experiment 1**

**1. Implement recursive linear search and determine the time required to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and the time taken versus n.**

**Recursive Linear search**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<timer.h>

int a[20], low, high, ele, pos;

int lin() {

if (pos < 0) {

return pos + 1;

} else if (a[pos] == ele) {

return pos + 1;

} else {

pos = pos - 1;

return lin();

}

}

void main() {

int i, n, temp;

Timer t;

clrscr();

printf("\nEnter the number of elements ");

scanf("%d", & n);

randomize();

printf("\nEnter the search element ");

scanf("%d", & ele);

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

a[i] = rand();

}

pos = n - 1;

t.start();

temp = lin();

t.stop();

printf("\n Element found at position = %d and time taken = %lf", temp, (double) t.time());

getch();

}

**Output:**

Enter the number of elements: 5

Enter the search element: 10

Element found at position = 2

**Experiment 2**

**2. Implement recursive Binary search and determine the time required to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and the time taken versus n.**

**Recursive Binary search**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<timer.h>

int a[20], low, high, ele, pos;

int bin() {

int mid = (low + high) / 2;

if (low > high) {

return 0;

}

if (a[mid] == ele) {

return mid + 1;

} else if (a[mid] > ele) {

high = mid - 1;

return bin();

} else if (a[mid] < ele) {

low = mid + 1;

return bin();

}

}

void main() {

int i, n, temp;

Timer t;

clrscr();

printf("\nEnter the number of elements ");

scanf("%d", & n);

randomize();

printf("\nEnter the search element ");

scanf("%d", & ele);

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

a[i] = i;

}

low = 0;

high = n - 1;

t.start();;

temp = bin();

t.stop();

printf("\nElement found at position = %d and time taken = %lf", temp, (double) t.time());

getch();

}

**Output:**

Enter the number of elements: 5

Enter the search element: 2

Element found at position = 1

**Experiment 3**

**3. Find the minimum cost spanning tree of a given undirected graph using Kruskal's algorithm.**

**Kruskal's algorithm.**

#include<stdio.h>

#include<conio.h>

void main() {

int a[20][20], b[20][20], c[20][20], d[20][20], nod = 0, n, val1 = 0, i, j, k, t, m = 0, posx, posy, val;

clrscr();

printf("\nEnter the value of n ");

scanf("%d", & n);

printf("\nEnter the adjacency matrix ");

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

scanf("%d", & a[i][j]);

b[i][j] = (i == j ? 0 : a[i][j]);

m = m + (b[i][j] ? 1 : 0);

c[i][j] = 0;

d[i][j] = 0;

}

}

for (m = m / 2; m != 0 && (nod != (n - 1)); m--) {

val = 32767;

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

if (b[i][j] != 0 && b[i][j] < val) {

posx = i;

posy = j;

val = b[i][j];

}

}

}

b[posx][posy] = 0;

b[posy][posx] = 0;

if (c[posx][posy] == 0) {

c[posx][posy] = 1;

c[posy][posx] = 1;

for (k = 0; k < n; k++) {

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

c[i][j] = c[i][j] | (c[i][k] & c[k][j]);

}

}

}

val1 = val1 + a[posx][posy];

nod = nod + 1;

d[posx][posy] = a[posx][posy];

d[posy][posx] = a[posy][posx];

}

}

if (nod == n - 1) {

for (i = 0; i < n; i++) {

printf("\n");

for (j = 0; j < n; j++) {

printf("%d ", d[i][j]);

}

}

printf("\n Spanning tree has a cost of %d", val1);

} else {

printf("\n Spanning tree does not exist!!");

}

getch();

}

**Output:**

Enter the value of n: 5

Enter the adjacency matrix:

2 3 4 5 3 4 3 6 8 4 9 2 1 5 1 9 8 3 4 8 1 2 8 4 6 5

Spanning tree has a cost of 18

**Experiment 4**

**4. Sort a given set of elements using selection sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and the time taken versus n.**

**Selection sort.**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<malloc.h>

#include<timer.h>

void sel(int a[], int n) {

int i, j, temp, pos;

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

pos = i;

for (j = i + 1; j < n; j++) {

if (a[j] < a[pos]) {

pos = j;

}

}

temp = a[i];

a[i] = a[pos];

a[pos] = temp;

}

}

void main() {

int \* a, n, i;

Timer t;

clrscr();

printf("\nEnter the value of n ");

scanf("%d", & n);

a = (int \* ) malloc(n);

randomize();

for (i = 0; i < n; i++) {

a[i] = rand();

}

t.start();

sel(a, n);

t.stop();

printf("\nTime taken = %lf", t.time());

getch();

}

**Output:**

Enter the value of n: 5

Time taken 10ms

**Experiment 5**

**5. Sort a given set of elements using heap sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and the time taken versus n.**

#include<stdio.h>

#include<timer.h>

#include<stdlib.h>

#include<conio.h>

void heap(int a[], int n) {

int i, j, k, temp;

for (i = 2; i <= n; i++) {

j = i;

k = j / 2;

temp = a[j];

while (k > 0 && a[k] < temp) {

a[j] = a[k];

j = k;

k = k / 2;

}

a[j] = temp;

}

}

void heap1(int a[], int n) {

int i, j, k, temp;

for (i = n / 2; i > 0; i--) {

k = i;

temp = a[k];

j = 2 \* k;

while (j <= n) {

if (j < n && a[j] < a[j + 1]) {

j = j + 1;

}

if (temp < a[j]) {

a[k] = a[j];

k = j;

j = 2 \* k;

} else {

break;

}

}

a[k] = temp;

}

}

void adjust(int a[], int n) {

int i = 2, temp = a[1];

while (i <= n) {

if (i < n && a[i] < a[i + 1]) {

i = i + 1;

}

if (a[i] > temp) {

a[i / 2] = a[i];

i = i \* 2;

} else {

break;

}

}

a[i / 2] = temp;

}

void main() {

int a[10000], n, i, temp;

Timer t;

clrscr();

printf("\nEnter the value of n ");

scanf("%d", & n);

for (i = 1; i <= n; i++) {

a[i] = n - i;

}

t.start();

heap1(a, n);

for (i = n; i >= 2; i--) {

temp = a[1];

a[1] = a[i];

a[i] = temp;

adjust(a, i - 1);

}

t.stop();

printf("\n Time taken is %lf ", t.time());

getch();

}

**Output:**

Enter the value of n: 5

Time taken 12ms

**Experiment 6**

**6. Sort a given set of elements using merge sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements** **in the list to be sorted and the time taken versus n.**

**Merge sort**

#include<stdio.h>

#include<conio.h>

#include<timer.h>

#include<stdlib.h>

void copy(int a[], int b[], int st, int end) {

int i;

for (i = st; i < end; i++) {

b[i - st] = a[i];

}

}

void merge(int a[], int n) {

int \* b, \* c, i, j;

if (n == 1) {

return;

}

b = (int \* ) malloc(n / 2);

copy(a, b, 0, n / 2);

merge(b, n / 2);

c = (int \* ) malloc(n - (n / 2));

copy(a, c, n / 2, n);

merge(c, n - (n / 2));

j = 0;

for (i = 0;

(i + j) < n;) {

if (j == n - (n / 2)) {

a[i + j] = b[i];

i = i + 1;

} else if (i == (n / 2)) {

a[i + j] = c[j];

j = j + 1;

} else if (b[i] > c[j]) {

a[i + j] = c[j];

j = j + 1;

} else {

a[i + j] = b[i];

i = i + 1;

}

}

}

void main() {

int \* a, n, i;

Timer t;

clrscr();

printf("\nEnter the value of n ");

scanf("%d", & n);

a = (int \* ) malloc(n);

randomize();

for (i = 0; i < n; i++) {

a[i] = rand();

}

t.start();

merge(a, n);

t.stop();

printf("\nTime taken = %lf", t.time());

getch();

}

**Output:**

Enter the value of n: 5

Time taken 8ms

**Experiment 7**

**7. Sort a given set of elements using quick sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and the time taken versus n.**

**Quick sort.**

#include<stdio.h>

#include<conio.h>

#include<TIMER.H>

#include<alloc.h>

#include<stdlib.h>

int n;

int quick(int a[], int l, int h) {

int piv, temp, i, j;

if (l == h) {

return l;

}

piv = a[l];

i = l + 1;

j = h;

while (i <= j) {

for (; i <= h && a[i] <= piv; i++);

for (; j >= (l + 1) && a[j] >= piv; j--);

if (j == l) {

return l;

} else if (i == h + 1) {

a[l] = a[h];

a[h] = piv;

return h;

} else if (i > j) {

a[l] = a[j];

a[j] = piv;

return (j);

} else {

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

}

void quicksort(int a[], int st, int end) {

int pos;

if (st < end) {

pos = quick(a, st, end);

quicksort(a, st, pos - 1);

quicksort(a, pos + 1, end);

}

}

void main() {

int \* a, i;

Timer t;

clrscr();

printf("\nEnter the value of n ");

scanf("%d", & n);

a = (int \* ) malloc(n);

randomize();

for (i = 0; i < n; i++) {

a[i] = rand();

}

t.start();

quicksort(a, 0, n - 1);

t.stop();

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

if (a[i] > a[i + 1]) {

printf("%d %d", a[i], i);

}

}

printf("\nTime taken = %lf", t.time());

getch();

}

**Output:**

Enter the value of n: 5

9 23 24 56 78

Time taken 8ms

**Experiment 8**

**8. Find the minimum cost spanning tree of a given undirected graph using Prim's algorithm.**

**Prim's algorithm.**

#include<stdio.h>

#include<conio.h>

void main() {

int a[20][20], add[20], f[20][20], min, posx, posy, i, j, k, n, tot, flag = 0, weight = 0;

clrscr();

tot = 0;

printf("\nEnter the value of n ");

scanf("%d", & n);

printf("\nEnter the adjacency matrix ");

for (i = 0; i < n; i++) {

add[i] = 0;

for (j = 0; j < n; j++) {

scanf("%d", & a[i][j]);

f[i][j] = 0;

}

}

add[0] = 1;

while (flag == 0) {

flag = 1;

min = 9999;

for (i = 0; i < n; i++) {

if (add[i] == 1) {

for (j = 0; j < n; j++) {

if (add[j] == 0 && a[i][j] != 0 && a[i][j] < min) {

min = a[i][j];

posx = i;

posy = j;

}

}

}

}

if (min != 9999) {

f[posx][posy] = min;

f[posy][posx] = min;

weight = weight + f[posx][posy];

tot++;

flag = 0;

add[posx] = 1;

add[posy] = 1;

}

}

if (tot != n - 1) {

printf("\n Minimum spanning does not exist!!");

} else {

printf("\n Minimum spanning tree with weight=%d is ", weight);

for (i = 0; i < n; i++) {

printf("\n");

for (j = 0; j < n; j++) {

printf("%d ", f[i][j]);

}

}

}

getch();

}

**Output:**

Enter the value of n: 5

Enter the adjacency matrix:

2 3 4 5 3 4 3 6 8 4 9 2 1 5 1 9 8 3 4 8 1 2 8 4 6 5

Spanning tree has a cost of 18

**Experiment 9**

**9. Implement 0/1 knapsack using dynamic programming.**

**0/1 knapsack.**

#include<stdio.h>

#include<conio.h>

void main() {

int p[20], w[20], kn[20][20], x[20], i, j, n, weight;

printf("\nEnter the value of n ");

scanf("%d", & n);

printf("\nEnter the price of the items ");

for (i = 0; i < n; i++) {

scanf("%d", & p[i]);

}

printf("\nEnter the weight of the items ");

for (i = 0; i < n; i++) {

scanf("%d", & w[i]);

}

printf("\nEnter the weight of the knapsack ");

scanf("%d", & weight);

printf("\n The knapsack is ");

for (i = 0; i <= n; i++) {

printf("\n");

for (j = 0; j <= weight; j++) {

if (i == 0 || j == 0) {

kn[i][j] = 0;

} else if (w[i - 1] > j) {

kn[i][j] = kn[i - 1][j];

} else {

kn[i][j] = (kn[i - 1][j] > (kn[i - 1][j - w[i - 1]] + p[i - 1])) ? kn[i - 1][j] : (kn[i - 1][j - w[i - 1]] + p[i - 1]);

}

printf("%d ", kn[i][j]);

}

}

printf("\n The optimal solution is %d", kn[n][weight]);

i = n;

j = weight;

while (i != 0) {

if (kn[i][j] == kn[i - 1][j]) {

x[i - 1] = 0;

i = i - 1;

} else {

x[i - 1] = 1;

j = j - w[i - 1];

i = i - 1;

}

}

printf("\n The 0/1 knapsack is ");

for (i = 0; i < n; i++) {

printf("\n X[%d]=%d", i + 1, x[i]);

}

getch();

}

**Output:**

Enter the value of n: 5

Enter the price of the items: 3 4 5 6 7

Enter the weight of the items: 1 2 3 4 5

Enter the weight of the knapsack: 2

The knapsack is :

0 0 0

0 3 3

0 3 4

0 3 4

0 3 4

0 3 4

The optimal solution is 4

The 0/1 knapsack is

X[1] =0

X[2]=1

X[3]=0

X[4]=0

X[5]=0

**Experiment 10**

**10. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.**

**Dijkstra's algorithm.**

#include<stdio.h>

#include<conio.h>

struct prioq {

int dist, pr, s;

};

void display(struct prioq p[20], int source, int dest, int d) {

if (dest == source) {

printf("%d", source);

return;

}

display(p, source, p[dest].s, d);

printf("->%d", dest);

}

void main() {

struct prioq p[20];

int i, j, n, k, min, pos, source, a[20][20];

clrscr();

printf("\nEnter the value of n ");

scanf("%d", & n);

printf("\nEnter the adjacency matrix ");

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

scanf("%d", & a[i][j]);

}

p[i].pr = 0;

}

printf("\nEnter the source node ");

scanf("%d", & source);

for (i = 0; i < n; i++) {

p[i].s = source;

p[i].dist = a[source][i];

}

pos = source;

for (j = 0; j < n; j++) {

i = pos;

p[i].pr = -1;

min = 9999;

for (k = 0; k < n; k++) {

if (p[i].dist + a[i][k] < p[k].dist) {

p[k].dist = p[i].dist + a[i][k];

p[k].s = i;

}

if (p[k].pr != -1 && p[k].dist < min) {

pos = k;

min = p[k].dist;

}

}

}

for (i = 0; i < n; i++) {

printf("\n");

if (i == source) {

printf("%d->%d", source, source);

} else {

display(p, source, i, i);

}

printf(" %d", p[i].dist);

}

getch();

}

**Output:**

Enter the value of n 5

Enter the adjacency matrix 1 2 3 4 5 6 7 8 9 8 7 6 5 4 3 2 1 2 3 4 5 6 7 8 9

Enter the source node 1

1->0 6

1->1 7

1->2 8

1->3 9

1->4 8

**Experiment 11**

**11. Implement N Queens problem using back tracking.**

**N-Queens problem.**

#include<stdio.h>

#include<conio.h>

#include<math.h>

void main() {

int n, arr[20][20], i, j, k, tot = 0, x[20], top;

char ch;

clrscr();

printf("\nEnter the total number of queens ");

scanf("%d", & n);

top = 0;

x[top] = -1;

printf("\n The solution to the %d queens problem is ", n);

while (top >= 0) {

x[top] = x[top] + 1;

while (x[top] < n) {

for (i = 0; i < top; i++) {

if (x[i] == x[top] || (abs(x[top] - x[i]) == abs(top - i))) {

break;

}

}

if (i == top) {

break;

}

x[top] = x[top] + 1;

}

if (x[top] == n) {

top = top - 1;

} else if (top == n - 1) {

tot = tot + 1;

ch = getch();

if (ch == 0) {

ch = getch();

}

printf("\n");

for (i = 0; i < n; i++) {

printf("\n");

for (j = 0; j < n; j++) {

if (x[i] == j) {

printf("Q ");

} else {

printf("X ");

}

}

}

top = top - 1;

} else {

top = top + 1;

x[top] = -1;

}

}

if (tot == 0) {

printf("not possible!!");

} else {

printf("\n The total number of solutions is %d.", tot);

}

getch();

}

**Output:**

Enter the total number of queens 8

The solution to the 8 queens problem is

Q X X X X X X X

X X X X Q X X X

X X X X X X X Q

X X X X X Q X X

X X Q X X X X X

X X X X X X Q X

X Q X X X X X X

X X X Q X X X X