**UTS PERANCANGAN DAN ANALISA ALGORITMA**

Kelompok :

1. WINDY CHIKITA CORNIA PUTRI 06.2016.1.90457
2. GENA VIANTARI 06.2017.1.90467
3. STEVEN KUSUMA RAHARDJO 06.2017.1.90462

(KELAS H)

1. Bubble Sort

* Algoritma

Bubble sort : Penukaran data di sebelahnya secara terus menerus sampai bisa dipastikan dalam satu iterasi tertentu tidak ada perubahan.

Contoh : “4 2 4 3 9”

Tahapannya :

1. 4 2 5 3 9 -> 2 4 5 3 9

2 4 5 3 9 -> 2 4 5 3 9

2 4 5 3 9 -> 2 4 3 5 9

2 4 3 5 9 -> 2 4 3 5 9

1. 2 4 3 5 9 -> 2 4 3 5 9

2 4 3 5 9 -> 2 3 4 5 9

2 3 4 5 9 -> 2 3 4 5 9

2 3 4 5 9 -> 2 3 4 5 9

1. 2 3 4 5 9 -> 2 3 4 5 9

2 3 4 5 9 -> 2 3 4 5 9

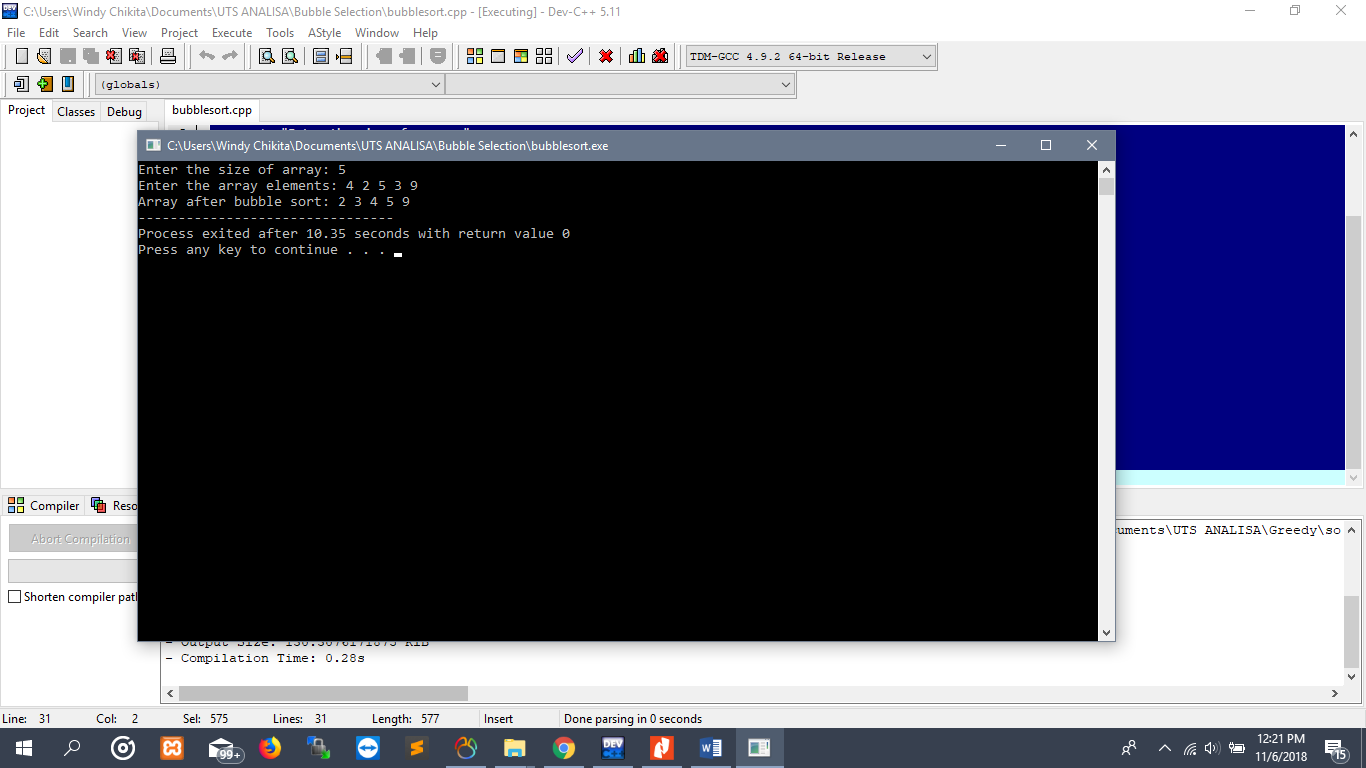
2 3 4 5 9 -> 2 3 4 5 9

2 3 4 5 9 -> 2 3 4 5 9

* Source Code



* Output Program



* Time Complexity

**BIG O : О(n2)**

Source Code

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

{

for(j=0;j<(n-i);++j)

if(a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

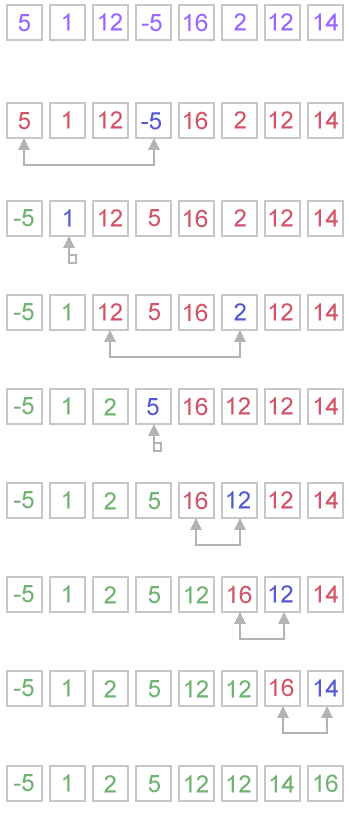
1. Selection Sort

* Algoritma

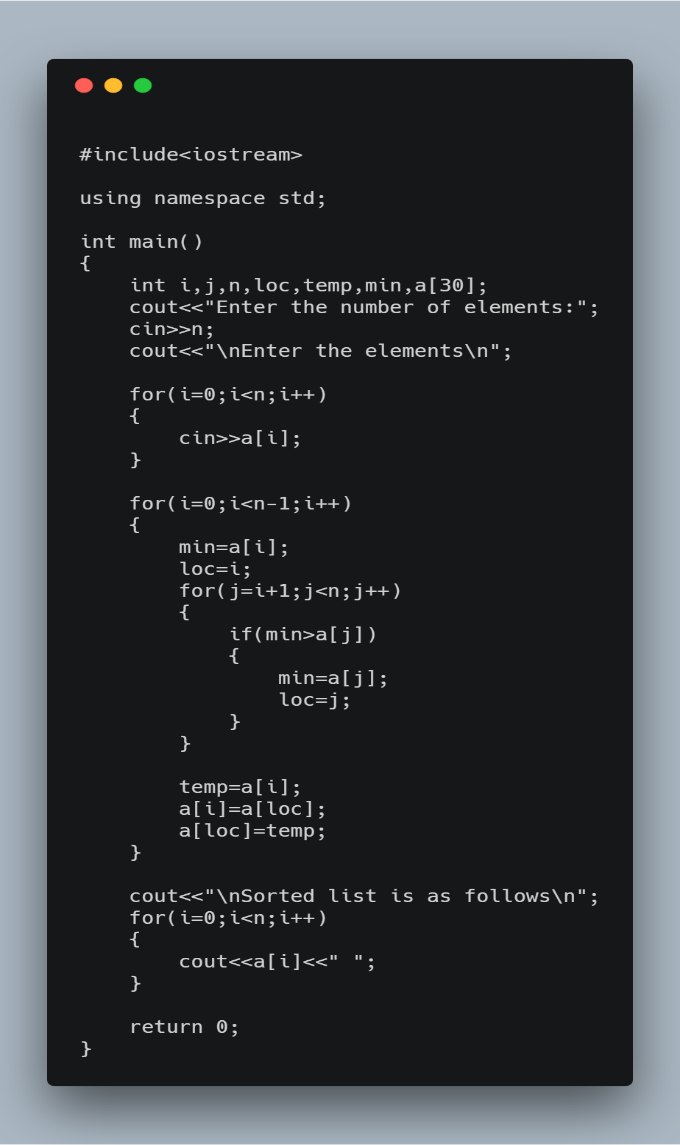
Selection sort : Mencari elemen yang tepat untuk diletakkan di posisi yang telah diketahui dan meletakkannya di posisi tersebut setelah data tersebut ditemukan.

Selection sort : Membandingkan elemen yang sekarang dengan elemen berikutnya sampai dengan elemen yang terakhir. Jika ditemukan elemen lain yang lebih kecil dari elemen sekarang maka dicatat posisinya dan kemudian ditukar.

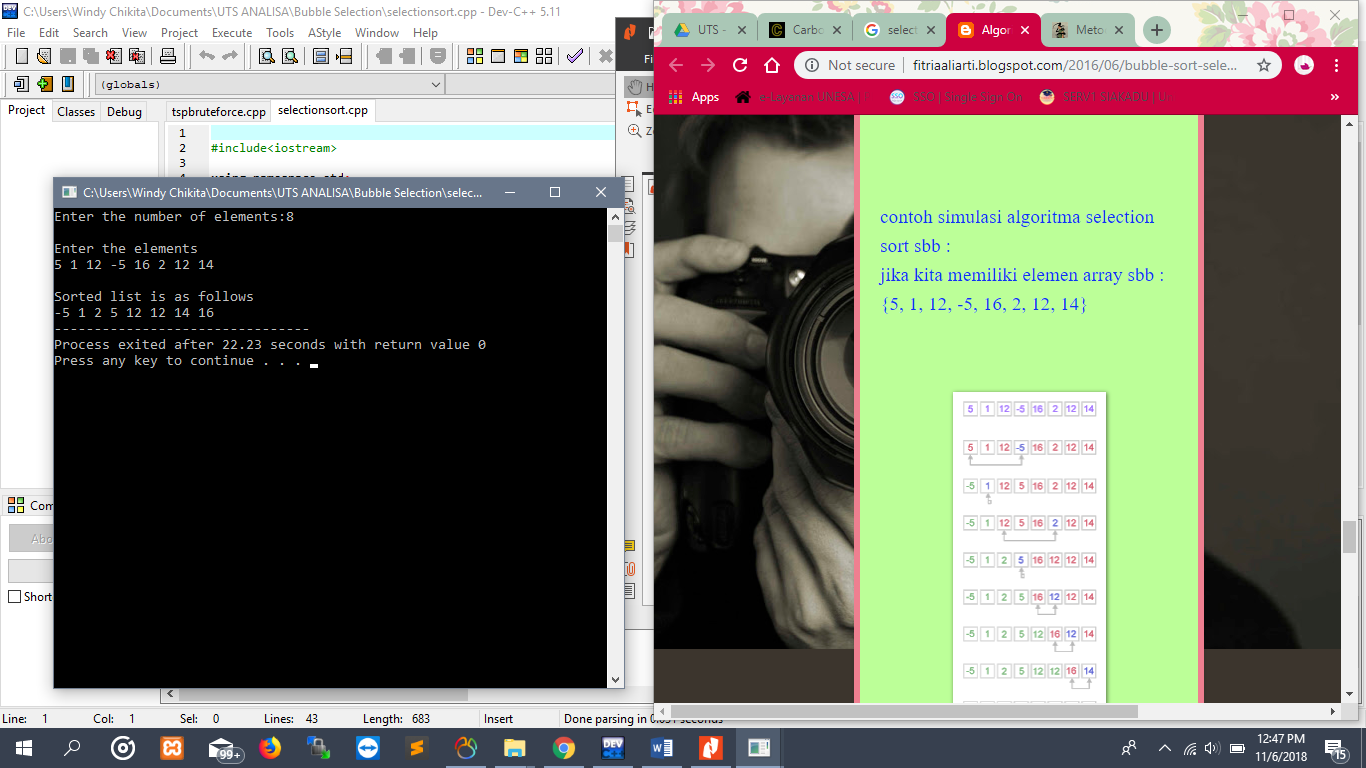
Contoh :



* Source Code



* Output Program



* Time Complexity

**BIG O : О(n2)**

Source Code

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

{

min=a[i];

loc=i;

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

{

if(min>a[j])

{

min=a[j];

loc=j;

}

}

temp=a[i];

a[i]=a[loc];

a[loc]=temp;

}

1. Penjadwalan

* Algoritma

Brute Force : Pendekatan problem solving berdasarkan pernyataan masalah dan definisi konsep yang dilibatkan. Strategi design yang paling sederhana dan mudah diaplikasikan.

Contoh :

Waktu Tunggu Pelanggan

Pelanggan 1 : 10 Pelanggan 4 : 8

Pelanggan 2 : 45 Pelanggan 5 : 12

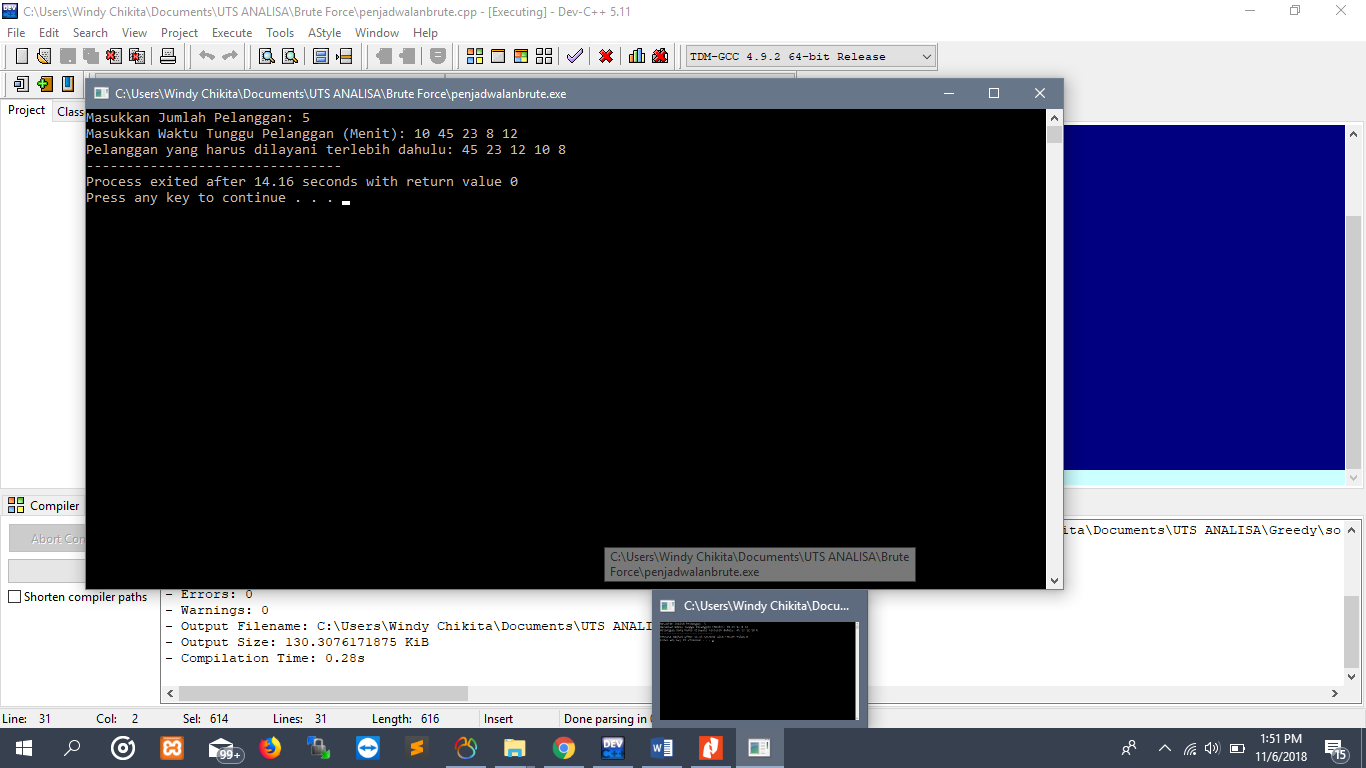
Pelanggan 3 : 23

Pelanggan yang harus dilayani : 2 -> 3 ->5->1->4

* Source Code



* Output Program



* Time Complexity

**BIG O : О(n2)**

Source Code

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

{

for(j=0;j<(n-i);++j)

if(a[j]<a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

1. Knapsack

* Algoritma

Brute Force : Pendekatan problem solving berdasarkan pernyataan masalah dan definisi konsep yang dilibatkan. Strategi design yang paling sederhana dan mudah diaplikasikan.

Contoh :

W1 = 20 P1 = 100

W2 = 10 P2 = 60

W3 = 30 P3 = 120

Kapasitas Maks = 50

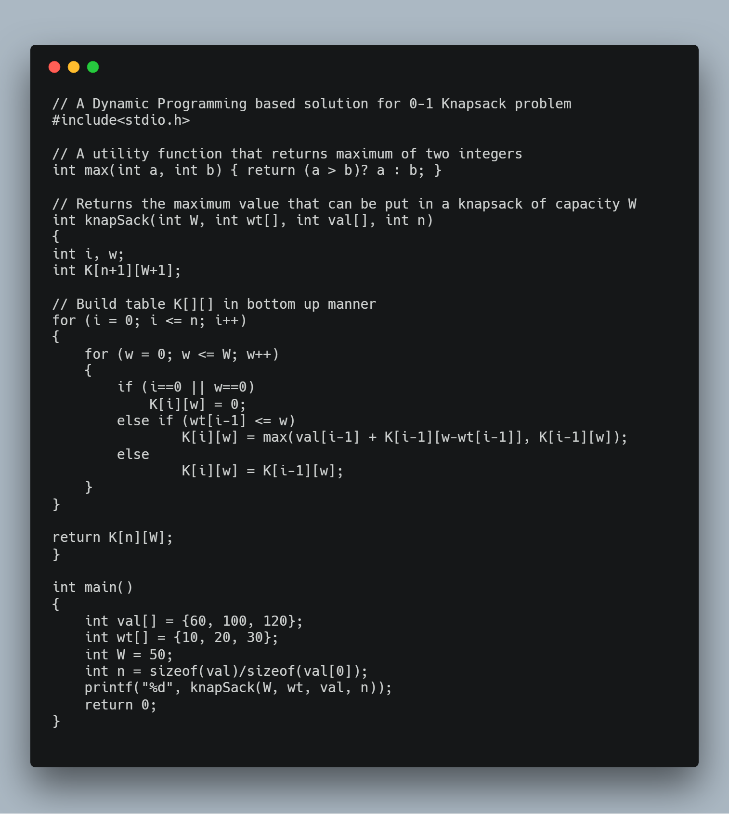
W = (20+10) P = (100+60)

W = (30+10) P = (120+60)

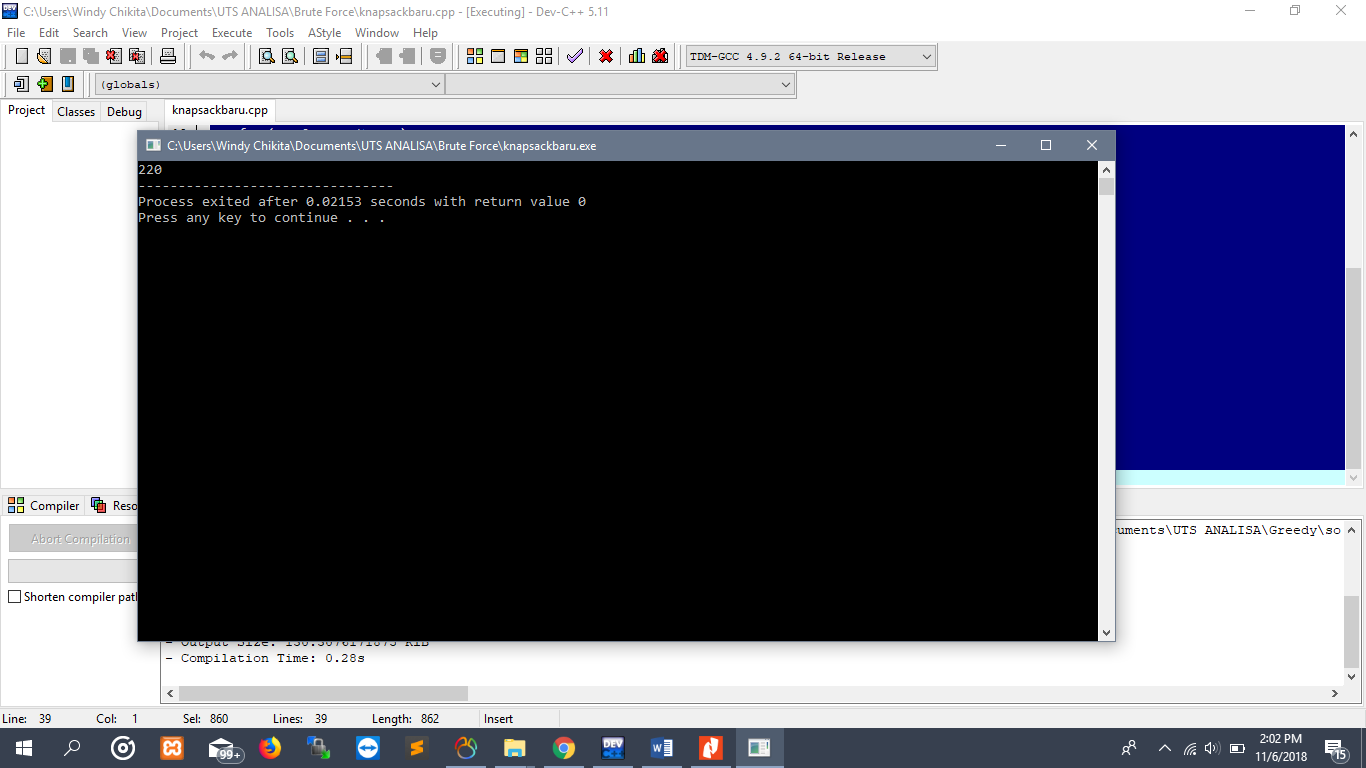
W = (30+20) P = (120+100)

W = (30+20, 120+100) >50

* Source Code



* Output Program



* Time Complexity

**BIG O : О(n2)**

Source Code

inFile >> n;

inFile >> W;

w = new int[n];

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

inFile >> w[i];

v = new int[n];

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

inFile >> v[i];

1. Penjadwalan

* Algoritma

Greedy : Prinsip “take what you can get now”, membentuk solusi langkah per langkah.

Contoh :

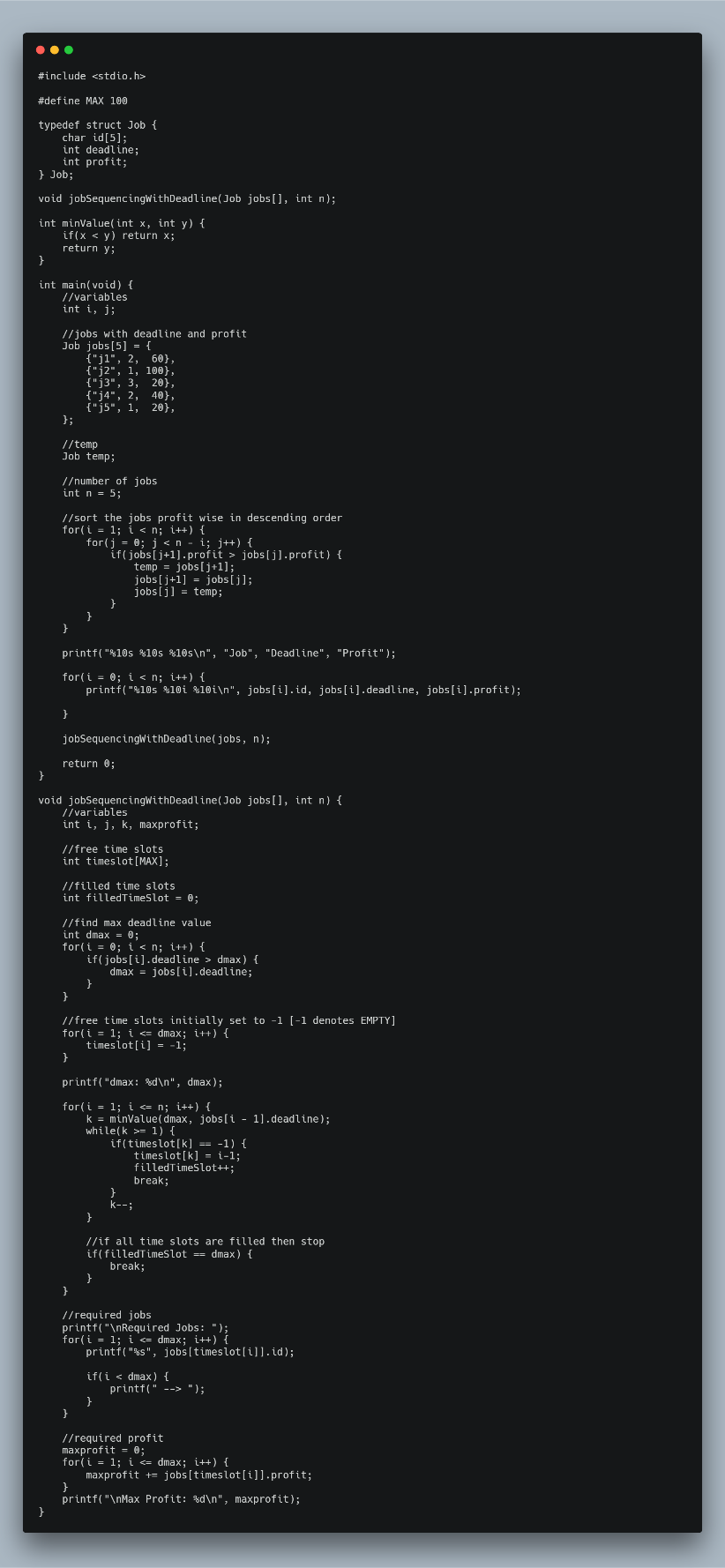
|  |  |  |
| --- | --- | --- |
| Pekerjaan | Deadline | Profit |
| 1 | 1 | 100 |
| 2 | 2 | 60 |
| 3 | 2 | 40 |
| 4 | 3 | 20 |
| 5 | 1 | 20 |

Kapasitas Maks = 3

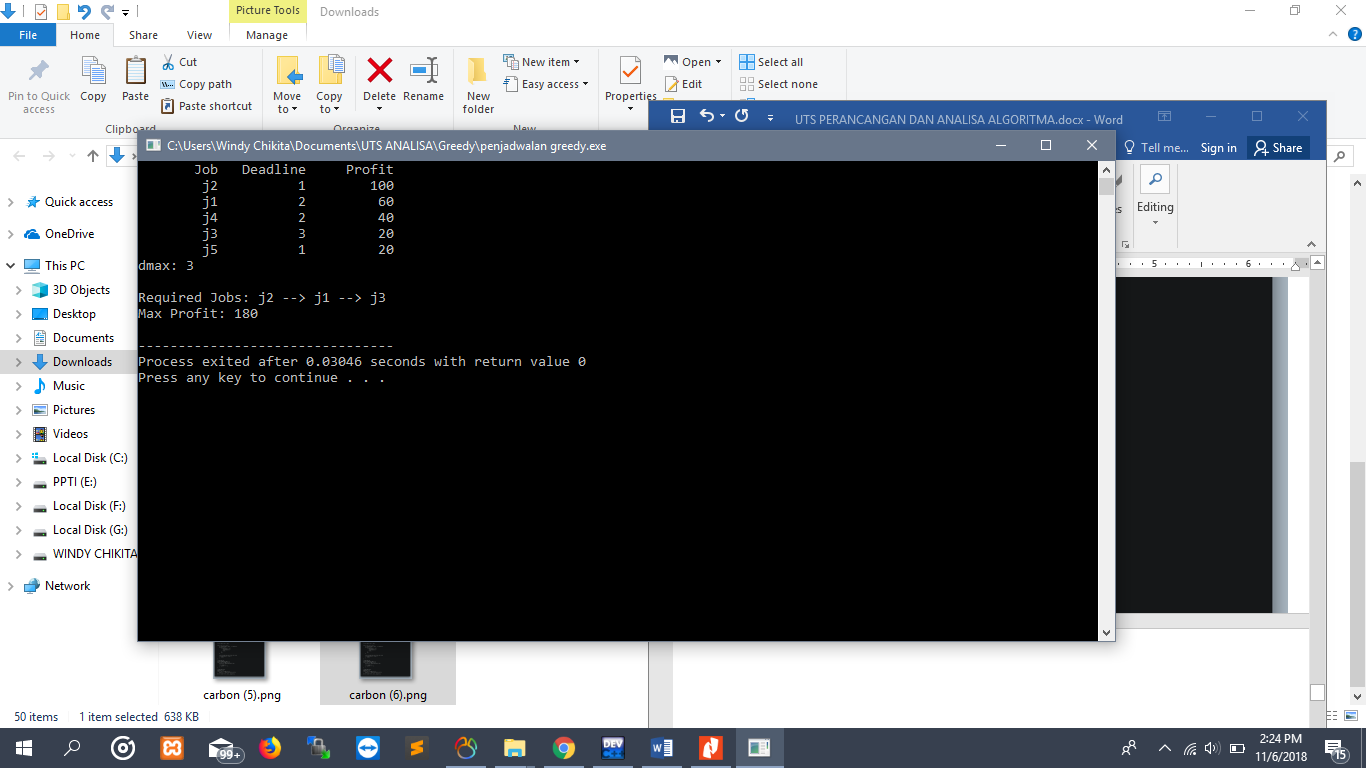
Pekerjaan yang harus didahulukan

2 -> 1 -> 3

Source Code



* Output Program



* Time Complexity

**BIG O : О(n2)**

Source Code

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

k = minValue(dmax, jobs[i - 1].deadline);

while(k >= 1) {

if(timeslot[k] == -1) {

timeslot[k] = i-1;

filledTimeSlot++;

break;

}

k--;

}

//if all time slots are filled then stop

if(filledTimeSlot == dmax) {

break;

}

}

1. Knapsack

* Algoritma

Greedy : Prinsip “take what you can get now”, membentuk solusi langkah per langkah.

Contoh :

W1 = 20 P1 = 100

W2 = 10 P2 = 60

W3 = 30 P3 = 120

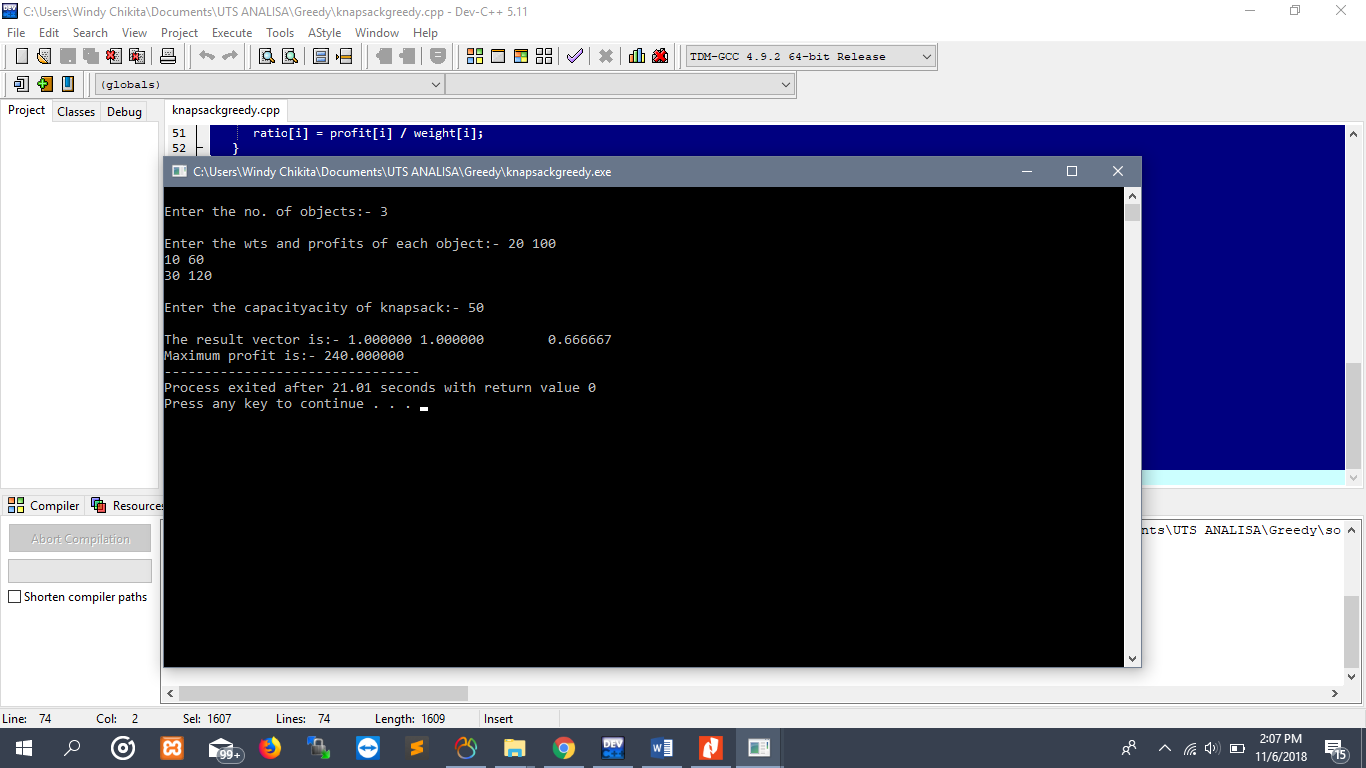
Kapasitas Maks = 50

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| I | Wi | Pi | Pi/wi | Profit | Weight | Density | Solusi |
| 1 | 20 | 100 | 5 | 1 | 1 | 1 | 1 |
| 2 | 10 | 60 | 6 | 0 | 0 | 1 | 0 |
| 3 | 30 | 120 | 4 | 1 | 1 | 0 | 1 |
| Total bobot | | | | 50 | 50 | 30 | 50 |
| Total Keuntungan | | | | 220 | 220 | 160 | 220 |

* Source Code

|  |
| --- |
| # include<stdio.h>  void knapsack(int n, float weight[], float profit[], float capacity) {  float x[20], tp = 0;  int i, j, u;  u = capacity;  for (i = 0; i < n; i++)  x[i] = 0.0;  for (i = 0; i < n; i++) {  if (weight[i] > u)  break;  else {  x[i] = 1.0;  tp = tp + profit[i];  u = u - weight[i];  }  }  if (i < n)  x[i] = u / weight[i];  tp = tp + (x[i] \* profit[i]);  printf("\nThe result vector is:- ");  for (i = 0; i < n; i++)  printf("%f\t", x[i]);  printf("\nMaximum profit is:- %f", tp);  }  int main() {  float weight[20], profit[20], capacity;  int num, i, j;  float ratio[20], temp;  printf("\nEnter the no. of objects:- ");  scanf("%d", &num);  printf("\nEnter the wts and profits of each object:- ");  for (i = 0; i < num; i++) {  scanf("%f %f", &weight[i], &profit[i]);  }  printf("\nEnter the capacityacity of knapsack:- ");  scanf("%f", &capacity);  for (i = 0; i < num; i++) {  ratio[i] = profit[i] / weight[i];  }  for (i = 0; i < num; i++) {  for (j = i + 1; j < num; j++) {  if (ratio[i] < ratio[j]) {  temp = ratio[j];  ratio[j] = ratio[i];  ratio[i] = temp;  temp = weight[j];  weight[j] = weight[i];  weight[i] = temp;  temp = profit[j];  profit[j] = profit[i];  profit[i] = temp;  }  }  }  knapsack(num, weight, profit, capacity);  return(0);  } |

* Output Program



* Time Complexity

**BIG O : О(n2)**

Source Code

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

x[i] = 0.0;

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

if (weight[i] > u)

break;

else {

x[i] = 1.0;

tp = tp + profit[i];

u = u - weight[i];

}

}

1. Shortest Path

* Algoritma

Greedy : Prinsip “take what you can get now”, membentuk solusi langkah per langkah.

Contoh :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| 0 | 0 | 12 | 5 | 10 |
| 1 | 12 | 0 | 8 | 9 |
| 2 | 5 | 8 | 0 | 15 |
| 3 | 10 | 9 | 15 | 0 |

Jarak Terpendek dari 0 adalah

0 -> 1 -> 0 = 12

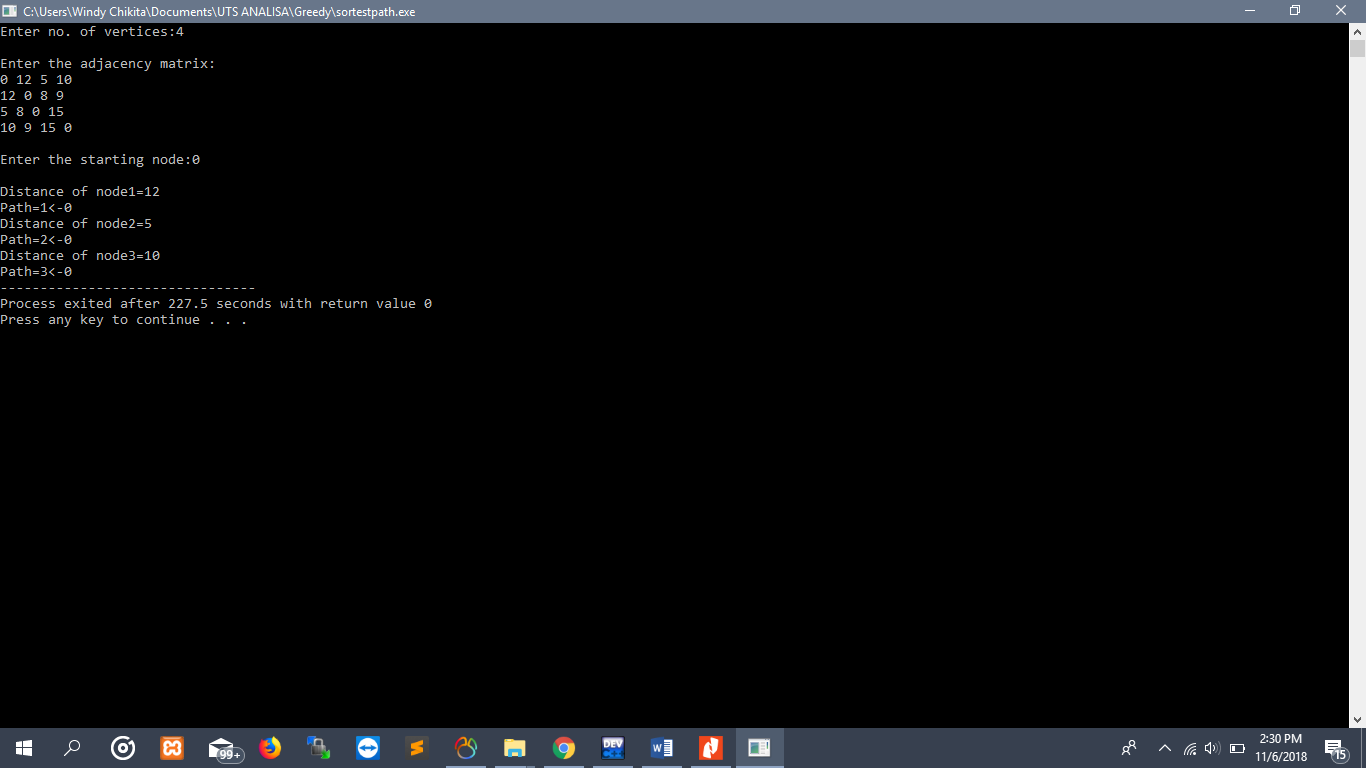
0 -> 2 -> 0 = 5

0 -> 3 -> 0 = 10

* Source Code

|  |
| --- |
| #include<stdio.h>  #include<conio.h>  #define INFINITY 9999  #define MAX 10    void dijkstra(int G[MAX][MAX],int n,int startnode);    int main()  {  int G[MAX][MAX],i,j,n,u;  printf("Enter no. of vertices:");  scanf("%d",&n);  printf("\nEnter the adjacency matrix:\n");    for(i=0;i<n;i++)  for(j=0;j<n;j++)  scanf("%d",&G[i][j]);    printf("\nEnter the starting node:");  scanf("%d",&u);  dijkstra(G,n,u);    return 0;  }    void dijkstra(int G[MAX][MAX],int n,int startnode)  {    int cost[MAX][MAX],distance[MAX],pred[MAX];  int visited[MAX],count,mindistance,nextnode,i,j;    //pred[] stores the predecessor of each node  //count gives the number of nodes seen so far  //create the cost matrix  for(i=0;i<n;i++)  for(j=0;j<n;j++)  if(G[i][j]==0)  cost[i][j]=INFINITY;  else  cost[i][j]=G[i][j];    //initialize pred[],distance[] and visited[]  for(i=0;i<n;i++)  {  distance[i]=cost[startnode][i];  pred[i]=startnode;  visited[i]=0;  }    distance[startnode]=0;  visited[startnode]=1;  count=1;    while(count<n-1)  {  mindistance=INFINITY;    //nextnode gives the node at minimum distance  for(i=0;i<n;i++)  if(distance[i]<mindistance&&!visited[i])  {  mindistance=distance[i];  nextnode=i;  }    //check if a better path exists through nextnode  visited[nextnode]=1;  for(i=0;i<n;i++)  if(!visited[i])  if(mindistance+cost[nextnode][i]<distance[i])  {  distance[i]=mindistance+cost[nextnode][i];  pred[i]=nextnode;  }  count++;  }    //print the path and distance of each node  for(i=0;i<n;i++)  if(i!=startnode)  {  printf("\nDistance of node%d=%d",i,distance[i]);  printf("\nPath=%d",i);    j=i;  do  {  j=pred[j];  printf("<-%d",j);  }while(j!=startnode);  }  } |

* Output Program



* Time Complexity

**BIG O : O(n2)**

Source Code

while(count<n-1)

{

mindistance=INFINITY;

//nextnode gives the node at minimum distance

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

if(distance[i]<mindistance&&!visited[i])

{

mindistance=distance[i];

nextnode=i;

}

//check if a better path exists through nextnode

visited[nextnode]=1;

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

if(!visited[i])

if(mindistance+cost[nextnode][i]<distance[i])

{

distance[i]=mindistance+cost[nextnode][i];

pred[i]=nextnode;

}

count++;

}

//print the path and distance of each node

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

if(i!=startnode)

{

printf("\nDistance of node%d=%d",i,distance[i]);

printf("\nPath=%d",i);

j=i;

do

{

j=pred[j];

printf("<-%d",j);

}while(j!=startnode);

}