a. Consider that “ST Computer” is a printing shop. In this shop there are five price range for printing pages. Assume that the shop max printing capacity per day P = 1000 pages. Our target is to make maximum profit every day.

Item(i)	1	2	3	4	5
Pages(pg)	200	300	400	500	700
Price	200	250	350	400	550

b.

```
#include<stdio.h>

void knapSack(int P, int n, int price[], int pg[]);
int getMax(int x, int y);

int main(void)
{

  int price[] = {200,250,350,400,550};
  int pg[] = {200,300,400,500,700};

  int n = 5;
  int P = 1000;

  knapSack(P, n, price, pg);

  return 0;
}

int getMax(int x, int y)
{
  if(x > y)
  {
    return x;
  }
  else
```

```
  {
    return y;
  }
}

void knapSack(int P, int n, int price[], int pg[]) {
  int i, p;
  int N[n+1][P+1];

  for(p = 0; p <= P; p++)
  {
    N[0][p] = 0;
  }

  for(i = 0; i <= n; i++)
  {
    N[i][0] = 0;
  }

  for(i = 1; i <= n; i++)
  {
    for(p = 1; p <= P; p++)
    {
      if(pg[i] <= p)
      {
        N[i][p] = getMax(N[i-1][p], price[i] + N[i-1][p - pg[i]]);
      }
      else
      {
        N[i][p] = N[i-1][p];
      }
    }
  }

  printf("Maximum earn: %d\n", N[n][P]);
}
```

c.

The worst-case time complexity of 0/1 knapsack algorithm is **O(N*W)**. N represent capacity and W represent the value of object.

d.

Dynamic programming is an effective method for fixing problems. Dynamic programming works through solving subproblems and using the results of those subproblems to extra quickly calculate the solution to a bigger problem. The divide-and-conquer paradigm (which additionally makes use of the concept of solving subproblems), dynamic programming usually involves solving all possible subproblems instead of a small component. One use of dynamic programming is the problem of 0/1 knapsack. In this dynamic programming problem, we've n objects each with a related pages and charges. The goal is to fill the knapsack with objects such that we've a maximum price without crossing the page limit of the knapsack. Dynamic programming produces a simpler algorithm. The key point to eliminate is that the using dynamic programming, we will reduce the problems of finding all of the shortest paths to fixing a series of subproblems that can be reused again and again to resolve large problems. Every time we attempt to solve a problem using dynamic programming.

e. Dry run

1st phase

for(i=1; i<=5; i++)
  for(p=200; p<=1000; p++)
if(pg[1]<=200)
N[1][200] = getMax(N[0][200], price[1] + N[0][200 - pg[1]);
= N[0][200] = 0, 200 + N[0][0]
= getMax(200)
N[1][200] = 200

for(i=1; i<=5; i++)
  for(p=300; p<=1000; p++)
if(pg[1]<=300)
N[1][300] = getMax(N[1-1][300], price[1] + N[1-1][300 - pg[1]);
= N[0][300] = 0, 200 + N[0][300-200]
= N[0][300] = 0, 200 + N[0][100]
= getMax(200)
N[1][300] = 200

for(i=1; i<=5; i++)
  for(p=400; p<=1000; p++)
if(pg[1]<=400)
N[1][400] = getMax(N[1-1][400], price[1] + N[1-1][400 - pg[1]);
= N[0][400] = 0, 200 + N[0][400-200]
= N[0][400] = 0, 200 + N[0][200]
= getMax(200)
N[1][400] = 200

for(i=1; i<=5; i++)
  for(p=500; p<=1000; p++)
if(pg[1]<=500)
N[1][500] = getMax(N[1-1][500], price[1] + N[1-1][500 - pg[1]);
= N[0][500] = 0, 200 + N[0][500-200]
= N[0][500] = 0, 200 + N[0][300]
= getMax(200)
N[1][500] = 200

for(i=1; i<=5; i++)

  for(p=700; p<=1000; p++)

if(pg[1]<=700)

N[1][700] = getMax(N[0][700], price[1] + N[0][700 - pg[1]);

= N[0][700] = 0, 200 + N[0][700-200]

= N[0][500] = 0, 200 + N[0][500]

= getMax(200)

N[1][700] = 200

2nd phase

for(i=2; i<=5; i++)

  for(p=200; p<=1000; p++)

if(pg[2]<=200)

N[2][200] = getMax(N[2-1][200], price[2] + N[2-1][200 - pg[2]);

= N[1][200] = 200, 250 + N[1][200-300]

= N[1][200] = 200, 250 + N[1][-100]

= getMax(200)

N[2][200] = 200

for(i=2; i<=5; i++)

  for(p=300; p<=1000; p++)

if(pg[2]<=300)

N[2][300] = getMax(N[2-1][300], price[2] + N[2-1][300 - pg[2]);

= N[1][300] = 0, 250 + N[1][300-300]

= N[1][300] = 200, 250 + N[1][0]

= getMax(250)

N[2][300] = 250

for(i=2; i<=5; i++)

for(p=400; p<=1000; p++)

if(pg[2]<=400)

N[2][400] = getMax(N[2-1][400], price[2] + N[2-1][400 - pg[2]);

= N[1][400] = 200, 250 + N[1][400-300]

= N[1][400] = 200, 250 + N[1][100]

= getMax(250)

N[2][400] = 250

for(i=2; i<=5; i++)

for(p=500; p<=1000; p++)

if(pg[2]<=500)

N[2][500] = getMax(N[2-1][500], price[2] + N[2-1][500 - pg[2]);

= N[1][500] = 200, 250 + N[1][500-300]

= N[1][500] = 200, 250 + N[1][200] ***[N[1][200] = 200]***

= getMax(450)

N[2][500] = 450

for(i=2; i<=5; i++)

for(p=700; p<=1000; p++)

if(pg[2]<=700)

N[2][700] = getMax(N[2-1][700], price[2] + N[2-1][700 - pg[2]);

= N[1][700] = 200, 250 + N[0][700-300]

= N[1][500] = 200, 250 + N[1][400] ***[N[1][400] = 200]***

= getMax(450)

N[2][700] = 450

N[i,p]	p=0	200	300	400	500	600	700	800	900	1000
i= 0	0	0	0	0	0	0	0	0	0	0
1	0	200	200	200	200	200	200	200	200	200
2	0	200	250	250	450	450	450	450	450	450
3	0	200	250	350	350	550	600	600	800	800
4	0	200	250	350	400	400	600	650	750	850
5	0	200	250	350	400	400	500	500	750	800