Lab - 01

Date 3/5/24
Page 01

Bubble Sort

```
#include <stdio.h>
#include <time.h>
void bubblesort(int[], int);
void main()
{
    int i, n, a[10];
    printf("Enter size: \n");
    scanf("%d", &n);
    printf("Enter array elements: \n");
    for(i=0; i<n; i++)
        scanf("%d", &a[i]);
    bubblesort(a, n);
}
void bubblesort(int a[], int n){
    clock_t start_t, end_t, total_t;
    start_t = clock();
    printf("Starting of linear search funct^n, \
        Start_t = %d\n", start_t);
    int t, i, j;
    for(i=0; i<n-1; i++){
        for(j=0; j<n-1-i; j++){
            if(a[j] > a[j+1]){
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
            }
        }
    }
    printf("\n In Ascending order: ");
    for(i=0; i<n; i++)
        printf("%d\t", a[i]);
    end_t = clock();
```

```
printf("\n End of the ~~linear search~~ funt^n",
        end_t = %d\n", end_t);
total_t = (double) (end_t - start_t) / clocks_per_sec;
printf("Total time taken by CPU : %d\n",
        total_t);
printf(" Exiting of the program ... \n");
```

O/P

```
Enter size : 5
Enter array element : 28 19 100 0 10
Starting of the funt^n = 1888
In ascending order = ~~0 10 19 28 100~~
                     0 10 19 28 100
End of the funt^n = 1888
Total time taken by CPU : 0
```

Selection Sort

```
#include <stdio.h>
#include <time.h>
void selectionsort(int, int[]);
void main(){
    int i, n, a[10];
    printf("Enter size: \n");
    scanf("%d", &n);
    printf("Enter array elements: \n");
    for(i=0; i<n; i++)
        scanf("%d", &a[i]);

    selectionsort(n, a);
}
void selectionsort(int n, int a[]){
    clock_t start_t, end_t, total_t;
    start_t = clock();
    printf("Starting of func -> start_t = %ld \n",
                start_t);
    int i, j, min;

    for(i=0; i<n-1; i++){
        min = i
        for(j=i+1; j<n; j++)
            if(a[j] < a[min])
                min = j
        if(min != i){
            int t = a[min];
            a[min] = a[i];
            a[i] = t;
        }
```

```
printf("\n In ascending order");
for(i=0; i<n; i++)
    printf("%d\t", a[i]);
                end_t = clock();
printf("\n End of the functo; end_t = %d\n",
                end_t);
Total_t = (double)(end_t - start_t) / clocks_per_sec;
printf("Total time taken by CPU: %d\n",
                total_t);
```

O/P

```
Enter size : 5
Enter array element : 21 81 0 2 110
Starting of funtn = 19921
In ascending order = 0 2 21 81 110
end of funtn = 19921
Total time taken by CPU : 0
```

~~Binary~~ linear search

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

void linearsearch (int, int, int[]);
int main (){
    int n, key, a[10];
    printf ("Enter the no of element: ");
    scanf ("%d", &n);
    printf ("Enter array element:\n");
    for (int i=0; i<n; i++)
        scanf ("%d", &a[i]);
    printf ("\n Enter key element:\n");
    scanf ("%d", &key);
    linearsearch (n, key, a);
}
void linearsearch (int n, int key, int a[]){
    for (int i=0; i<n; i++){
        if (a[i] == key){
            printf ("\n Successful search");
            break;
        }
    }
    printf ("\n unsuccessful search");
}
```

Op: Enter the no. of element: 5
Enter the array element: 51, 28, 19, 29 10
Enter key element: 10
Successful search

Date / /
Page 06

Binary search

```
#include <stdio.h>
#include <time.h>
void binarysearch(int, int, int[]);
void main(){
    int i, n, key, a[10];
    printf("Enter ~~key~~ size: \n");
    scanf("%d", &n);
    printf("Enter array element:");
    for(i=0; i<n; i++)
        scanf("%d", &a[i]);
    printf("\n Enter key:");
    scanf("%d", &key);
    binarysearch(n, key, a);
}
void binarysearch(int n, int key, int a[]){
    clock_t start_t, end_t, total_t;
    start_t = clock();
    printf("Starting of funct, start_t = %ld\n",
            start_t);
    int t, i, j;
    for(i=1; i<n; j++){
        for(j=0; j<n-i; j++){
            if(a[j] > a[j+1]){
                ~~if(a[j]>~~
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
            }
        }
    }
```

```
int h = n-1, l=0, mid, flag=0;
while (h >= l) {
  mid = (h+1)/2
    if (a[mid] == key) {
      flag = 1;
      printf ("Successful search");
      break;
    }
    if (a[mid] < key) {
      l = mid + 1;
    }
      if (a[mid] > key) {
        h = mid - 1;
      }
    }
    if (flag == 0)
      printf ("unsuccessful search");
      endt = clock();
printf ("\n End of funct^n, endt = %d\n", endt);
 total_t = (double) (endt - start_t) / clockpersec;
 printf ("Total time taken by CPU : %d\n", totalt
}
```

O/P
```
Enter size : 3
Enter array element : 91 81 23
Enter key 11
Starting of funct^n = 18182
unsuccessful search
End of funct^n = 18182
Total time taken by CPU : 0
```

Lab - 02

Date / /
Page 08

O/P for Computing median

Enter the number of element : 9
Enter the elements : 4 1 10 8 7 12 9 2 15
median = 2

O/P for GCD

Enter the Two integers : 26 6
GCD of 26 & 6 is 2

O/P for Tower of Hanoi

The moves involved in Tower of Hanoi are :
Move disk 1 from 1 to 3
move disk 2 from 1 to 3
move disk 1 from 3 to 2
move disk 3 from 1 to 3
move disk 1 from 2 to 1
move disk 2 from 2 to 2
move disk 1 from 2 to 1

O/P Topological sort DFS

Enter the no. of nodes 4
Enter adjacent matrix

0 0 0 1
1 0 1 0
0 0 1 0
0 1 0 0

Topological sort is : 0 3 1 0

Date / /
Page 09

o/p :- Topological sort - Source removal

Enter the no/. of nodes : 3
Enter the adjacent matrix :

0 1 0 0
0 1 1 0
1 0 1 0

Topological order
Graph has a cycle Topological sort not possible

Merge sort

```
#include <stdio.h>
void merge (int a[], int low, int mid, int high){
    int i, j, k;
    int n = high - low + 1;
    int c[n];
    i = low;
    j = mid + 1;
    k = 0;
    while (i <= mid && j <= high){
        if (a[i] < a[j]){
            c[k++] = a[i++];
        }else{
            c[k++] = a[j++];
        }
    }
    while (i <= mid){
        c[k++] = a[i++];
    }
    while (j <= high){
        c[k++] = a[j++];
    }
    for (i=0; i<n; i++){
        a[low+i] = c[i];
    }
}

void merg_sort (int a[], int low, int high){
    if (low < high){
        int mid = (low + high)/2;
        mergesort (a, low, mid);
```

```
        mergesort(a, mid+1, high);
        merge(a, low, mid, high);
    }
}
int main(){
    int arr[20];
    int i, n;
    printf("Enter the number of element: ");
    scanf("%d", &n);
    printf("Enter the array element:\n");
    for(i=0; i<n; i++){
        scanf("%d", &arr[i]);
    }
    merge_sort(arr, 0, n-1);
    printf("Sorted Array:\n");
    for(i=0; i<n; i++){
        printf("%d", arr[i]);
    }
    printf("\n")
    return 0;
}
```

O/P:-

Enter the number of element: 7
Enter the array element:
3 8 12 0 26 4 9
Sorted Array:
0 3 4 8 9 12 26

Quick Sort

```
#include <stdio.h>
int partition(int a[], int low, int high){
    int i, j, pivot;
    i = low;
    j = high + 1;
    pivot = a[low];
    int temp;
    while(i <= j){
        do
            i = i + 1;
        while(a[i] < pivot && i <= high);
        do
            j = j - 1;
        while(a[j] > pivot && j <= high);
        if(i < j){
            a[i] = temp;
            temp = a[j];
            a[j] = a[i];
        }
    }
    a[low] = a[j];
    a[j] = pivot;
    return j;
}

void QuickSort(int a[], int low, int high){
    if(low < high){
        int mid = partition(a, low, high);
        QuickSort(a, low, mid - 1);
        QuickSort(a, mid + 1, high);
    }
}
```

```
int main(){
    int arr[20];
    int i, n;
    printf("Enter the number of element:")
    scanf("%d", &n);
    printf("Enter the array element:\n");
    for(i=0; i<n; i++){
        scanf("%d", &arr[i]);
    }
    QuickSort(arr, 0, n-1);
    printf("Sorted array:\n");
    for(int i=0; i<n; i++){
        printf("%d", arr[i]);
    }
    printf("\n");
    return 0;
}
```

o/p

Enter the number of elements: 4
Enter the array elements:
9 2 8 1
Sorted array:
1 2 8 9

7/6/24

1] Warshall algorithm

```
#include <stdio.h>
int a[10][10], n;
void warshall (int a[][10], int n){
  int p[10][10];
  int i j k;
  for(i=0; i<n; i++)
  {
     for (j=0; j<n; j++){
        p[i][j] = a[i][j];
     }
  }
  for (k=0; k<n; k++)
  {
     for(i=0; i<n; i++)
     {
        for (j=0; j<n; j++)
        {
           if (p[i][j] == 0 && (p[i][k] == 1 &&
              p[k][j] == 1))
                 p[i][j] = 1;
        }
     }
  }
  printf ("Path matrix : \n");
  for(i=0; i<n; i++)
  {
     for (j=0; j<n; j++)
        printf ("%d\t", p[i][j]);
```

```
    printf("\n");
}
void main(){
    int b[10][10];
    int n, i, j;
    printf("Enter Size: ");
    scanf("%d", &n);
    printf("Enter elements: \n");
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
            scanf("%d", &b[i][j]);
    }
    warshall(b, n);
}
```

O/P
```
Enter Size : 4
Enter elements
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
Path matrix
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1
```

3) Knapsack Algorithm [using dynamic programming]

```
#include <stdio.h>
int v[10][10];
void knapsack (int n, int m, int w[], int p[]) {
    int i, j;
    for (i=0; i<n; i++)
    {
        for (j=0; j<=m; j++) {
            if (i==0 || j==0)
                v[i][j] = 0;
            else if (w[i] > j)
                v[i][j] = v[i-1][j];
            else {
                int max = v[i-1][j];
                if (max < (v[i-1][j-w[i]] + p[i]))
                    max = v[i-1][j-w[i]] + p[i];
                v[i][j] = max;
            }
        }
    }
    printf ("Distance matrix : \n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            printf ("%d\t", v[i][j]);
        }
        printf ("\n");
    }
}

void main () {
    int m, n, w[10], p[10], v[10][10];
```

Date / /
Page 17

```
scanf("%d %d", &n, &m);
printf("Enter weights: \n");
for(i=0; i<n; i++){
    scanf("%d", &w[i]);
}
printf("Enter profits: \n");
for(i=0; i<n; i++)
    scanf("%d", &p[i]);
knapsack(n, m, w, p);
}
```

O/P :-

```
Enter number of items & no. of knapsack:
4 5
Enter weights:
2 1 3 2

Enter profits:
12 10 20 15

Distance matrix:
0  0  12  12
0  10 12  22
0  10 12  22
0  10 15  25
```

3) Floyd's Algorithm

```
#include <stdio.h>
int a[10][10], n;
void floyds (int a[10][10], int n){
    int j, i, k;
    int d[10][10];
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            d[i][j] = a[i][j];
        }
    }
    for(k=0; k<n; k++){
        for(i=0; i<n; i++){
            for(j=0; j<n; j++){
                int min = d[i][j];
                if(min>d[i][k] + d[k][j]){
                    min = d[i][k] + d[k][j];
                }
                d[i][j] = min;
            }
        }
    }
    printf("Distance matrices: \n");
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            printf("%d\t", d[i][j]);
        }
        printf("\n");
    }
}
```

```
void main ()
{
    int b[10][10];
    int n, i, j;
    printf("Enter Size : ");
    scanf("%d", &n);
    printf("Enter elements : \n");
    for (i=0; i<n; i++)
    {
        for (j=0; j<n; j++)
        {
            scanf("%d", b[i][j]);
        }
    }
    floyds(b, n);
}
```

O/P

```
Enter Size : 4
Enter elements
0   999   3   999
2   0     999 999
999 6     0   1
7   999   999 0
```

Distance matrices

```
0  9   3  4
2  0   5  6
8  6   0  1
7  16  10 0
```

4 Presorting using mergesort

```
#include <stdio.h>
int a[10];
void merge (int a[], int l, int m, int x)
{
   int i, j, k;
   int n1 = m-l+1;
   int n2 = x-m;
   int L[n1], R[n2];
   for (i=0; i<n1; i++)
      L[i] = a[l+i]
   for (j=0; j<n2; j++)
      R[j] = arr[m+1+j];
  i = 0;
  j = 0;
  k = l;
  while (i<n1 && j<n2)
  {
    if (L[i] <= R[j])
    {
      a[k] = L[i];
      i++;
    }
    else
      a[k] = R[j];
      j++;
    }
    k++;
  }
  while (i<n1){
    a[k] = R[i];
    i++;
  k++;
```

```
while (j<n2) {
    a[k] = R[j]
    j++;
    k++;
}
}

void mergesort (int a[], int l, int r) {
    if (l<r)
    {
        int m = l+(r-1)/2;
        mergesort(a, l, m);
        mergesort (a, m+1, r);
        merge (a, l, m, r);
    }
}

int main () {
    int i, n;
    printf ("Enter the no. of elements: ");
    scanf ("%d", &n);
    int a[n];
    printf ("Enter %d integers: \n", n);
    for (i=0; i<n; i++)
        scanf ("%d", &arr[i]);

    mergesort (a, 0, n-1);
    printf ("Array after presorting using
            merge sort: \n");
    for (i=0; i<n; i++)
        printf ("%d\t", a[i]);
    return 0;
}
```

O/P Enter the number of elements

Enter 5 integers

6 3 9 8 2 10 15 1

Array after presorting using mergesort

1 2 3 6 8 9 10 15

21/6/24

21/6/24

Date / /
Page 23

Horspool Algorithm

```
#include <stdio.h>
#include <string.h>
#define MAX_CHAR 128
void shift_table(char P[], int m, int S[]){
    for(int i = 0; i < MAX_CHAR; i++){
        S[i] = m;
    }
    for(int i = 0; i < m-1; i++){
        S[P[i]] = m-1-i;
    }
}
int horspool(char P[], char t[]){
    int n = strlen(t);
    int m = strlen(P);
    int S[MAX_CHAR];
    shift_table(p, m, S);

    int i = m-1;
    while(i <= n-1){
        int k = 0;
        while(k <= m-1 && t[i-k] == p[m-1-k]){
            k++;
        }
        if(k == m){
            return i-m+1;
        }
        else{
            i += S[t[i]];
        }
    }
    return -1;
}
```

```
int main(){
    char text[1000];
    char pattern[100];
    printf("Enter text : ");
     scanf("%[^\n]", text);
    printf("Enter pattern : ");
    scanf("%s", pattern);
    int pos = horspool(pattern, text);
    if(pos == -1){
        printf("Pattern not found.\n");
    }
    else{
        printf("Pattern found at position : %d\n",
                                    pos);
    }
    return 0;
}
```

O/P

Enter text : JIM SAW ME IN BARBER SHOP
Enter pattern : BARBER
pattern found at position : 15

Enter text : JIM SAW ME IN BARBER SHOP
Enter pattern : BABA
pattern not found

Heapify

```
#include <stdio.h>
int a[10], n;
void heapify (int [], int);
int main (){
    printf ("Enter The number of array
                element : ");
    Scanf ("%d", &n);
    int i;
    printf (" Enter array elements : ");
    for (i=0 ; i<n ; i++){
        Scanf ("%d", &a[i]);
    }
    heapify (a, n);
    printf ("Array element : ");
    for (i=0 ; i<n ; i++){
        printf ("%d", a[i]);
    }
    return 0;
}

Void heapify ( int a[], int n) {
    int k;
    for (k=1; k<n ; k++){
        int key = a[k];
        int c = k
        int p = (c-1)/2 ;
        while (c>0 && key > a[p]){
            a[c] = a[p];
            c = p;
            p = (c-1)/2 ;
        }
        a[c] = key ;
    }
```

int ~~bte~~

O/P

Enter the number of array element : 7
Enter the array element : 50, 25, 30 75 100 45 80
Array elements : 100 75 80 25 50 80 45

21/6/24

Lab-06

Date 5/7/24
Page 07

Prims Algorithm

```
#include <stdio.h>

void prims(int cost[10][10], int n)
{
  int i, j;
      u, v;
      sum, k;
      t[10][2];
      p[10], d[10], s[10];
      min = 999;
      source = 0;
  for(i=0; i<n; i++){
      for(j=0; j<n; j++){
          if(cost[i][j] != 0 && cost[i][j] < min){
              min = cost[i][j];
              source = i;
          }
      }
  }
  for(i=0; i<n; i++){
      d[i] = cost[source][i];
      s[i] = 0;
      p[i] = source;
  }
  s[source] = 1;
  sum = 0;
  k = 0;
  for(i=1; i<n; i++){
      min = 999;
      u = -1
      for(j=0; j<n; j++){
```

```
        if(s[j] == 0){
            if (d[j] <= min){
                min = d[j];
                u = j;
            }
        }
    }
    t[k][0] = u;
    t[k][1] = p[u];
    k = k+1;
    sum = sum + cost[u][p[u]];
    s[u] = 1
    for (v=0; v<n; v++){
        if (s[v] == 0 && cost[u][v] < d[v]){
            d[v] = cost[u][v];
            P[v] = u;
        }
    }
}
printf("\n Weighted minimum Spanning tree\n");
for(i=0; i<n; i++){
    printf("(%d, %d) -> weight : %d\n",
        p[i], i, cost[p[i]][i]);
}
printf("\n Sum of minimum Spanning tree : %d", sum);
}

void main(){
    int i, j, n;
    int cost[10][10];
    printf("Enter number of vertices:");
```

```
scanf("%d", &n);
printf("\nEnter cost adjacency matrix:\n");
for(i=0; i<n; i++) {
    for(j=0; j<n; j++) {
        scanf("%d", &cost[i][j]);
    }
}
prims(cost, n);
}
```

Output :-

Enter the number of vertices : 4
Enter the cost adjacency matrix

0	1	5	2
1	0	999	999
5	999	0	3
2	999	3	0

Weighted minimum spanning tree
(0, 0) → weight : 0
(0, 1) → weight : 1
(3, 2) → weigth : 3

Kruskal's Algorithm

```
#include <stdio.h>
void kruskals(int cost[][10], int n){
    int i, j ;
    int u, v ;
    int sum=0 ;
    int k=0 ;
    int t[10][2] ;
    int p[10] ;
    int min=999 ;
    int count=0 ;

for (i=0 ; i<n ; i++){
        p[i] = i ;
}
while(count <n-1){
    min = 999 ;
    for(i=0 ; i<n ; i++){
        for(j=0 ; j<n ; j++){
            if(cost[i][j] < min){
                min = cost[i][j] ;
                u=i ;
                v=j ;
            }
        }
    }
    if (u != v){
        t[k][0] = u ;
        t[k][1] = v ;
        k = k+1 ;
        count = count+1 ;
        sum = sum + min ;
    }
```

```
    cost[u][v] = cost[v][u] = 999;
}
printf("Minimum Spanning Tree: \n");
for(i=0; i<n-1; i++){
    printf("%d -> %d\n", t[i][0], t[i][1]);
}
printf("Total cost: %d\n", sum);
}
int main(){
    int cost[10][10], n;
    int i, j;
    printf("Enter the no of vertices: ");
    scanf("%d", &n);

    printf("Enter the cost adjacency matrix:\n");
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            scanf("%d", &cost[i][j]);
        }
    }
    kruskals(cost, n)
    return 0;
}
```

Output

Enter the no of vertices: 4

Enter the cost adjacency matrix:

```
0  2  999  6
2  0  3    8
999 3  0  999
6  8  999  0
```

Minimum Spanning tree:

```
0 -> 1
1 -> 2
0 -> 3
```

Total cost : 11

Dijkstra's Algorithm

```
#include <stdio.h>
void dijkstras(int a[10][10], int n){
  int i,j;
    u,v;
    p[10], visited[10], result[10];
    s, min;
    u, v;
    k;
printf("Enter source: ");
scanf("%d", &n);
printf("\n Enter cost adjacency matrix:\n");
for(i=0; i<n; i++)
  for(j=0; j<n; j++){
          scanf("%d", &cost[i][j]);
  }
}
dijkstras(cost,n);

for(i=0; i<n; i++){
    d[i] = a[s][i];
    visited[i] = 0;
    p[i] = s;
}
visited[s] = 1;
for(i=0; i<n; i++){
    min = 999;
    u = 0;
    for(j=0; j<n; j++){
        if(visited[j] == 0){
            if(d[j] < min){
                min = d[j];
                u = j;
}
```

```
visited[u] = 1;
for(v=0; v<n; v++){
    if(visited[v]==0 && (d[u] + a[u][v] < d[v])){
        d[v] = d[u] + a[u][v];
        p[v] = u;
    }
}
}
printf("\n Shortest distance from single source to all other vertices:\n");
j=0;
for(i=1; i<n; i++){
    printf("%d to %d => distance = %d, \n", s, j, d[j]);
    j = j+1;
}
}
void main(){
    int i, j, n;
    int cost[10][10];
    printf("Enter no. of vertices: ");
    scanf("%d", &n);
    printf("Enter cost adjacency matrix:\n");
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            scanf("%d", &cost[i][j]);
        }
    }
    dijkstras(cost, n);
}
```

output

Enter the no of vertices: 5
Enter the cost adjacency matrix:

0	3	999	7	999
3	0	4	2	999
999	4	0	5	6
999	999	6	4	0
7	2	5	0	4

Enter source: 0
shortest distance from single source to all other vertices:

0 to 1 → distance = 0
0 to 2 → distance = 3
0 to 3 → distance = 7
0 to 4 → distance = 5

Date / /
Page 35

Fractional knapsack Problem

```
#include <stdio.h>
void swap (double a, double b, double temp){
    temp = a;
    a = b;
    b = temp;
}
void fractionalknapsack (int n, double weight[],
    double values[], double capacity){
    double valuePerweight[n];
    for(int i=0; i<n; i++){
        valuePerweight[i] = values[i]/weight[i];
    }
    for(int i=0; i<n-1; i++){
        for(int j=0; j<n-i-1; j++){
            if(valuePerweight[j]<valuePerweight[j+1]){
                double temp = weight[j];
                weight[j] = weight[j+1];
                weight[j+1] = temp;
                temp = values[j];
                values[j] = values[j+1];
                values[j+1] = temp;
                temp = valuePerweight[j];
                valuePerweight[j] = valuePerweight
                                    [j+1];
                valuePerweight[j+1] = temp;
            }
        }
    }
    double totalvalue = 0.0;
    double currentweight = 0.0;
```

```
for(int i=0; i<n; i++){
    if(currentweight + weight[i] <= capacity){
        currentweight += weights[i];
        totalvalue += values[i];
    }else{
        double remainingcapacity = capacity
                - currentweight;
        totalvalue += values[i] * (remainingCap
                -acity / weights[i]);
        break;
    }
}
printf("Maximum value in knapsack:%f\n",
                totalvalue);
}
int main(){
    int n;
    printf("Enter no. of item:");
    scanf("%d", &n);
    double weights[n], values[n], capacity;
    printf("Enter weights & values of each
                item:\n");
    for(int i=0; i<n; i++){
        scanf("%lf %lf", &weights[i],
                &values[i]);
    }
    printf("Enter capacity of knapsack:");
    scanf("%lf", &capacity);
    fractionalknapsack(n, weights, values,
                capacity);
    return 0;
}
```

output

Enter no of items : 7
Enter weight & values of each item
1 5
3 10
5 15
4 7
1 8
3 9
2 4

Enter the profit : 5 10 15 7 8 9 4
Maximum value in knapsack = 47.25
minimum value in knapsack = 46
profit / weight value in knapsack = 5.

nqueens

```
#include <stdio.h>
int place (int x[], int k) {
    for (int i = 0; i <k; i++)
    {
        if (x[i] == x[k] || i-x[i] == k-x[k] || i+x[i] ==
                                                k+x[k]) {

            return 0;
        }
    }
    return 1;
}

void nqueens(int n) {
    int x[10];
    int count = 0, s = 0;
    int k = 1;
    x[k] = 0;
    while (k != 0) {
        x[k] = x[k] + 1;
        while(x[k] <= n && !place (x, k)) {
            x[k] = x[k] + 1;
        }
        if (x[k] <= n) {
            if (k == n) {
                s = s + 1;
            }
            else {
                k++;
                x[k] = 0;
            }
        } else {
```

```
    } else {
        k--;
    }
  }
  printf("number of solutions: %d", s);
}

void main() {
    int n;
    printf("Enter number of n queens to be
                    placed: ");
    scanf("%d", &n);
    nqueens(n);
}
```

Output:

Enter the number of n queens to be placed: 4
number of solutions: 2