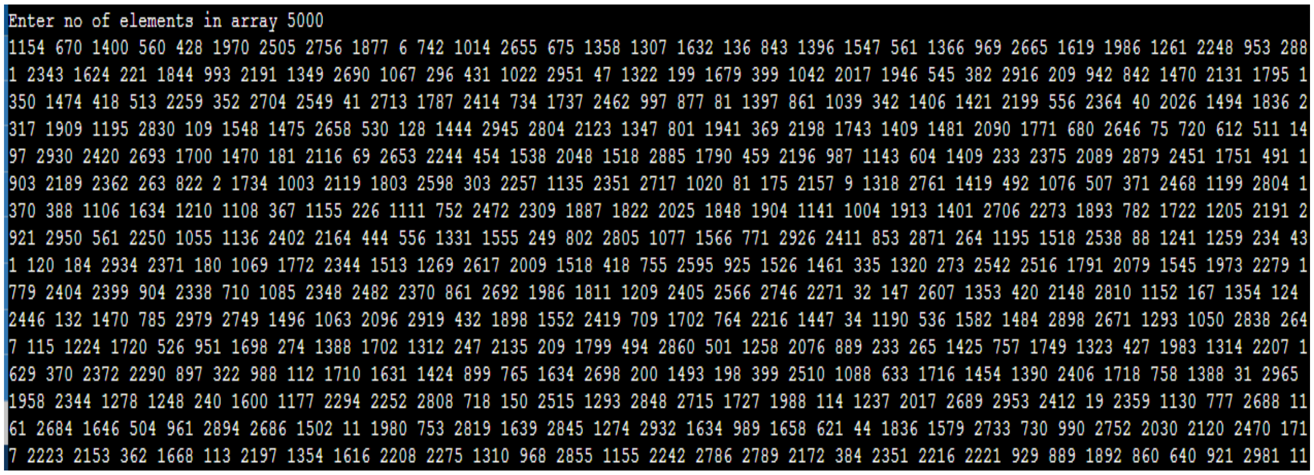
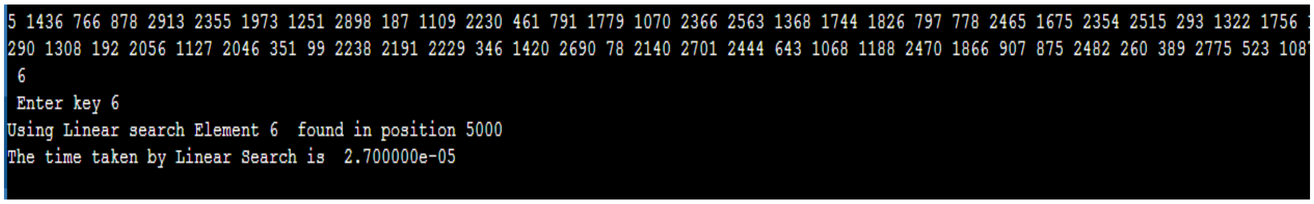
Linear Search

```
#include <stdio.h>
#include <time.h>
#include<math.h>
clock_t st,et;
double ts;
int n,i,key,locat,a[10000];

 int linear_find(int a[],int n,int key)
   {
      int i;
       for(i=0;i<n;i++)
       {
          if( key== a[i])
          locat=i+1;
       }
       if (locat==0)
          printf("Using Linear Search Element %d Not Found \n", key);
       else
          printf("Using Linear search Element %d  found in position %d\n",key,locat);

   }

int main()
{
   int r1=0,r2=3000;
   printf("Enter no of elements in array");
   scanf("%d",&n);
   for(i=0;i<n;i++)
           {
                    a[i]=(rand() % (r2 - r1 + 1)) + r1;
                     printf("%d ",a[i]);

           }
   printf("\n Enter key");
   scanf("%d",&key);

   st= clock();
   linear_find( a,n,key);
   et= clock();
   ts= (double) (et-st)/CLOCKS_PER_SEC;
   printf("The time taken by Linear Search is  %e\n", ts);

}
```



```
Enter no of elements in array 5000
1154 670 1400 560 428 1970 2505 2756 1877 6 742 1014 2655 675 1358 1307 1632 136 843 1396 1547 561 1366 969 2665 1619 1986 1261 2248 953 288
1 2343 1624 221 1844 993 2191 1349 2690 1067 296 431 1022 2951 47 1322 199 1679 399 1042 2017 1946 545 382 2916 209 942 842 1470 2131 1795 1
350 1474 418 513 2259 352 2704 2549 41 2713 1787 2414 734 1737 2462 997 877 81 1397 861 1039 342 1406 1421 2199 556 2364 40 2026 1494 1836 2
317 1909 1195 2830 109 1548 1475 2658 530 128 1444 2945 2804 2123 1347 801 1941 369 2198 1743 1409 1481 2090 1771 680 2646 75 720 612 511 14
97 2930 2420 2693 1700 1470 181 2116 69 2653 2244 454 1538 2048 1518 2885 1790 459 2196 987 1143 604 1409 233 2375 2089 2879 2451 1751 491 1
903 2189 2362 263 822 2 1734 1003 2119 1803 2598 303 2257 1135 2351 2717 1020 81 175 2157 9 1318 2761 1419 492 1076 507 371 2468 1199 2804 1
370 388 1106 1634 1210 1108 367 1155 226 1111 752 2472 2309 1887 1822 2025 1848 1904 1141 1004 1913 1401 2706 2273 1893 782 1722 1205 2191 2
921 2950 561 2250 1055 1136 2402 2164 444 556 1331 1555 249 802 2805 1077 1566 771 2926 2411 853 2871 264 1195 1518 2538 88 1241 1259 234 43
1 120 184 2934 2371 180 1069 1772 2344 1513 1269 2617 2009 1518 418 755 2595 925 1526 1461 335 1320 273 2542 2516 1791 2079 1545 1973 2279 1
779 2404 2399 904 2338 710 1085 2348 2482 2370 861 2692 1986 1811 1209 2405 2566 2746 2271 32 147 2607 1353 420 2148 2810 1152 167 1354 124
2446 132 1470 785 2979 2749 1496 1063 2096 2919 432 1898 1552 2419 709 1702 764 2216 1447 34 1190 536 1582 1484 2898 2671 1293 1050 2838 264
7 115 1224 1720 526 951 1698 274 1388 1702 1312 247 2135 209 1799 494 2860 501 1258 2076 889 233 265 1425 757 1749 1323 427 1983 1314 2207 1
629 370 2372 2290 897 322 988 112 1710 1631 1424 899 765 1634 2698 200 1493 198 399 2510 1088 633 1716 1454 1390 2406 1718 758 1388 31 2965
1958 2344 1278 1248 240 1600 1177 2294 2252 2808 718 150 2515 1293 2848 2715 1727 1988 114 1237 2017 2689 2953 2412 19 2359 1130 777 2688 11
61 2684 1646 504 961 2894 2686 1502 11 1980 753 2819 1639 2845 1274 2932 1634 989 1658 621 44 1836 1579 2733 730 990 2752 2030 2120 2470 171
7 2223 2153 362 1668 113 2197 1354 1616 2208 2275 1310 968 2855 1155 2242 2786 2789 2172 384 2351 2216 2221 929 889 1892 860 640 921 2981 11
```



```
5 1436 766 878 2913 2355 1973 1251 2898 187 1109 2230 461 791 1779 1070 2366 2563 1368 1744 1826 797 778 2465 1675 2354 2515 293 1322 1756
290 1308 192 2056 1127 2046 351 99 2238 2191 2229 346 1420 2690 78 2140 2701 2444 643 1068 1188 2470 1866 907 875 2482 260 389 2775 523 108
 6
 Enter key 6
Using Linear search Element 6  found in position 5000
The time taken by Linear Search is  2.700000e-05
```

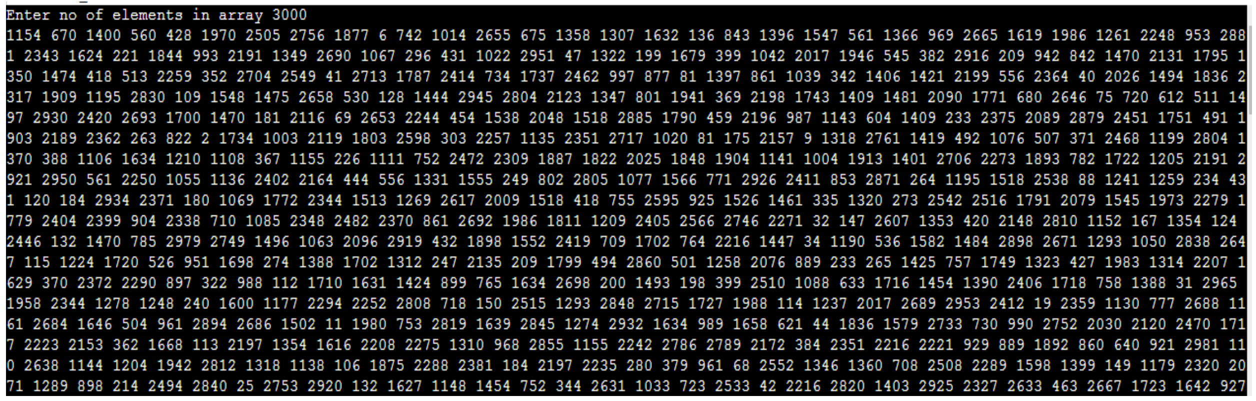
Binary Search

```
#include <stdio.h>
#include <time.h>
#include<math.h>
clock_t st,et;
double ts;
int i,n,key,locat,a[10000];
int beg=0,end,mid,position;

 void binary_search(int a[],int n, int key) {
        int f = 0, r = n,mid;

   while (f <= r)
    {
            mid = (f+r)/2;
            if (a[mid] == key)
                        {
                       printf("\nUsing Binary search Element %d  found in position %d\n", key, mid+1);
                       break;
                        }
          else if (a[mid] < key)
                        f = mid + 1;
          else
                         r = mid - 1;
     }

   if (f > r)
      printf("\nUsing Binary search Element %d Not Found \n", key);
}
```

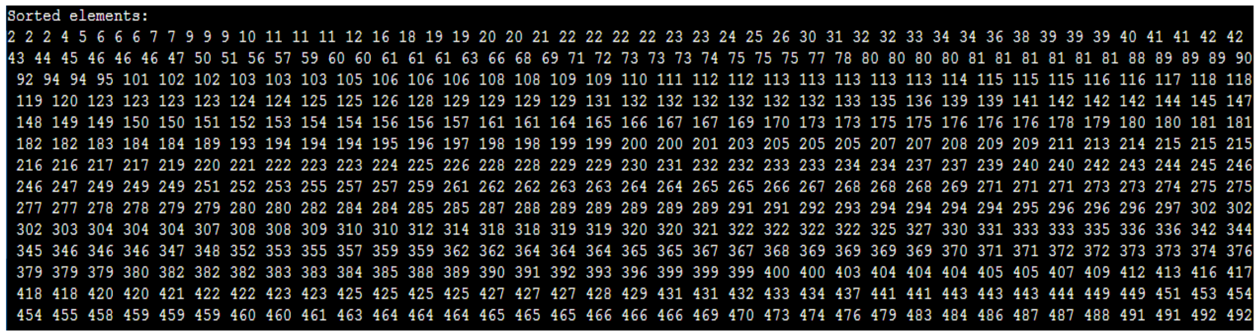
```
 int cmpfunc (const void * a, const void * b) {
   return ( *(int*)a - *(int*)b );
}

int main()
{
    int r1=0,r2=300;
    printf("Enter no of elements in array");
    scanf("%d",&n);

    //printf("Enter elements of Array");
    //for(i=0;i<n;i++)
    //scanf("%d",&a[i]);

    for(i=0;i<n;i++)
        {
          a[i]=(rand() % (r2 - r1 + 1)) + r1;
          qsort(a, n, sizeof(int),cmpfunc);
          printf( "%d  ", a[i]);
        }

    printf("\n Enter key");
    scanf("%d",&key);

    st= clock();
    binary_search(a,n,key);
    et= clock();
    ts= (double) (et-st)/CLOCKS_PER_SEC;
    printf("The time taken by Binary Search is  %e\n", ts);
}
```

Quick Sort

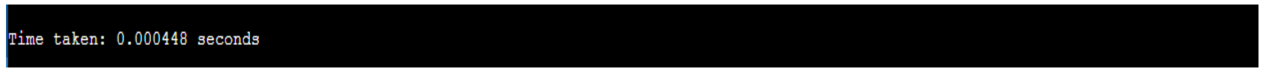
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function to swap two elements
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Function to partition the array
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }

    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}

// Function to perform Quick Sort
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivotIndex = partition(arr, low, high);
        quickSort(arr, low, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, high);
    }
}

int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    // Generate random elements
    int arr[n];
    srand(time(0));
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000;  // Random numbers between 0 and 99
    }

    // Print the unsorted array
    printf("Unsorted array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");



    // Calculate the time taken for sorting
    clock_t start = clock();
    quickSort(arr, 0, n - 1);
    clock_t end = clock();
    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;

    printf("\nSorted elements:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    printf("\nTime taken: %lf seconds\n", time_taken);

    return 0;
}
```



```
Enter no of elements in array 3000
1154 670 1400 560 428 1970 2505 2756 1877 6 742 1014 2655 675 1358 1307 1632 136 843 1396 1547 561 1366 969 2665 1619 1986 1261 2248 953 288
1 2343 1624 221 1844 993 2191 1349 2690 1067 296 431 1022 2951 47 1322 199 1679 399 1042 2017 1946 545 382 2916 209 942 842 1470 2131 1795 1
350 1474 418 513 2259 352 2704 2549 41 2713 1787 2414 734 1737 2462 997 877 81 1397 861 1039 342 1406 1421 2199 556 2364 40 2026 1494 1836 2
317 1909 1195 2830 109 1548 1475 2658 530 128 1444 2945 2804 2123 1347 801 1941 369 2198 1743 1409 1481 2090 1771 680 2646 75 720 612 511 14
97 2930 2420 2693 1700 1470 181 2116 69 2653 2244 454 1538 2048 1518 2885 1790 459 2196 987 1143 604 1409 233 2375 2089 2879 2451 1751 491 1
903 2189 2362 263 822 2 1734 1003 2119 1803 2598 303 2257 1135 2351 2717 1020 81 175 2157 9 1318 2761 1419 492 1076 507 371 2468 1199 2804 1
370 388 1106 1634 1210 1108 367 1155 226 1111 752 2472 2309 1887 1822 2025 1848 1904 1141 1004 1913 1401 2706 2273 1893 782 1722 1205 2191 2
921 2950 561 2250 1055 1136 2402 2164 444 556 1331 1555 249 802 2805 1077 1566 771 2926 2411 853 2871 264 1195 1518 2538 88 1241 1259 234 43
1 120 184 2934 2371 180 1069 1772 2344 1513 1269 2617 2009 1518 418 755 2595 925 1526 1461 335 1320 273 2542 2516 1791 2079 1545 1973 2279 1
779 2404 2399 904 2338 710 1085 2348 2482 2370 861 2692 1986 1811 1209 2405 2566 2746 2271 32 147 2607 1353 420 2148 2810 1152 167 1354 124
2446 132 1470 785 2979 2749 1496 1063 2096 2919 432 1898 1552 2419 709 1702 764 2216 1447 34 1190 536 1582 1484 2898 2671 1293 1050 2838 264
7 115 1224 1720 526 951 1698 274 1388 1702 1312 247 2135 209 1799 494 2860 501 1258 2076 889 233 265 1425 757 1749 1323 427 1983 1314 2207 1
629 370 2372 2290 897 322 988 112 1710 1631 1424 899 765 1634 2698 200 1493 198 399 2510 1088 633 1716 1454 1390 2406 1718 758 1388 31 2965
1958 2344 1278 1248 240 1600 1177 2294 2252 2808 718 150 2515 1293 2848 2715 1727 1988 114 1237 2017 2689 2953 2412 19 2359 1130 777 2688 11
61 2684 1646 504 961 2894 2686 1502 11 1980 753 2819 1639 2845 1274 2932 1634 989 1658 621 44 1836 1579 2733 730 990 2752 2030 2120 2470 171
7 2223 2153 362 1668 113 2197 1354 1616 2208 2275 1310 968 2855 1155 2242 2786 2789 2172 384 2351 2216 2221 929 889 1892 860 640 921 2981 11
0 2638 1144 1204 1942 2812 1318 1138 106 1875 2288 2381 184 2197 2235 280 379 961 68 2552 1346 1360 708 2508 2289 1598 1399 149 1179 2320 20
71 1289 898 214 2494 2840 25 2753 2920 132 1627 1148 1454 752 344 2631 1033 723 2533 42 2216 2820 1403 2925 2327 2633 463 2667 1723 1642 927
```



```
Sorted elements:
2 2 2 4 5 6 6 6 7 7 9 9 9 10 11 11 11 12 16 18 19 19 20 20 21 22 22 22 22 23 23 24 25 26 30 31 32 32 33 34 34 36 38 39 39 39 40 41 41 42 42
43 44 45 46 46 46 47 50 51 56 57 59 60 60 61 61 61 63 66 68 69 71 72 73 73 73 74 75 75 75 77 78 80 80 80 80 81 81 81 81 81 81 88 89 89 89 90
 92 94 94 95 101 102 102 103 103 103 105 106 106 106 108 108 109 109 110 111 112 112 113 113 113 113 113 114 115 115 115 116 116 117 118 118
 119 120 123 123 123 123 124 124 125 125 126 128 129 129 129 129 131 132 132 132 132 132 132 133 135 136 139 139 141 142 142 142 144 145 147
 148 149 149 150 150 151 152 153 154 154 156 156 157 161 161 164 165 166 167 167 169 170 173 173 175 175 176 176 176 178 179 180 180 181 181
 182 182 183 184 184 189 193 194 194 194 195 196 197 198 198 199 199 200 200 201 203 205 205 205 207 207 208 209 209 211 213 214 215 215 215
 216 216 217 217 219 220 221 222 223 223 224 225 226 228 228 229 229 230 231 232 232 233 233 234 234 237 237 239 240 240 242 243 244 245 246
 246 247 249 249 249 251 252 253 255 257 257 259 261 262 262 263 263 264 264 265 265 266 267 268 268 268 269 271 271 271 273 273 274 275 275
 277 277 278 278 279 279 280 280 282 284 284 285 285 287 288 289 289 289 289 289 291 291 292 293 294 294 294 294 295 296 296 296 297 302 302
 302 303 304 304 304 307 308 308 309 310 310 312 314 318 318 319 319 320 320 321 322 322 322 322 325 327 330 331 333 333 335 336 336 342 344
 345 346 346 346 347 348 352 353 355 357 359 359 362 362 364 364 364 365 365 367 367 368 369 369 369 369 370 371 371 372 372 373 373 374 376
 379 379 379 380 382 382 382 383 383 384 385 388 389 390 391 392 393 396 399 399 399 400 400 403 404 404 404 405 405 407 409 412 413 416 417
 418 418 420 420 421 422 422 423 423 425 425 425 425 427 427 427 428 429 431 431 432 433 434 437 441 441 443 443 443 444 449 449 451 453 454
 454 455 458 459 459 459 460 460 461 463 464 464 464 465 465 465 466 466 466 469 470 473 474 476 479 483 484 486 487 487 488 491 491 492 492
```



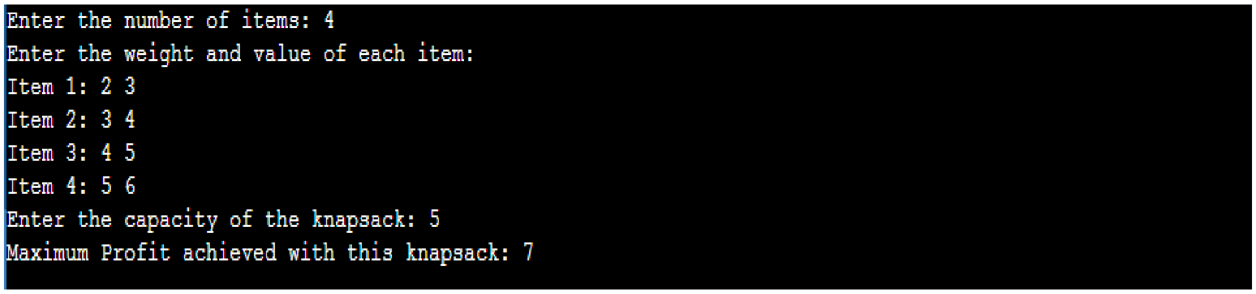
```
Time taken: 0.000448 seconds
```

Merge Sort

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

// Function to merge two subarrays
void merge(int arr[], int left[], int right[], int left_size, int right_size) {
    int i = 0, j = 0, k = 0;

    // Merge elements from left[] and right[] back into arr[]
    while (i < left_size && j < right_size) {
        if (left[i] <= right[j]) {
            arr[k] = left[i];
            i++;
        } else {
            arr[k] = right[j];
            j++;
        }
        k++;
    }
    // Copy the remaining elements of left[], if any
    while (i < left_size) {
        arr[k] = left[i];
        i++;
        k++;
    }

    // Copy the remaining elements of right[], if any
    while (j < right_size) {
        arr[k] = right[j];
        j++;
        k++;
    }
}

// Merge Sort function
void mergeSort(int arr[], int size) {
    if (size < 2) {
        return;  // Base case: array is already sorted
    }

    int mid = size / 2;
    int left[mid];
    int right[size - mid];

    // Split the array into two subarrays
    for (int i = 0; i < mid; i++) {
        left[i] = arr[i];
    }
    for (int i = mid; i < size; i++) {
        right[i - mid] = arr[i];
    }

    // Recursive calls to mergeSort() for the subarrays
    mergeSort(left, mid);
    mergeSort(right, size - mid);

    // Merge the sorted subarrays
    merge(arr, left, right, mid, size - mid);
}

int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    // Generate random elements
    int arr[n];
    srand(time(0));
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 100;  // Random numbers between 0 and 99
    }

    // Print the unsorted array
    printf("Unsorted array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Perform Merge Sort
    clock_t start_time = clock();
    mergeSort(arr, n);
    clock_t end_time = clock();

    // Print the sorted array
    printf("Sorted array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Calculate and print the time taken
    double time_taken = (double)(end_time - start_time) / CLOCKS_PER_SEC;
    printf("Time taken: %.6fs\n", time_taken);

    return 0;
}
```

Dynamic Programming method

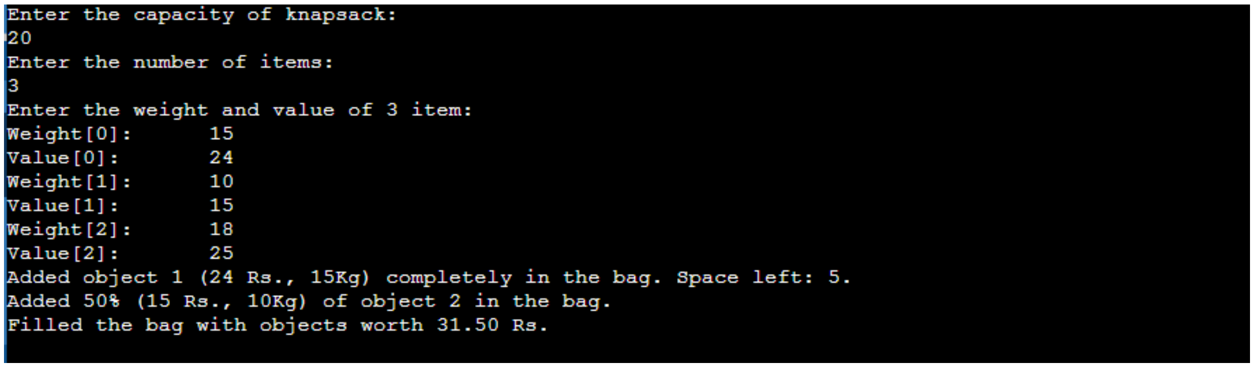
```
#include <stdio.h>
#include <string.h>
int findMax(int n1, int n2){
   if(n1>n2) {
      return n1;
   } else {
      return n2;
   }
}
int knapsack(int W, int wt[], int val[], int n){
   int K[n+1][W+1];
   for(int i = 0; i<=n; i++) {
      for(int w = 0; w<=W; w++) {
         if(i == 0 || w == 0) {
            K[i][w] = 0;
         } else if(wt[i-1] <= w) {
            K[i][w] = findMax(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);
         } else {
            K[i][w] = K[i-1][w];
         }
      }
   }
   return K[n][W];
}
int main(){
   int n, W, i;

    printf("Enter the number of items: ");
    scanf("%d", &n);

    int val[n], wt[n];

    printf("Enter the weight and value of each item:\n");
    for (i = 0; i < n; i++) {
        printf("Item %d: ", i + 1);
        scanf("%d%d", &wt[i], &val[i]);
    }

    printf("Enter the capacity of the knapsack: ");
    scanf("%d", &W);

   int len = sizeof val / sizeof val[0];
   printf("Maximum Profit achieved with this knapsack: %d", knapsack(W, wt, val, len));
}
```



```
Enter the number of items: 4
Enter the weight and value of each item:
Item 1: 2 3
Item 2: 3 4
Item 3: 4 5
Item 4: 5 6
Enter the capacity of the knapsack: 5
Maximum Profit achieved with this knapsack: 7
```

Greedy Method

```
#include <stdio.h>

void main()
{
    int capacity, no_items, cur_weight, item;
    int used[10];
    float total_profit;
    int i;
    int weight[10];
    int value[10];

    printf("Enter the capacity of knapsack:\n");
    scanf("%d", &capacity);

    printf("Enter the number of items:\n");
    scanf("%d", &no_items);
    printf("Enter the weight and value of %d item:\n", no_items);
    for (i = 0; i < no_items; i++)
    {
        printf("Weight[%d]:\t", i);
        scanf("%d", &weight[i]);
        printf("Value[%d]:\t", i);
        scanf("%d", &value[i]);
    }

    for (i = 0; i < no_items; ++i)
        used[i] = 0;

    cur_weight = capacity;
    while (cur_weight > 0)
    {
        item = -1;
        for (i = 0; i < no_items; ++i)
            if ((used[i] == 0) &&
                ((item == -1) || ((float) value[i] / weight[i] > (float) value[item] / weight[item])))
                item = i;

        used[item] = 1;
        cur_weight -= weight[item];
        total_profit += value[item];
        if (cur_weight >= 0)
            printf("Added object %d (%d Rs., %dKg) completely in the bag. Space left: %d.\n", item + 1,
value[item], weight[item], cur_weight);
        else
        {
            int item_percent = (int) ((1 + (float) cur_weight / weight[item]) * 100);
            printf("Added %d%% (%d Rs., %dKg) of object %d in the bag.\n", item_percent, value[item],
weight[item], item + 1);
            total_profit -= value[item];
            total_profit += (1 + (float)cur_weight / weight[item]) * value[item];
        }
    }

    printf("Filled the bag with objects worth %.2f Rs.\n", total_profit);
}
```

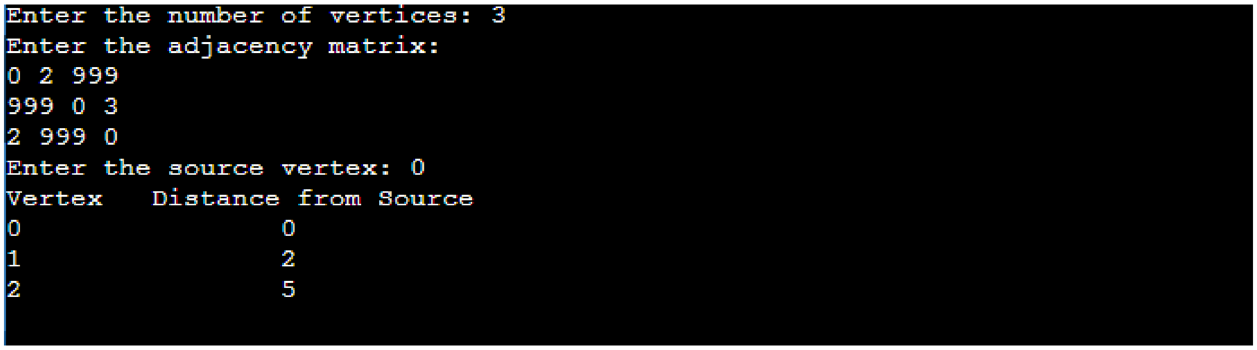


```
Enter the capacity of knapsack:
20
Enter the number of items:
3
Enter the weight and value of 3 item:
Weight[0]:      15
Value[0]:       24
Weight[1]:      10
Value[1]:       15
Weight[2]:      18
Value[2]:       25
Added object 1 (24 Rs., 15Kg) completely in the bag. Space left: 5.
Added 50% (15 Rs., 10Kg) of object 2 in the bag.
Filled the bag with objects worth 31.50 Rs.
```

Dijkstra's algorithm

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

// Function to find the vertex with the minimum distance value
int minDistance(int dist[], bool visited[], int V) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++) {
        if (!visited[v] && dist[v] <= min) {
            min = dist[v];
            min_index = v;
        }
    }

    return min_index;
}

// Function to print the constructed distance array
void printSolution(int dist[], int V) {
    printf("Vertex \t Distance from Source\n");
    for (int i = 0; i < V; i++) {
        printf("%d \t\t %d\n", i, dist[i]);
    }
}

// Function to implement Dijkstra's algorithm
void dijkstra(int graph[][100], int source, int V) {
    int dist[V];     // Array to store the shortest distance from source to vertex i
    bool visited[V]; // Array to keep track of visited vertices

    // Initialize all distances as infinite and visited array as false
    for (int i = 0; i < V; i++) {
        dist[i] = INT_MAX;
        visited[i] = false;
    }

    // Distance of source vertex from itself is always 0
    dist[source] = 0;

    // Find the shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, visited, V);
        visited[u] = true;

        // Update dist[v] only if it's not visited, there's an edge from u to v,
        // and the total weight of the path from source to v through u is smaller
        for (int v = 0; v < V; v++) {
```

```c
            if (!visited[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    // Print the constructed distance array
    printSolution(dist, V);
}

int main() {
    int V; // Number of vertices in the graph
    printf("Enter the number of vertices: ");
    scanf("%d", &V);

    int graph[100][100]; // Adjacency matrix
    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    int source; // Starting vertex
    printf("Enter the source vertex: ");
    scanf("%d", &source);

    dijkstra(graph, source, V);

    return 0;
}
```



```
Enter the number of vertices: 3
Enter the adjacency matrix:
0 2 999
999 0 3
2 999 0
Enter the source vertex: 0
Vertex   Distance from Source
0                0
1                2
2                5
```