UNIVERSITY OF DELHI

HINDU COLLEGE

BSC (HONS) COMPUTER SCIENCE

SEMESTER - 4

DESIGN AND ANALYSIS OF ALGORITHMS

RISHAV RAJ

21667

Note : For the algorithms at S.No 1 to 3 test run the algorithm on 100 different inputs of sizes varying from 30 to 1000. Count the number of comparisons and draw the graph. Compare it with a graph of nlogn.

Question 1 :

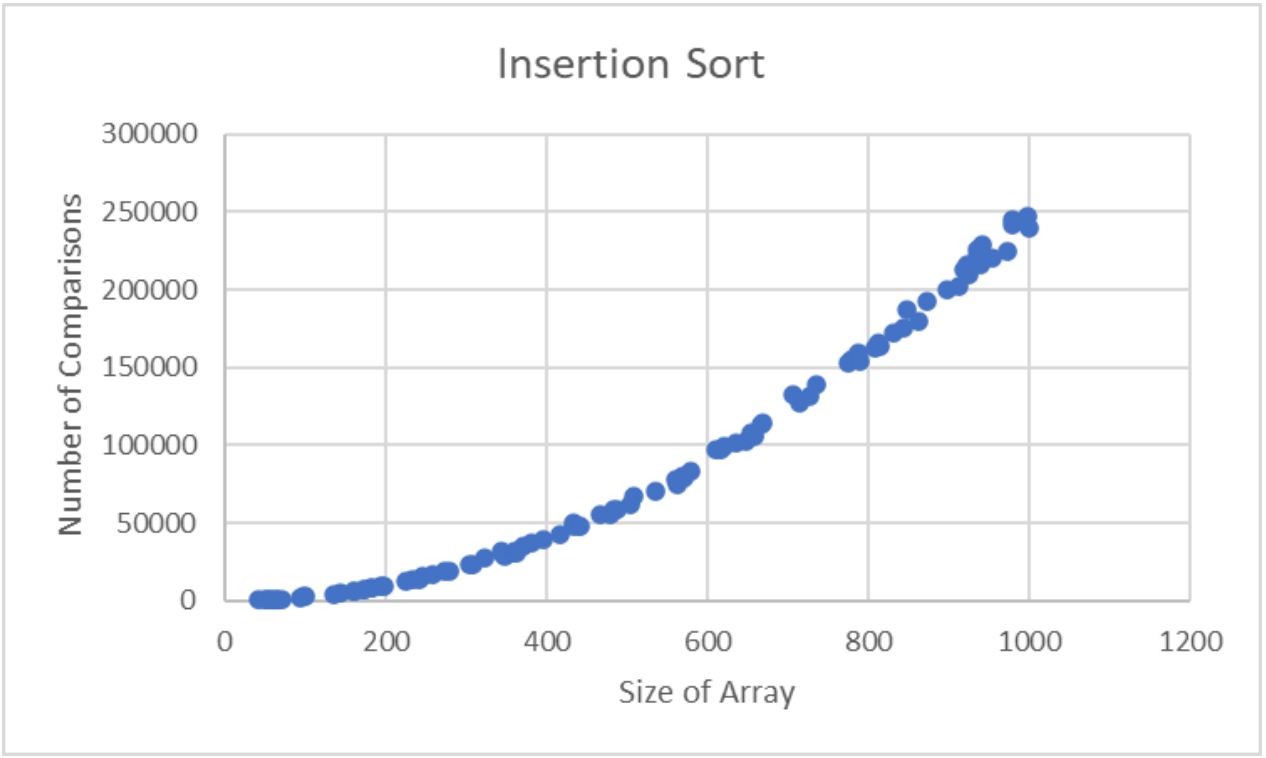
A. Implement Insertion Sort (The program should report the number of comparisons)

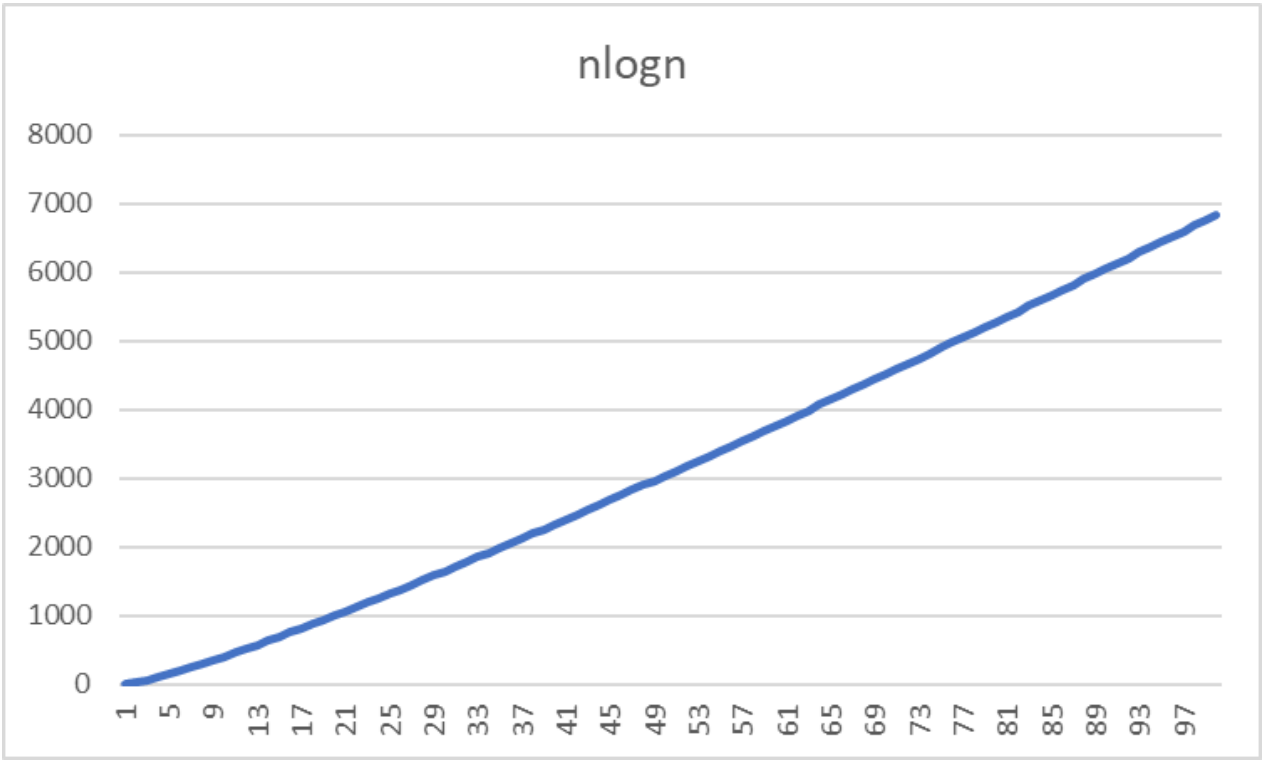
Solution 1.A :

```
# Created By : RISHAV RAJ
#include <iostream>
#include <fstream>
using namespace std;

// Insertion Sort Function - return type is set to integer as number of
comparisons needs to be returned
template <class T>
int insertionSort(T arr[], int size)
{
    T key;
    int i, j, comp = -1;
    for (i = 0; i < size; i++)
    {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
            comp++;
            arr[j + 1] = arr[j];
            j--;
        }
        comp++;
        arr[j + 1] = key;
    }
    return comp; // Returns no. of comparisons
}

int main()
{

    int n = 100;
```

```
    // Opening and initializing file
    ofstream outputFile("data.csv");
    for (int i = 0; i < n; i++)
    {
        int size;
        // Taking a random size between 30 and 1000
        do
        {
            size = rand() % 1000;
        } while (size < 30 || size > 1000);

        int *arr = new int[size];

        // Initializing array with random values till 1000
        for (int j = 0; j < size; j++)
        {
            arr[j] = rand() % 1000;
        }

        // Applying insertion sort
        int comp = insertionSort<int>(arr, size);

        // Saving data to file
        outputFile << "\n"<< size << "," << comp;
    }
    outputFile.close();
    cout << endl;
    return 0;
}
```

Output 1.A :


Insertion Sort
Number of Comparisons
300000
250000
200000
150000
100000
50000
0
0
200
400
600
800
1000
1200
Size of Array


nlogn
8000
7000
6000
5000
4000
3000
2000
1000
0
1
5
9
13
17
21
25
29
33
37
41
45
49
53
57
61
65
69
73
77
81
85
89
93
97

B. Implement Merge Sort (The program should report the number of comparisons)

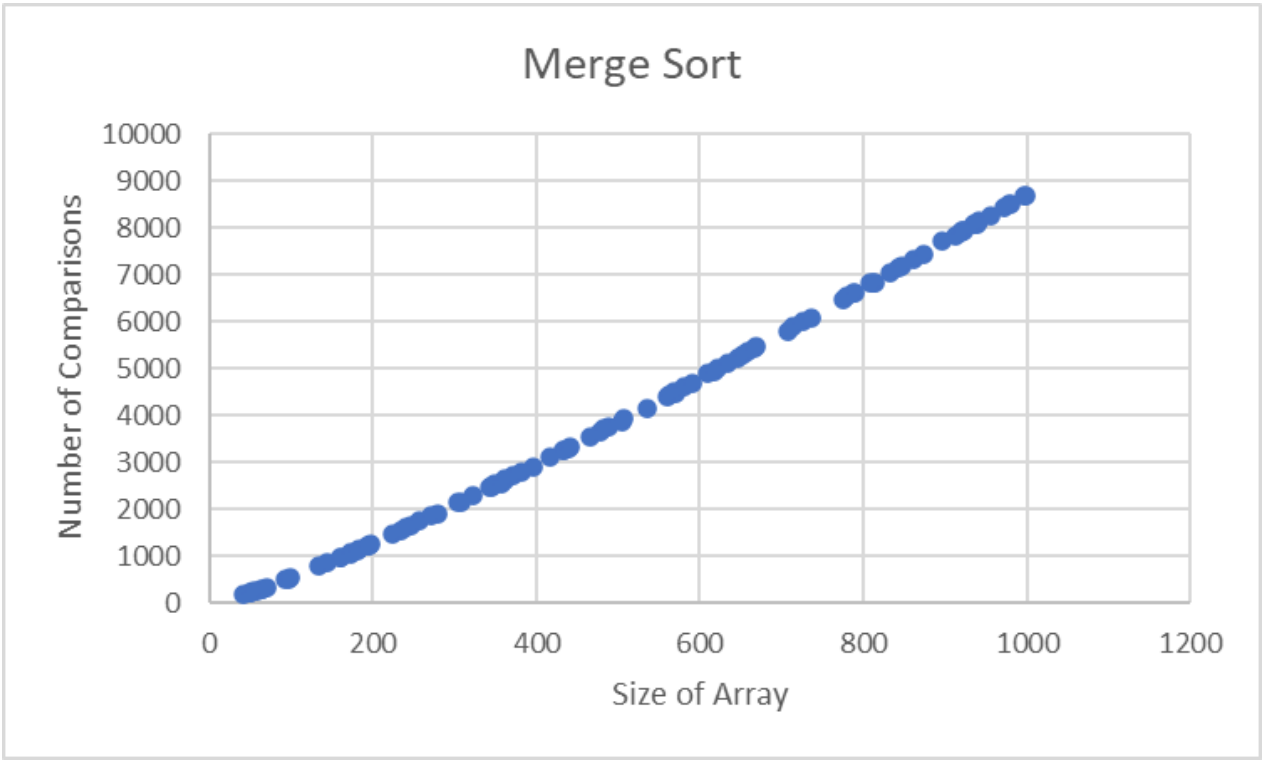
Solution 1.B :

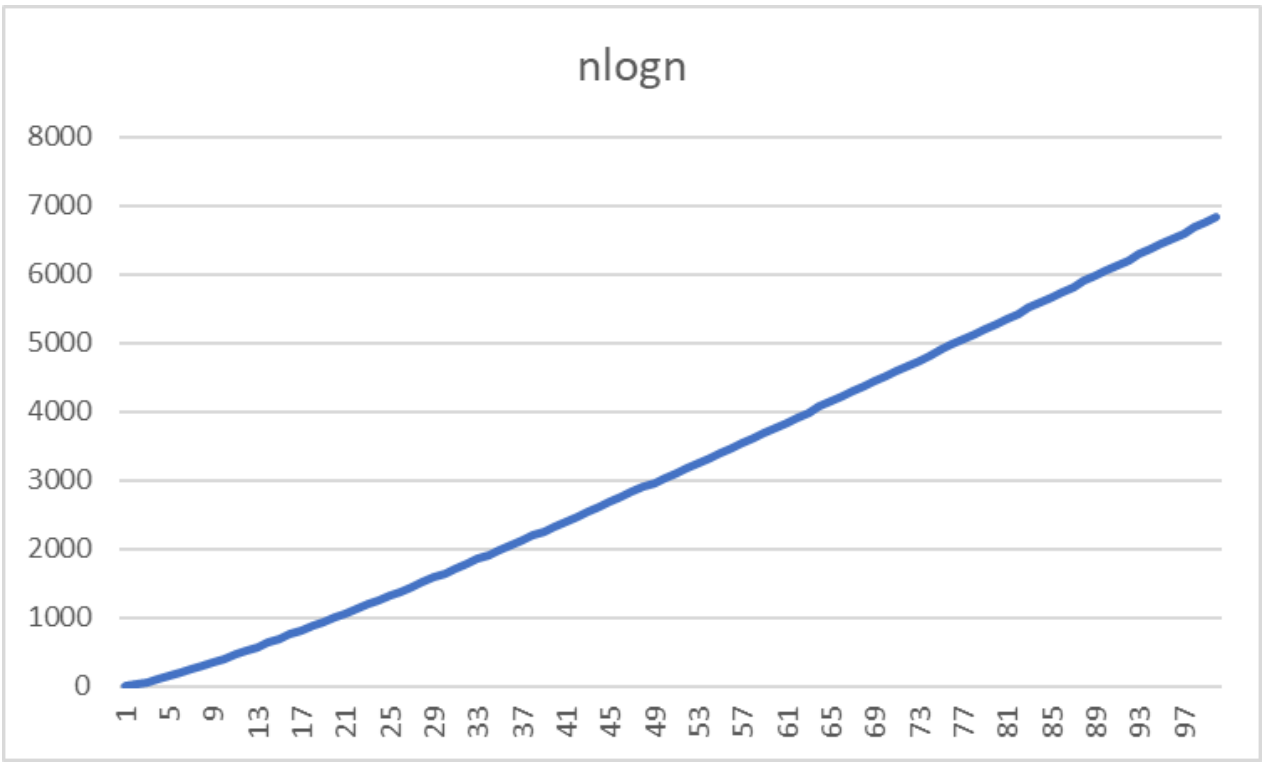
```
# Created By : RISHAV RAJ
#include <iostream>
#include <fstream>
using namespace std;

// Comparison counter
int comps;

// Merge function to merge two subarrays into arr
void merge(int arr[], int left, int mid, int right)
{

    // Create L <- arr[left:mid] and M <- arr[mid+1:right]
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], M[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        M[j] = arr[mid + 1 + j];

    // Current indices of subarrays and arr
    int i = 0, j = 0, k = left;

    /**
      Iterate over elements of L and M and pick larger element from L
and M
      until we reach end of either array, add the elements to correct
position in arr
    **/
    while (i < n1 && j < n2)
    {
        if (L[i] <= M[j])
        {
            arr[k] = L[i];
```

```
            i++;
        }
        else
        {
            arr[k] = M[j];
            j++;
        }
        comps++; // Increment comparison counter
        k++;
    }

    // Put remaining elements in arr
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2)
    {
        arr[k] = M[j];
        j++;
        k++;
    }
}

// Merge Sort function
void mergeSort(int arr[], int left, int right)
{
    if (left < right)
    {
        // Finding mid of array
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

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

```cpp
int main()
{
    // Number of iterations
    int n = 100;

    // Opening and initializing file
    ofstream outputFile("data.csv");
    for (int i = 0; i <= n; i++)
    {
        int size;
        // Taking a random size between 30 and 1000
        do
        {
            size = rand() % 1000;
        } while (size < 30 || size > 1000);
        int *arr = new int[size];

        // Initializing array with random values
        for (int j = 0; j < size; j++)
        {
            arr[j] = rand() % 1000;
        }

        // Initializing comps and applying merge sort
        comps = 0;
        mergeSort(arr, 0, size - 1);

        // Saving data to file
        outputFile << "\n" << size << "," << comps;
    }

    outputFile.close(); // Close the file
    cout << endl;
    return 0;
}
```

Output 1.B :


Merge Sort
Number of Comparisons
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
200
400
600
800
1000
1200
Size of Array


nlogn
8000
7000
6000
5000
4000
3000
2000
1000
0
1
5
9
13
17
21
25
29
33
37
41
45
49
53
57
61
65
69
73
77
81
85
89
93
97

Question 2 :

Implement Heap Sort(The program should report the number of comparisons)

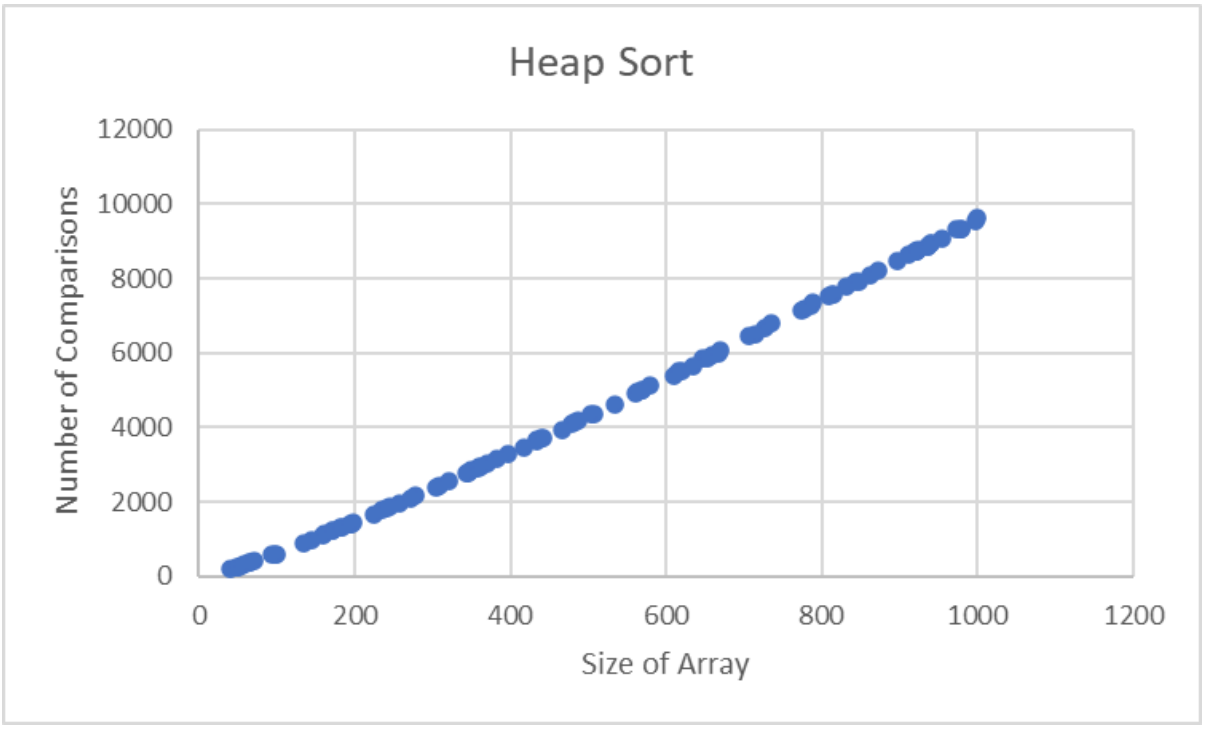
Solution 2 :

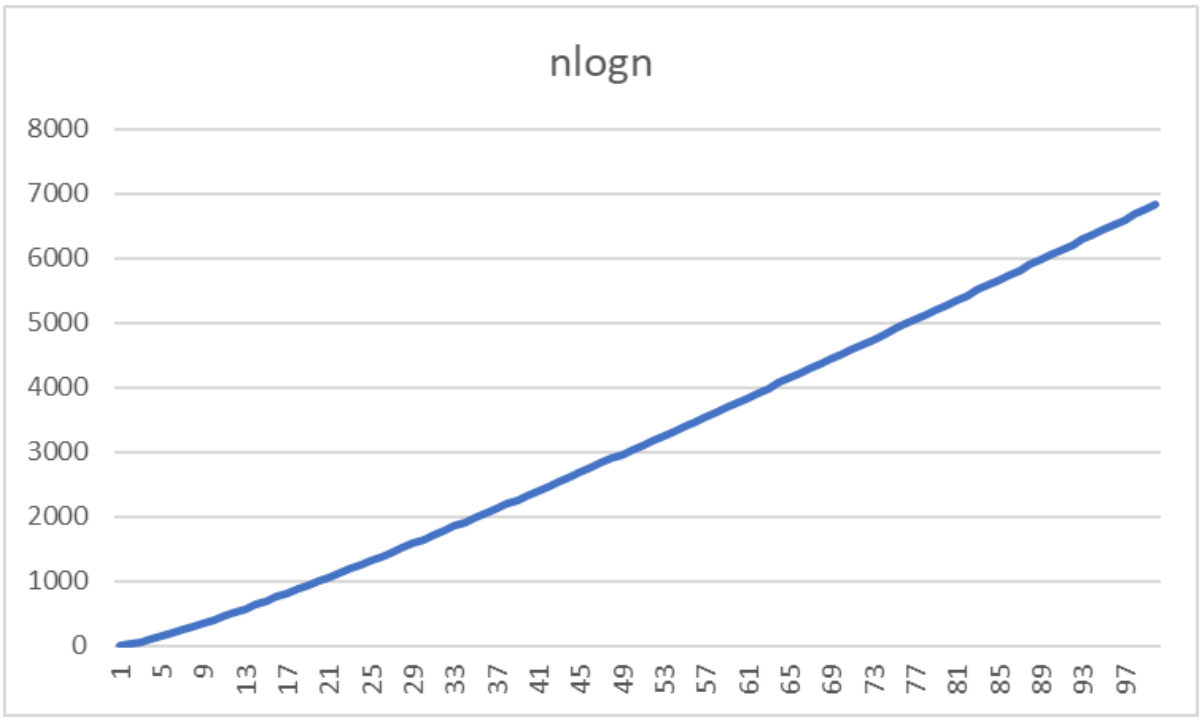
```
# Created By : RISHAV RAJ
#include <iostream>
#include <fstream>
using namespace std;

// Max Heapify function
int max_heapify(int arr[], int i, int size)
{
    int count = 0; // comparison counter
    int maxIndex = i;
    int l = 2 * i + 1; // left child
    int r = 2 * i + 2; // right child

    // Comparisons with left child
    if (l < size && arr[l] > arr[maxIndex])
        maxIndex = l;
    // Comparisons with left child
    if (r < size && arr[r] > arr[maxIndex])
        maxIndex = r;

    // Main logic
    if (i != maxIndex)
    {
        count++;
        swap(arr[i], arr[maxIndex]);
        count += max_heapify(arr, maxIndex, size);
    }
    return count;
}
/*
    Build Heap function - iterating over all the
    non leaf nodes and applying max_heapify
*/
int buildMaxHeap(int arr[], int size)
{
    int count = 0;
    for (int i = size / 2 - 1; i >= 0; --i)
        count += max_heapify(arr, i, size) + 1;
```

```
    return count;
}

/*
    Heap Sort function - iterates (size-1) times,
    swap max element (root) and last element and then
    applying max_heapify on root element (and decrease size by 1)
*/
int heapSort(int arr[], int size)
{
    int count = buildMaxHeap(arr, size);
    for (int i = size - 1; i > 0; i--)
    {
        swap(arr[0], arr[i]);
        count += max_heapify(arr, 0, i) + 1;
    }
    return count;
}
int main()
{
    // Number of iterations
    int n = 100;

    // Opening and initializing file
    ofstream outputFile("data.csv");

    for (int i = 0; i < n; i++)
    {
        int size;
        // Taking a random size between 30 and 1000
        do
        {
            size = rand() % 1000;
        } while (size < 30 || size > 1000);
        int *arr = new int[size];

        // Initializing array with random values
        for (int j = 0; j < size; j++)
        {
            arr[j] = rand() % 1000;
        }

        // Initializing comps and applying heap sort
        int comps = heapSort(arr, size);
```

```
        // Saving data to file
        outputFile << "\n"
                   << size << "," << comps;
    }

    outputFile.close(); // Close the file
    cout << endl;
    return 0;
}
```

Output 2 :





Question 3 :

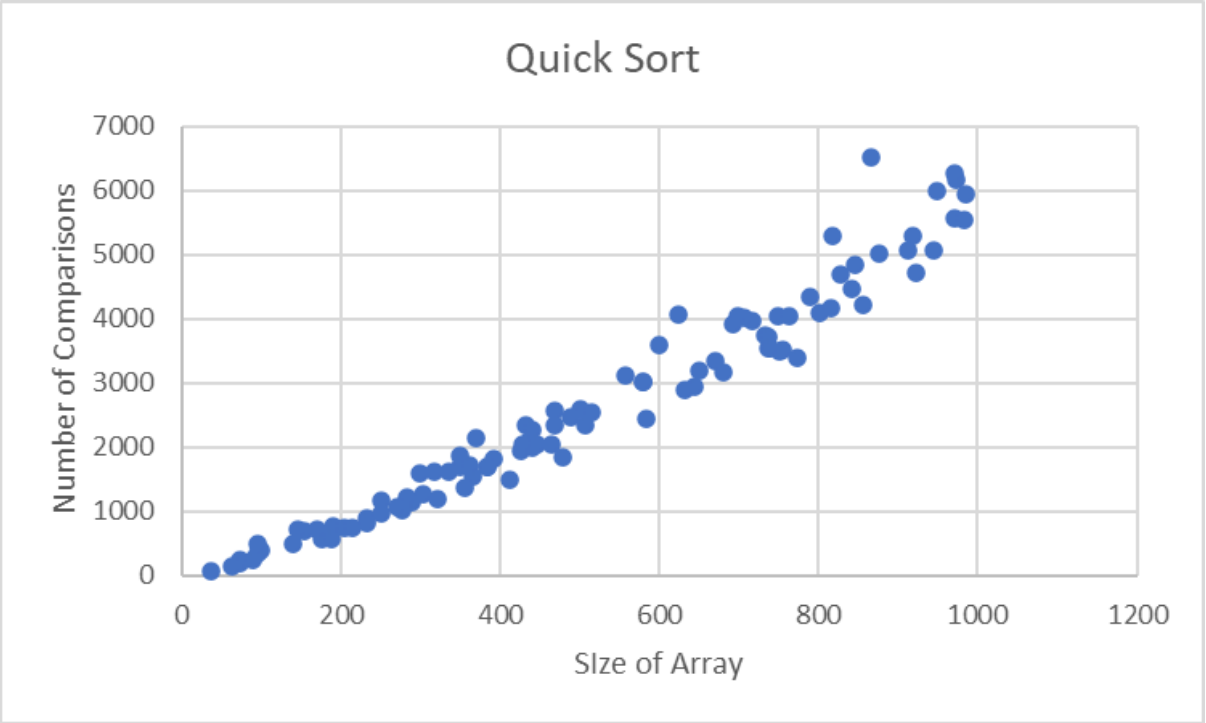
Implement Randomized Quick sort (The program should report the number of comparisons)

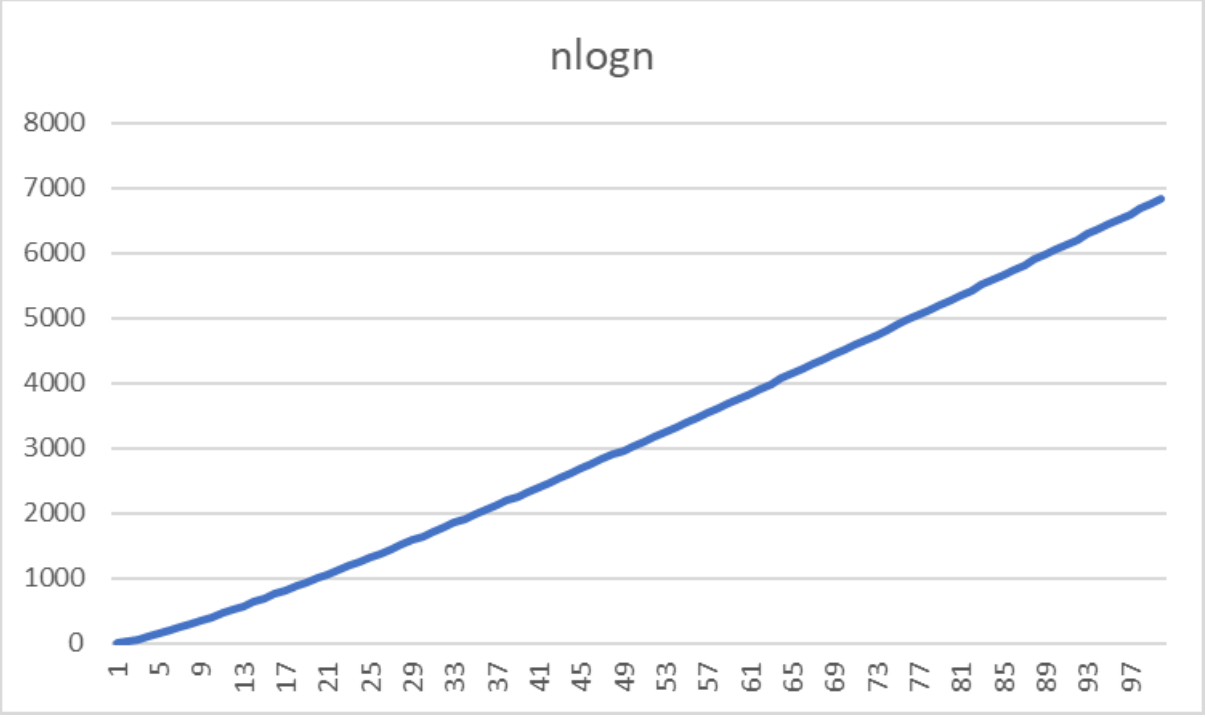
Solution 3 :

```
# Created By : RISHAV RAJ
#include <iostream>
#include <fstream>
using namespace std;
// Comparison Counter
int comps;
// Function to make partitions of array according to a pivot
int partition(int array[], int low, int high)
{
    // Select the pivot element
    int pivot = array[high];
    int i = (low - 1);

    // Put the elements smaller than pivot on the left
    // and greater than pivot on the right of pivot
    for (int j = low; j < high; j++)
    {
        if (array[j] <= pivot)
        {
            i++;
            swap(array[i], array[j]);
            comps++; // Comparison + 1
        }
    }
    swap(array[i + 1], array[high]);
    return (i + 1);
}
// Function to generate random pivot, and then
// call partition on that pivot
int randomPartition(int arr[], int low, int high)
{
    int random = low + rand() % (high - low);
    swap(arr[random], arr[high]);

    return partition(arr, low, high);
}
// Quick Sort function
void quickSort(int array[], int low, int high)
```

```
{
    if (low < high)
    {
        // Make partition based on random pivot
        int pi = randomPartition(array, low, high);

        // Sort the elements on the left of pivot
        quickSort(array, low, pi - 1);

        // Sort the elements on the right of pivot
        quickSort(array, pi + 1, high);
    }
}
// Main function
int main()
{
    // Number of iterations
    int n = 100;
    // Opening and initializing file
    ofstream outputFile("data.csv");
    for (int i = 0; i < n; i++)
    {
        int size;
        // Taking a random size between 30 and 1000
        do{
            size = rand() % 1000;
        } while (size < 30 || size > 1000);
        int *arr = new int[size];
        // Initializing array with random values
        for (int j = 0; j < size; j++)
        {
            arr[j] = rand() % 1000;
        }
        // Initializing comps and applying Quick sort algorithm
        comps = 0;
        quickSort(arr, 0, size);
        // Saving data to file
        outputFile << "\n" << size << "," << comps;
    }
    outputFile.close(); // Close the file
    cout << endl;
    return 0;
}
```

Output 3 :


Quick Sort
Number of Comparisons
7000
6000
5000
4000
3000
2000
1000
0
0
200
400
600
800
1000
1200
SIze of Array


nlogn
8000
7000
6000
5000
4000
3000
2000
1000
0
1
5
9
13
17
21
25
29
33
37
41
45
49
53
57
61
65
69
73
77
81
85
89
93
97

Question 4 :

Implement Radix Sort

Solution 4 :

```
# Created By : RISHAV RAJ
#include <iostream>
using namespace std;
// Get maximum value from array.
int getMax(int arr[], int n)
{
    int max = arr[0];
    for (int i = 1; i < n; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}
// Count sort of arr[].
void countSort(int arr[], int n, int exp)
{
    // Count[i] array will be counting the number of array values having
that 'i' digit at their (exp)th place.
    int output[n], i, count[10] = {0};

    // Count the number of times each digit occurred at (exp)th place in
every input.
    for (i = 0; i < n; i++)
        count[(arr[i] / exp) % 10]++;
    // Calculating their cumulative count.
    for (i = 1; i < 10; i++)
        count[i] += count[i - 1];

    // Inserting values according to the digit '(arr[i] / exp) % 10'
fetched into count[(arr[i] / exp) % 10].
    for (i = n - 1; i >= 0; i--)
    {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }
    // Assigning the result to the arr pointer of main().
    for (i = 0; i < n; i++)
```

```
        arr[i] = output[i];
}
// Sort arr[] of size n using Radix Sort.
void radixsort(int arr[], int n)
{
    int exp, m;
    m = getMax(arr, n);

    // Calling countSort() for digit at (exp)th place in every input.
    for (exp = 1; m / exp > 0; exp *= 10)
        countSort(arr, n, exp);
}
int main()
{
    cout << endl;
    int n = 8;
    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
    cout << "Original Data : " << endl;
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
    radixsort(arr, n);

    // Printing the sorted data.
    cout << "Sorted Data : " << endl;
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
    return 0;
}
```

Output 4 :

```
Original Data :
170 45 75 90 802 24 2 66
Sorted Data :
2 24 45 66 75 90 170 802
```

Question 5 :

Implement Bucket Sort

Solution 5 :

```
# Created By : RISHAV RAJ
#include <iostream>
#include <string>
using namespace std;
struct bucket
{
    int ptr;
    float *value;
};
template <typename T>
void insertionSort(T arr[], int size)
{
    T key;
    int i, j;
    for (i = 0; i < size; ++i)
    {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key){
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
void print(float ar[], int size, string str)
{
    cout << str << "[";
    for (int i = 0; i < size; ++i)
    {
        cout << (i == 0 ? "" : ", ") << ar[i];
    }
    cout << "]" << endl;
}
// Bucket Sort Function
void bucketSort(float ar[], int n)
{
    struct bucket B[n];
```

```
    for (int i = 0; i < n; i++)
    {
        B[i].ptr = -1;
        B[i].value = new float[n];
    }
    for (int i = 0; i < n; i++)
    {
        int idx = n * ar[i];
        B[idx].value[++B[idx].ptr] = ar[i];
    }
    for (int i = 0; i < n; i++)
    {
        insertionSort<float>(B[i].value, B[i].ptr + 1);
        // cout << "  Bucket " << i;
        // print(B[i].value, B[i].ptr+1, " = ");
    }
    int idx = 0;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < B[i].ptr + 1; j++)
            ar[idx++] = B[i].value[j];
        delete[] B[i].value; // and free the memory
    }
}
int main()
{
    // Taking a random size ranging from 5 to 20
    int size = rand() % 15 + 5;
    float *arr = new float[size];

    // Initializing array with random values between 0 and 1 with 2
digits after decimal
    for (int j = 0; j < size; j++)
    {
        arr[j] = (float)(rand() % 100) / 100;
    }
    print(arr, size, "\nUnsorted Array = ");
    bucketSort(arr, size);
    print(arr, size, "\nSorted Array = ");
    cout << endl;
    return 0;
}
```

Output 5 :

```
Unsorted Array = [0.67, 0.34, 0, 0.69, 0.24, 0.78, 0.58, 0.62, 0.64, 0.05, 0.45, 0.81, 0.27, 0.61, 0.91, 0.95]
Sorted Array = [0, 0.05, 0.24, 0.27, 0.34, 0.45, 0.58, 0.61, 0.62, 0.64, 0.67, 0.69, 0.78, 0.81, 0.91, 0.95]
```

Question 6 :

Implement Randomized Select

Solution 6 :

```
# Created By : RISHAV RAJ
#include <iostream>
using namespace std;

// Partition function to partition the array around a pivot element
int Partition(int A[], int p, int r) {
    int i = p - 1;
    for (int j = p; j < r; j++) {
        if (A[j] <= A[r]) {
            i++;
            swap(A[i], A[j]);
        }
    }
    swap(A[i + 1], A[r]);
    return i + 1;
}

// Randomized partition function to select a random pivot element
int RandomizedPartition(int A[], int p, int r) {
    int i = p + rand() % (r - p + 1);
    swap(A[r], A[i]);
    return Partition(A, p, r);
}

// Randomized Select function to select ith order statistic from an
unsorted array
int RandomizedSelect(int A[], int p, int r, int i) {
    if (p == r) {
        return A[p];
    }
    int q = RandomizedPartition(A, p, r);
    int k = q - p + 1;
```

```
    if (i == k) {
        return A[q];
    } else if (i < k) {
        return RandomizedSelect(A, p, q - 1, i);
    } else {
        return RandomizedSelect(A, q + 1, r, i - k);
    }
}

int main() {
    int A[] = { 3, 5, 2, 7, 6, 1, 4 };
    int n = sizeof(A) / sizeof(A[0]);
    int i = 3;
    int order_statistic = RandomizedSelect(A, 0, n - 1, i);
    cout << "The " << i << "th order statistic is: " << order_statistic
<< endl;
    return 0;
}
```

Output 6 :

```
The 3th order statistic is: 3
```

Question 7 :

Implement Breadth-First Search in a graph

Solution 7 :

```
# Created By : RISHAV RAJ
#include <bits/stdc++.h>
using namespace std;
class Graph {
    // No. of vertices
    int V;
    // Pointer to an array containing adjacency lists
    vector<list<int> > adj;

public:
    Graph(int V);
    // Function to add an edge to graph
    void addEdge(int v, int w);
    // Prints BFS traversal from a given source s
    void BFS(int s);
};
Graph::Graph(int V)
{
    this->V = V;
    adj.resize(V);
}
void Graph::addEdge(int v, int w)
{
    // Add w to v's list.
    adj[v].push_back(w);
}

void Graph::BFS(int s)
{
    // Mark all the vertices as not visited
    vector<bool> visited;
    visited.resize(V, false);
    // Create a queue for BFS
    list<int> queue;
    // Mark the current node as visited and enqueue it
    visited[s] = true;
    queue.push_back(s);
```

```
    while (!queue.empty()) {
        // Dequeue a vertex from queue and print it
        s = queue.front();
        cout << s << " ";
        queue.pop_front();

        // Get all adjacent vertices of the dequeued
        // vertex s. If a adjacent has not been visited,
        // then mark it visited and enqueue it
        for (auto adjacent : adj[s]) {
            if (!visited[adjacent]) {
                visited[adjacent] = true;
                queue.push_back(adjacent);
            }
        }
    }
}

int main()
{
    cout<<endl;cout<<endl;cout<<endl;
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal "
        << "(starting from vertex 2) \n";
    g.BFS(2);
    cout<<endl;cout<<endl;cout<<endl;
    return 0;
}
```

Output 7 :

```
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
```

Question 8 :

Implement Depth-First Search in a graph

Solution 8 :

```
# Created By : RISHAV RAJ
#include <bits/stdc++.h>
using namespace std;
class Graph {
public:
    map<int, bool> visited;
    map<int, list<int> > adj;

    // function to add an edge to graph
    void addEdge(int v, int w);

    // DFS traversal of the vertices
    // reachable from v
    void DFS(int v);
};

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}

void Graph::DFS(int v)
{
    // Mark the current node as visited and
    // print it
    visited[v] = true;
    cout << v << " ";

    // Recur for all the vertices adjacent
    // to this vertex
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFS(*i);
}
int main()
{
```

```
    Graph g;
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);
    cout << "Following is Depth First Traversal"
            " (starting from vertex 2) \n";
    g.DFS(2);
    return 0;
}
```

Output 8 :

```
Following is Depth First Traversal (starting from vertex 2)
2 0 1 3
```

Question 9 :

Write a program to determine the minimum spanning tree of a graph using both Prims and Kruskal's algorithm

Solution 9.A : Kruskal Algorithm

```
# Created By : RISHAV RAJ
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

#define edge pair<int, int>

class Graph {
   private:
  vector<pair<int, edge> > G;  // graph
  vector<pair<int, edge> > T;  // mst
  int *parent;
  int V;  // number of vertices/nodes in graph
   public:
```

```
  Graph(int V);
  void AddWeightedEdge(int u, int v, int w);
  int find_set(int i);
  void union_set(int u, int v);
  void kruskal();
  void print();
};
Graph::Graph(int V) {
  parent = new int[V];

  //i 0 1 2 3 4 5
  //parent[i] 0 1 2 3 4 5
  for (int i = 0; i < V; i++)
    parent[i] = i;

  G.clear();
  T.clear();
}
void Graph::AddWeightedEdge(int u, int v, int w) {
  G.push_back(make_pair(w, edge(u, v)));
}
int Graph::find_set(int i) {
  // If i is the parent of itself
  if (i == parent[i])
    return i;
  else
    // Else if i is not the parent of itself
    // Then i is not the representative of his set,
    // so we recursively call Find on its parent
    return find_set(parent[i]);
}

void Graph::union_set(int u, int v) {
  parent[u] = parent[v];
}
void Graph::kruskal() {
  int i, uRep, vRep;
  sort(G.begin(), G.end());  // increasing weight
  for (i = 0; i < G.size(); i++) {
    uRep = find_set(G[i].second.first);
    vRep = find_set(G[i].second.second);
    if (uRep != vRep) {
      T.push_back(G[i]);  // add to tree
```

```
      union_set(uRep, vRep);
    }
  }
}
void Graph::print() {
  cout << "Edge\t:"
     << " Weight" << endl;
  for (int i = 0; i < T.size(); i++) {
    cout << T[i].second.first << " - " << T[i].second.second << "\t: "
       << T[i].first;
    cout << endl;
  }
}
int main() {
  Graph g(6);
  g.AddWeightedEdge(0, 1, 4);
  g.AddWeightedEdge(0, 2, 4);
  g.AddWeightedEdge(1, 2, 2);
  g.AddWeightedEdge(1, 0, 4);
  g.AddWeightedEdge(2, 0, 4);
  g.AddWeightedEdge(2, 1, 2);
  g.AddWeightedEdge(2, 3, 3);
  g.AddWeightedEdge(2, 5, 2);
  g.AddWeightedEdge(2, 4, 4);
  g.AddWeightedEdge(3, 2, 3);
  g.AddWeightedEdge(3, 4, 3);
  g.AddWeightedEdge(4, 2, 4);
  g.AddWeightedEdge(4, 3, 3);
  g.AddWeightedEdge(5, 2, 2);
  g.AddWeightedEdge(5, 4, 3);
  g.kruskal();
  g.print();
  return 0;
}
```

Output 9.A :

```
Edge    : Weight
1 - 2   : 2
2 - 5   : 2
2 - 3   : 3
3 - 4   : 3
0 - 1   : 4
```

Solution 9.B : Prims Algorithm

```
# Created By : RISHAV RAJ
#include <bits/stdc++.h>
using namespace std;
// Number of vertices in the graph
#define V 5
int minKey(int key[], bool mstSet[])
{
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;
    return min_index;
}
void printMST(int parent[], int graph[V][V])
{
    cout << "Edge \tWeight\n";
    for (int i = 1; i < V; i++)
        cout << parent[i] << " - " << i << " \t" << graph[i][parent[i]]
<< " \n";
}
void primMST(int graph[V][V])
{
    int parent[V];
    int key[V];
    bool mstSet[V];
    // Initialize all keys as INFINITE
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < V - 1; count++)
    {
        int u = minKey(key, mstSet);
        mstSet[u] = true;
        for (int v = 0; v < V; v++)
            if (graph[u][v] && mstSet[v] == false && graph[u][v] <
key[v])
                parent[v] = u, key[v] = graph[u][v];
```

```
    }
    printMST(parent, graph);
}
int main()
{
    int graph[V][V] = {{0, 9, 75, 0, 0},
                       {9, 0, 95, 19, 42},
                       {75, 95, 0, 51, 66},
                       {0, 19, 51, 0, 31},
                       {0, 42, 66, 31, 0}};
    primMST(graph);
    return 0;
}
```

Output 9.B :

```
Edge    Weight
0 - 1   9
3 - 2   51
1 - 3   19
3 - 4   31
```

Question 10 :

Write a program to solve the weighted interval scheduling problem

Solution 10 :

```
# Created By : RISHAV RAJ
#include <iostream>
#include <algorithm>
using namespace std;

// A job has start time, finish time and profit.
struct Job
{
    int start, finish, profit;
};

// A utility function that is used for sorting events
// according to finish time
bool jobComparator(Job s1, Job s2)
{
    return (s1.finish < s2.finish);
```

```
}

// Find the latest job (in sorted array) that doesn't
// conflict with the job[i]. If there is no compatible job,
// then it returns -1.
int latestNonConflict(Job arr[], int i)
{
    for (int j=i-1; j>=0; j--)
    {
        if (arr[j].finish <= arr[i-1].start)
            return j;
    }
    return -1;
}

// A recursive function that returns the maximum possible
// profit from given array of jobs. The array of jobs must
// be sorted according to finish time.
int findMaxProfitRec(Job arr[], int n)
{
    // Base case
    if (n == 1) return arr[n-1].profit;

    // Find profit when current job is included
    int inclProf = arr[n-1].profit;
    int i = latestNonConflict(arr, n);
    if (i != -1)
    inclProf += findMaxProfitRec(arr, i+1);

    // Find profit when current job is excluded
    int exclProf = findMaxProfitRec(arr, n-1);

    return max(inclProf, exclProf);
}

// The main function that returns the maximum possible
// profit from given array of jobs
int findMaxProfit(Job arr[], int n)
{
    // Sort jobs according to finish time
    sort(arr, arr+n, jobComparator);

    return findMaxProfitRec(arr, n);
```

```
}
// Driver program
int main()
{
    cout<<endl;cout<<endl;cout<<endl;
    Job arr[] = {{3, 10, 20}, {1, 2, 50}, {6, 19, 100}, {2, 100, 200}};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "The optimal profit is " << findMaxProfit(arr, n);
    cout<<endl;cout<<endl;cout<<endl;
    return 0;
}
```

Output 10 :

```
The optimal profit is 250
```

Question 11 :

Write a program to solve the 0-1 knapsack problem

Solution 11 :

```
# Created By : RISHAV RAJ
#include <bits/stdc++.h>
using namespace std;
int max(int a, int b) { return (a > b) ? a : b; }
// Returns the maximum value that
// can be put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n)
{
    // Base Case
    if (n == 0 || W == 0)
        return 0;
    // If weight of the nth item is more
    // than Knapsack capacity W, then
    // this item cannot be included
    // in the optimal solution
    if (wt[n - 1] > W)
        return knapSack(W, wt, val, n - 1);
```

```
    // Return the maximum of two cases:
    // (1) nth item included
    // (2) not included
    else
        return max(
            val[n - 1]+ knapSack(W - wt[n - 1], wt, val, n - 1),
            knapSack(W, wt, val, n - 1));
}
int main()
{
    int profit[] = { 60, 100, 120 };
    int weight[] = { 10, 20, 30 };
    int W = 50;
    int n = sizeof(profit) / sizeof(profit[0]);
    cout << knapSack(W, weight, profit, n);
    return 0;
}
```

Output 11 :

```
220
```

END OF ASSIGNMENT

RISHAV RAJ | 21667