Atma Ram Sanatan Dharma College

University of Delhi

Design and analysis of Algorithms

Submitted By

Tanu Soni

College Roll No. 20/88067

BSc (Hons) Computer Science

Submitted To

Ms Archana Gahlaut

Department of Computer Science

1. Implement Insertion Sort

Source code

```
#include
<cstdlib>
#include
<fstream>
#include
<iomanip>
#include
<iostream>

#define MIN_SIZE 30
#define MAX_SIZE

1000 using namespace

std;

int insertionSort(int *, int);


int main()
{
  try
  {
    srand(time(0));

    int
    array[MAX_SIZE];
    int size,
    comparisons;

    ofstream fout("./results.csv");

    cout << "+________________________+\n";
    cout << "| Input Size | Best Case | Avg Case | Worst Case |\n";
```

```
cout << "+___________________________

+\n"; fout << "size,best,avg,worst\n";

for (int i = 0; i < 100; i++)
{
  // rand() % (upperBound + 1 - lowerBound) +
  lowerBound size = rand() % (MAX_SIZE + 1 -
  MIN_SIZE) + MIN_SIZE;

  // Input Size
  cout << "| " << setw(10) <<
  size; fout << size << ",";

  // Best Case
  for (int i = 0; i < size;
    i++) array[i] = i +
    1;
  comparisons = insertionSort(array, size);
  cout << " | " << setw(9) << right <<
  comparisons; fout << comparisons <<
  ",";

  // Average
  Case try
  {
    ifstream
    fin("./random.txt"); for
    (int i = 0; i < size; i++)
      fin >>
    array[i];
    fin.close();
    comparisons = insertionSort(array, size);
    cout << " | " << setw(8) << right << comparisons;
```

```
            fout << comparisons << ",";
        }
        catch (exception e)
        {
            cerr <<
            e.what(); return
            -1;
        }

        // Worst Case
        for (int i = 0; i < size;
            i++) array[i] = size
            - i;
        comparisons = insertionSort(array, size);
        cout << " | " << setw(10) << right << comparisons
        << " |\n"; fout << comparisons << "\n";
    }

    cout << "+____________________________

    +\n\n"; fout.close();

    return 0;
  }
  catch (exception e)
  {
    cerr << e.what();
    return -1;
  }
}
```

```
int insertionSort(int *array, int size)
{
   int i, j, k, key, count = 0;

   for (i = 1; i < size; i++)
   {
      key = array[i];

      for (j = i - 1; j >= 0; j--)
      {
         count++;

         if (array[j] > key)
         {
            array[j + 1] = array[j];
         }
         else
         {
            break;
         }
      }

      array[j + 1] = key;
   }

   return count;
}
```

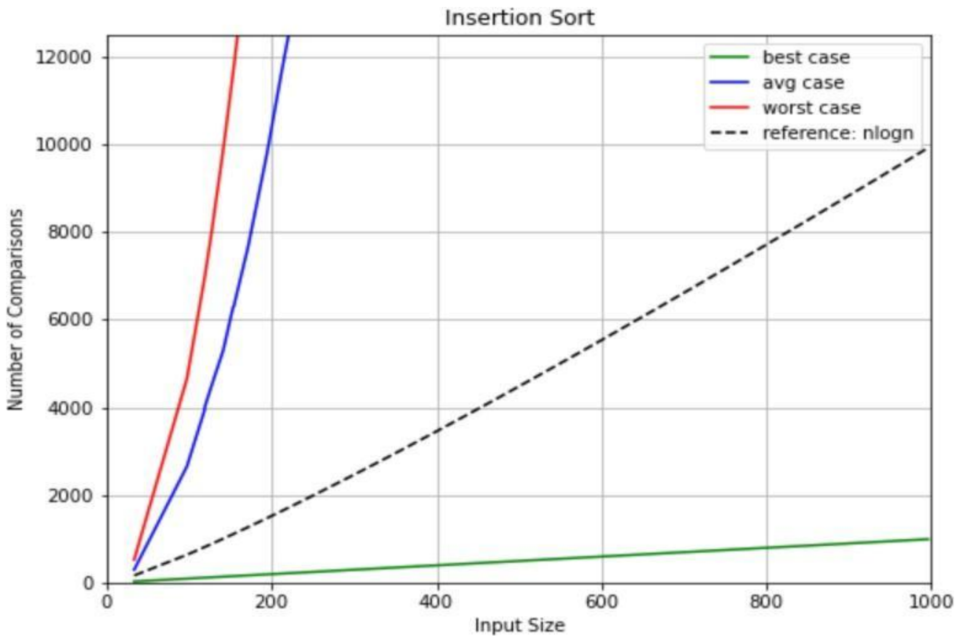
Output

Input Size	Best Case	Avg Case	Worst Case
475	474	474	112575
580	579	579	167910
883	882	882	389403
333	332	332	55278
531	530	530	140715
223	222	222	24753
153	152	152	11628
602	601	601	180901
711	710	710	252405
111	110	110	6105
272	271	271	36856
515	514	514	132355
555	554	554	153735
54	53	53	1431
260	259	259	33670
260	259	259	33670
786	785	785	308505
503	502	502	126253
230	229	229	26335
773	772	772	298378
234	233	233	27261
826	825	825	340725
918	917	917	420903
919	918	918	421821
281	280	280	39340
718	717	717	257403
74	73	73	2701
883	882	882	389403
524	523	523	137026
572	571	571	163306
348	347	347	60378

Plotting of graph

```
import math
import numpy as
np import pandas
as pd
import matplotlib.pyplot
as plt df =
pd.read_csv("results.csv"
) df =
df.sort_values("size")
plt.figure(figsize=(8, 6))
plt.plot(df['size'],
df['best'], 'g')
plt.plot(df['size'], df['avg'], 'b')
plt.plot(df['size'], df['worst'], 'r')
plt.plot(df['size'], df['size'] * np.log2(df['size']), 'k--')
```

```
plt.legend(['best case', 'avg case', 'worst case', 'reference: nlogn'])
plt.title('Insertion Sort')
plt.xlabel('Input Size')
plt.ylabel('Number of
Comparisons') plt.ylim(0, 12500)
plt.xlim(0, 1000)
plt.grid()
plt.savefig('plot.p
ng')
```



2. Implement Merge Sort

Source code

```
#include
<cstdlib>
#include
<fstream>
#include
<iomanip>
#include
<iostream>
```

```
#define MIN_SIZE 30
#define MAX_SIZE 1000

using namespace std;

int mergeSort(int *, int,
int); int merge(int *,
int, int, int);

int main()
{
  try
  {
    srand(time(0));

    int
    array[MAX_SIZE];
    int size,
    comparisons;

    ofstream fout("./results.csv");

    cout << "+____________________________+\n";
    cout << "| Input Size | Best Case | Avg Case | Worst
    Case |\n"; cout << "+_________________+\n";

    fout << "size,best,avg,worst\n";

    for (int i = 0; i < 100; i++)
    {
      // rand() % (upperBound + 1 - lowerBound) + lowerBound
```

```
size = rand() % (MAX_SIZE + 1 - MIN_SIZE) + MIN_SIZE;

// Input Size
cout << "| " << setw(10) <<
size; fout << size << ",";

// Best Case
for (int i = 0; i < size;
  i++) array[i] = i +
  1;
comparisons = mergeSort(array, 0, size -
1); cout << " | " << setw(9) << right <<
comparisons; fout << comparisons <<
",";

// Average
Case try
{
  ifstream
  fin("./random.txt"); for
  (int i = 0; i < size; i++)
    fin >>
  array[i];
  fin.close();
  comparisons = mergeSort(array, 0, size -
  1); cout << " | " << setw(8) << right <<
  comparisons; fout << comparisons <<
  ",";
}
catch (exception e)
{
  cerr <<
  e.what(); return
  -1;
```

```
        }

        // Worst Case
        for (int i = 0; i < size;
          i++) array[i] = size
          - i;
        comparisons = mergeSort(array, 0, size - 1);
        cout << " | " << setw(10) << right << comparisons
        << " |\n"; fout << comparisons << "\n";
      }

      cout << "+___________________________+\n\n";

      fout.close();

      return 0;
    }
    catch (exception e)
    {
      cerr << e.what();
      return -1;
    }
}

int mergeSort(int *array, int beg, int end)
{
    int comparisons
    = 0; if (beg <
    end)
    {
```

```
    int mid = (beg + end) / 2;
    comparisons += mergeSort(array, beg, mid);
    comparisons += mergeSort(array, mid + 1, end);
    comparisons += merge(array, beg, mid, end);
  }
  return comparisons;
}

int merge(int *array, int beg, int mid, int end)
{
  int comparisons = 0;

  int n1 = mid - beg
  + 1; int n2 = end -
  mid;
  int L[n1 + 1], R[n2 + 1];

  for (int i = 0; i < n1;
    i++) L[i] =
    array[beg + i];
  for (int j = 0; j < n2;
    j++) R[j] =
    array[mid + 1 + j];

  L[n1] = R[n2] = INT16_MAX;

  for (int i = 0, j = 0, k = beg; k <= end; k++)
  {
    if (L[i] != INT16_MAX
      && R[j] !=
      INT16_MAX)
      comparisons++;
```

```
        if (L[i] <= R[j])
          array[k] = L[i++];
        else
          array[k] = R[j++];

    }

    return comparisons;
}
```

Output

Input Size	Best Case	Avg Case	Worst Case
151	584	584	519
30	77	77	71
705	3528	3528	3203
748	3760	3760	3444
537	2500	2500	2383
688	3440	3440	3104
49	149	149	130
504	2284	2284	2244
98	347	347	309
692	3464	3464	3124
343	1544	1544	1374
195	787	787	712
994	5019	5019	4891
967	4904	4904	4709
235	961	961	898
102	365	365	323
50	153	153	133
552	2604	2604	2444
741	3726	3726	3401
867	4419	4419	4094
620	3036	3036	2760
432	1984	1984	1824
209	854	854	771
157	613	613	544
768	3840	3840	3584
175	701	701	618
362	1639	1639	1469
840	4276	4276	3940
373	1689	1689	1529
770	3857	3857	3589
658	3263	3263	2951
595	2886	2886	2635
403	1843	1843	1675
302	1319	1319	1189
501	2276	2276	2222

228	932	932	864
438	2013	2013	1855
819	4163	4163	3822
880	4480	4480	4176
291	1254	1254	1144
644	3168	3168	2892
563	2678	2678	2491
411	1884	1884	1714
551	2598	2598	2439
642	3153	3153	2885
845	4306	4306	3965
600	2916	2916	2660
167	664	664	583
798	4039	4039	3715
189	760	760	685
631	3096	3096	2821
170	679	679	595
864	4400	4400	4080
721	3623	3623	3284

Plotting of graph

```
import math
import numpy as
np import pandas
as pd
import matplotlib.pyplot
as plt df =
pd.read_csv("results.csv"
) df =
df.sort_values("size")
plt.figure(figsize=(8, 6))
plt.plot(df['size'],
df['best'], 'g')
plt.plot(df['size'], df['avg'], 'b')
plt.plot(df['size'], df['worst'], 'r')
plt.plot(df['size'], df['size'] * np.log2(df['size']), 'k--')
plt.legend(['best case', 'avg case', 'worst case',
'reference: nlogn']) plt.title('Merge Sort')
```

plt.xlabel('Input Size')

plt.ylabel('Number of

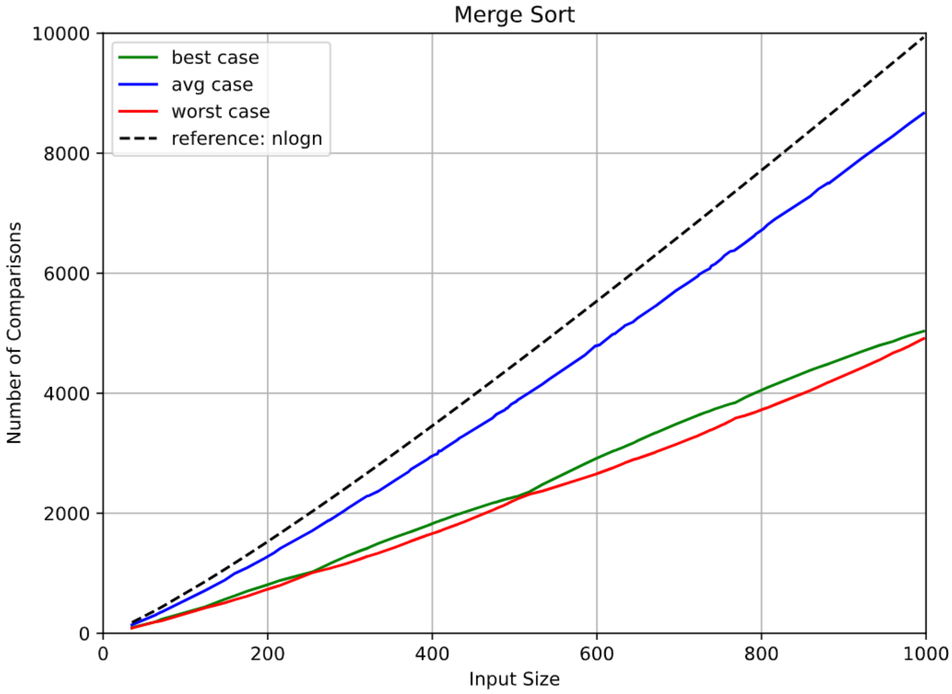
Comparisons') plt.ylim(0, 10000)

plt.xlim(0, 1000)

plt.grid()

plt.savefig('plot.p

ng')



3. Implement Heap Sort

Source code

#include

<cstdlib>

#include

<fstream>

#include

<iomanip>

#include

<iostream>

#define MIN_SIZE 30

#define MAX_SIZE 1000

```
using namespace std;

int comparisons;
int heap[MAX_SIZE];

int parent(int i)
{
  return (i - 1) / 2;
}

int left(int i)
{
  return 2 * i + 1;
}

int right(int i)
{
  return 2 * i + 2;
}

int maxHeapify(int *&A, int n, int i)
{
  int temp;
  int
  largest;
  int comparisons = 0;

  int l = left(i);
```

```
int r = right(i);

if (l <= n && A[l] > A[i])
{
   largest = l;
}
else
{
   largest = i;
}

if (r <= n && A[r] > A[largest])
{
   largest = r;
}

if (largest != i)
{
   comparisons++;

   temp = A[i];
   A[i] =
   A[largest];
   A[largest] =
   temp;

   comparisons += maxHeapify(A, n, largest);
}

return comparisons;
```

```
}

int buildHeap(int A[], int n)
{
  int comparisons = 0;

  for (int i = n / 2; i >= 0; i--)
    comparisons += maxHeapify(A,
    n, i);

  return comparisons;
}

int heapSort(int A[], int n)
{
  int comparisons = 0;

  comparisons += buildHeap(A, n);

  for (int i = n - 1; i > 0; i--)
  {
    swap(A[0], A[i]);
    comparisons += maxHeapify(A, i, 0);
  }

  return comparisons;
}

int main()
```

```
{
  try
  {
    srand(time(0));

    int
    array[MAX_SIZE];
    int size,
    comparisons;

    ofstream fout("./results.csv");

    cout << "+___________________________+\n";
    cout << "| Input Size | Best Case | Avg Case | Worst
    Case |\n"; cout << "+_________________+\n";

    fout << "size,best,avg,worst\n";

    for (int i = 0; i < 100; i++)
    {
      // rand() % (upperBound + 1 - lowerBound) +
      lowerBound size = rand() % (MAX_SIZE + 1 -
      MIN_SIZE) + MIN_SIZE;

      // Input Size
      cout << "| " << setw(10) <<
      size; fout << size << ",";

      // Best Case
      for (int i = 0; i < size;
        i++) array[i] = i +
        1;
```

```
    comparisons = heapSort(array, size);
    cout << " | " << setw(9) << right <<
    comparisons; fout << comparisons <<
    ",";

    // Average
    Case try
    {
        ifstream
        fin("./random.txt"); for
        (int i = 0; i < size; i++)
            fin >>
        array[i];
        fin.close();
        comparisons = heapSort(array, size);
        cout << " | " << setw(8) << right <<
        comparisons; fout << comparisons <<
        ",";
    }
    catch (exception e)
    {
        cerr <<
        e.what(); return
        -1;
    }

    // Worst Case
    for (int i = 0; i < size;
        i++) array[i] = size
        - i;
    comparisons = heapSort(array, size);
    cout << " | " << setw(10) << right << comparisons
    << " |\n"; fout << comparisons << "\n";
}
```

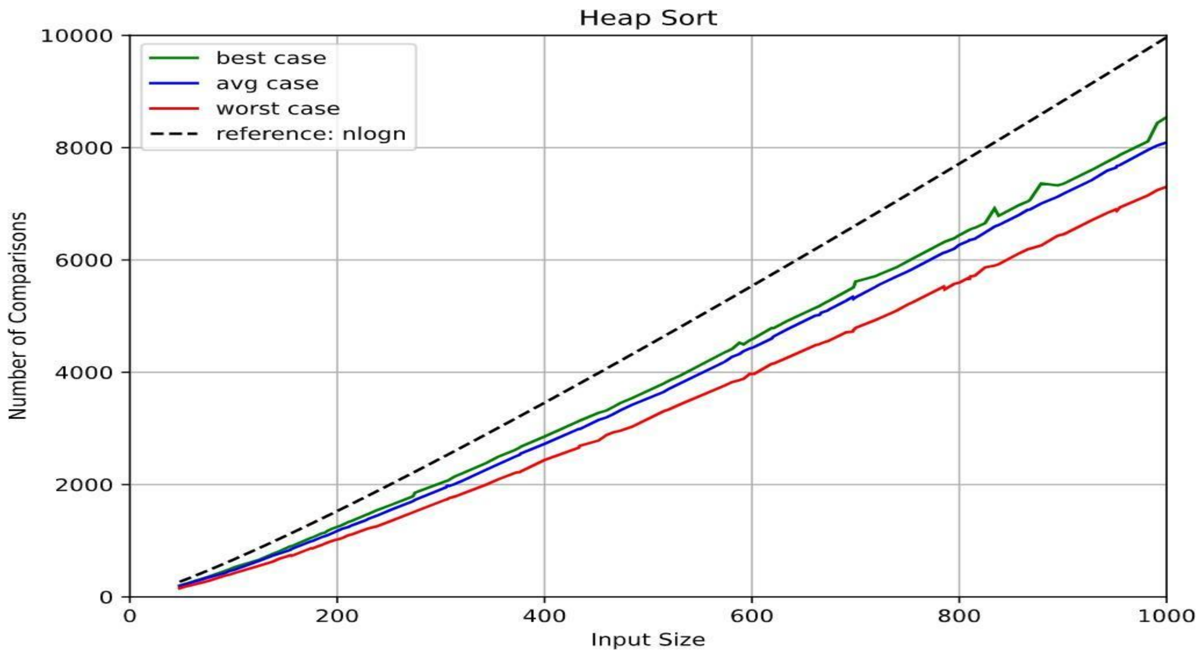
```
        cout << "+___________________________+\n\n";

        fout.close();

        return 0;
    }
    catch (exception e)
    {
        cerr << e.what();
        return -1;
    }
}
```

Output

Input Size	Best Case	Avg Case	Worst Case
571	4327	4473	3735
296	1990	2013	1677
105	549	554	442
859	7165	7042	6137
777	6239	6442	5412
609	4673	4835	4041
757	6049	6249	5259
248	1624	1591	1327
660	5175	5313	4450
103	533	537	436
92	463	469	380
682	5365	5540	4615
531	3942	4086	3430
635	4928	5087	4253
822	6625	6877	5773
259	1709	1698	1411
259	1709	1698	1411
786	6324	6523	5477
175	1054	1070	868
537	4004	4140	3457
594	4529	4691	3923
394	2795	2903	2392
338	2338	2422	1962
53	228	228	182
249	1627	1614	1343
323	2230	2283	1853
86	433	430	346
363	2551	2644	2125
755	6024	6236	5244
634	4913	5081	4235

Plotting of graph

```
import math
import numpy as
np import pandas
as pd
import matplotlib.pyplot
as plt df =
pd.read_csv("results.csv"
) df =
df.sort_values("size")
plt.figure(figsize=(8, 6))
plt.plot(df['size'],
df['best'], 'g')
plt.plot(df['size'], df['avg'], 'b')
plt.plot(df['size'], df['worst'], 'r')
plt.plot(df['size'], df['size'] * np.log2(df['size']), 'k--')
plt.legend(['best case', 'avg case', 'worst case',
'reference: nlogn']) plt.title('Heap Sort')
plt.xlabel('Input Size')
plt.ylabel('Number of
Comparisons') plt.ylim(0, 10000)
plt.xlim(0, 1000)
plt.grid()
plt.savefig('plot.p
ng')
```



4. Implement Randomized Quick Sort

Source code

```
#include
<cstdlib>
#include
<fstream>
#include
<iomanip>
#include
<iostream>
#define MIN_SIZE
30
#define MAX_SIZE 1000
```

```
using namespace std;

int partition(int *&A, int p, int r, int &ctr)
{
   int x =
   A[r]; int i
   = p - 1;

   for (int j = p; j < r; j++)
   {
      ctr++;
      if (A[j] <= x)
         swap(A[++i],
         A[j]);
   }


   swap(A[i + 1], A[r]);

   return i + 1;
}

int quickSort(int A[], int p, int r)
{
   int comparisons = 0;

   if (p < r)
   {
      int    q    =    partition(A,    p,    r,
      comparisons);  comparisons  +=
      quickSort(A,     p,     q     -     1);
      comparisons  += quickSort(A, q
      + 1, r);
```

```
  }

  return comparisons;
}

int randomPivotPartition(int A[], int p, int r, int &ctr)
{
  swap(A[r], A[p + rand() % (r - p
  + 1)]); return partition(A, p, r,
  ctr);
}

int randomizedQuickSort(int A[], int p, int r)
{
  int comparisons = 0;

  if (p < r)
  {
    int q = randomPivotPartition(A, p, r,
    comparisons); comparisons +=
    randomizedQuickSort(A, p, q - 1);
    comparisons += randomizedQuickSort(A,
    q + 1, r);
  }

  return comparisons;
}

int main()
{
```

```
try
{
  srand(time(0));

  int
  array[MAX_SIZE];
  int size,
  comparisons;

  ofstream fout("./results.csv");

  cout << "+____________________________________+\n";
  cout << "| Input Size | Best Case | Avg Case | Worst Case | Randomized
|\n";
  cout << "+____________________________________+\n";

  fout << "size,best,avg,worst,randomized\n";

  for (int i = 0; i < 100; i++)
  {
    // rand() % (upperBound + 1 - lowerBound) +
    lowerBound size = rand() % (MAX_SIZE + 1 -
    MIN_SIZE) + MIN_SIZE;

    // Input Size
    cout << "| " << setw(10) <<
    size; fout << size << ",";

    // Best Case - Post Order of
    Balanced Tree AVLTree tree;
    for (int i = 0; i < size; i++)
```

```
  tree.root = tree.insert(i + 1, tree.root);
int *postArray =
tree.getPostOrderArray(size);
comparisons = quickSort(postArray, 0,
size - 1); cout << " | " << setw(9) << right
<< comparisons; fout << comparisons <<
",";

// Average
Case try
{
  ifstream
  fin("./random.txt"); for
  (int i = 0; i < size; i++)
    fin >>
  array[i];
  fin.close();
  comparisons = quickSort(array, 0, size - 1);
  cout << " | " << setw(9) << right <<
  comparisons; fout << comparisons <<
  ",";
}
catch (exception e)
{
  cerr <<
  e.what(); return
  -1;
}

// Worst Case
for (int i = 0; i < size;
  i++) array[i] = size
  - i;
comparisons = quickSort(array, 0, size - 1);
cout << " | " << setw(10) << right << comparisons;
```

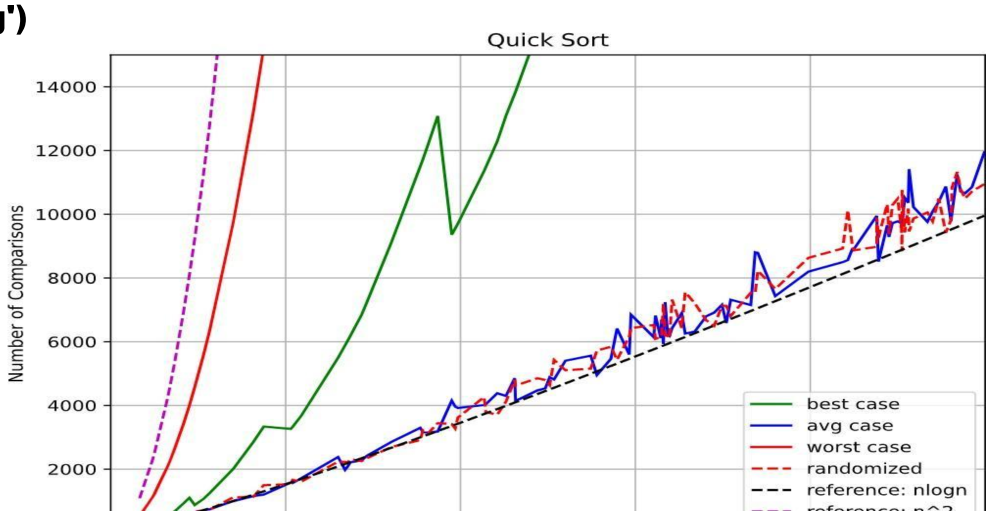
```
        fout << comparisons << ",";

        // Randomized Quick
        Sort for (int i = 0; i <
        size; i++)
          array[i] = i + 1;

        comparisons = randomizedQuickSort(array, 0,
        size - 1); cout << " | " << setw(9) << right <<
        comparisons << " |\n"; fout << comparisons <<
        "\n";
      }

      cout << "+___________________________________+\n";

      fout.close();

      return 0;
    }
    catch (exception e)
    {
      cerr << e.what();
      return -1;
    }
}
```

Output

Input Size	Best Case	Avg Case	Worst Case
273	1789	1703	1505
155	904	839	742
588	4526	4326	3859
834	6916	6594	5898
879	7363	7005	6260
633	4903	4755	4251
452	3277	3150	2784
991	8441	8039	7244
786	6324	6132	5477
515	3802	3651	3308
230	1476	1390	1220
698	5512	5306	4751
661	5147	5008	4481
955	7871	7690	6936
815	6572	6378	5720
473	3451	3321	2960
858	6980	6806	6121
982	8111	7959	7150
621	4785	4640	4157
674	5270	5114	4561
275	1855	1729	1520
531	3942	3805	3430
191	1180	1101	969
866	7041	6894	6190
204	1273	1208	1041
319	2188	2054	1825
811	6549	6358	5708
825	6657	6490	5869
433	3131	2992	2658
466	3381	3253	2928
619	4787	4601	4150
598	4571	4425	3973
574	4358	4189	3765
691	5442	5277	4712
750	5974	5792	5203
581	4417	4276	3827
377	2675	2552	2232
56	246	234	193
175	1054	991	868
868	7065	6898	6200
211	1335	1246	1099
372	2619	2502	2212
182	1105	1041	905
700	5614	5334	4792
697	5503	5342	4723
952	7837	7672	6875
76	362	347	286
674	5270	5114	4561
785	6315	6132	5526
602	4605	4448	3971
308	2085	1974	1759
144	811	769	678
313	2140	2007	1784
795	6386	6199	5572
195	1213	1133	997
138	767	737	629
101	528	482	426
809	6529	6441	5681
665	5178	5019	4499
344	2386	2268	2001
307	2074	1982	1748
252	1639	1553	1454
520	3849	3706	3349
185	1130	1064	923
227	1455	1361	1199
943	7751	7593	6841
176	1057	993	863
355	2494	2359	2092
800	6442	6268	5597
376	2660	2531	2220
53	230	221	182
460	3321	3198	2885
673	5261	5098	4554
838	6786	6623	5924
275	1855	1729	1520
92	463	424	380
755	6024	5839	5244
719	5710	5520	4929
901	7361	7184	6460
1000	8542	8095	7305
592	4496	4376	3884
434	3139	2989	2690
238	1542	1444	1255
138	767	737	629
723	5745	5562	4963
48	203	190	153
306	2067	1979	1738
156	904	853	730
484	3536	3417	3028
895	7327	7127	6431
96	492	460	396
400	2856	2726	2438
928	7605	7424	6711
188	1143	1082	948
951	7830	7644	6900
305	2061	1952	1734
214	1354	1273	1113
125	663	643	557
515	3802	3651	3308
739	5865	5702	5092
810	6539	6358	5670
667	5202	5063	4518

<u>Plotting of graph</u>

```
import math
import numpy as
np import pandas
as pd
import matplotlib.pyplot
as plt df =
pd.read_csv("results.csv"
) df =
df.sort_values("size")
plt.figure(figsize=(8, 6))
plt.plot(df['size'],
df['best'], 'g')
plt.plot(df['size'], df['avg'], 'b')
plt.plot(df['size'], df['worst'], 'r')
plt.plot(df['size'], df['randomized'], 'r--')
plt.plot(df['size'], df['size'] * np.log2(df['size']), 'k--')
plt.plot(df['size'], df['size'] ** 2, 'm--')
plt.legend(['best case', 'avg case', 'worst case', 'randomized',
'reference: nlogn', 'reference: n^2'])
plt.title('Quick Sort')
plt.xlabel('Input Size')
plt.ylabel('Number of
Comparisons') plt.ylim(0, 15000)
plt.xlim(0, 1000)
plt.grid()
plt.savefig('plot.p
ng')
```



5. Implement Radix Sort

Source code

```
#include
<cstdlib>
#include
<iomanip>
#include
<iostream>

#define MAX_SIZE 10

using namespace std;

int getMaximal(int A[], int n)
{
  int m = A[0];
  for (int i = 1; i < n;
    i++) if (A[i] > m)
      m = A[i];

  return m;
}

void countingSort(int A[], int n, int e)
{
  int i, B[n], C[10] = {0};

  for (i = 0; i < n;
    i++) C[(A[i] / e)
    % 10]++;
```

```
    for (i = 1; i < 10;
        i++) C[i] += C[i
        - 1];

    for (i = n - 1; i >= 0; i--)
    {
        B[C[(A[i] / e) % 10] - 1] = A[i];
        C[(A[i] / e) % 10]--;
    }

    for (i = 0; i < n;
        i++) A[i] =
        B[i];
}

void print(int A[], int n)
{
    for (int i = 0; i < n;
        i++) cout << A[i]
        << " ";
    cout << endl;

}

void radixSort(int A[], int n)
{
    int m = getMaximal(A, n);

    for (int e = 1, count = 1; (m / e) > 0; e *= 10, count++)
    {
        countingSort(A, n, e);
```

```
    cout << "Pass " << count
    << ": "; print(A, n);
  }
}

int main()
{
  try
  {
    srand(time(0));

    int
    array[MAX_SIZE];
    int size =
    MAX_SIZE;

    for (int i = 0; i < size; i++)
      array[i] = rand() % (1000 + 1 - 100) + 100;

    cout << "Before
    Sorting: "; print(array,
    size);

    cout << "Sorting using Radix Sort...\n";
    radixSort(array, size);

    cout << "After
    Sorting: ";
    print(array, size);

    return 0;
  }
```

```
  catch (exception e)
  {
     cerr << e.what();
     return -1;
  }
}
```

Output

```
Before Sorting: 884 864 399 687 791 646 571 127 480 720
Sorting using Radix Sort...
Pass 1: 480 720 791 571 884 864 646 687 127 399
Pass 2: 720 127 646 864 571 480 884 687 791 399
Pass 3: 127 399 480 571 646 687 720 791 864 884
After Sorting: 127 399 480 571 646 687 720 791 864 884
```

6. Implement Bucket Sort

Source code

```
#include <cmath>
#include <iomanip>
#include
<iostream> using
namespace std;

#define MAX_SIZE 8
#define MAX_BUCKETS 10
#define BUCKET_SIZE 10
```

```
template <class
T> class Node
{
public:
  T info;
  Node
  *next;
};

template <class T>
Node<T> *insertionSort(Node<T> *list)
{
  Node<T> *k, *nodeList;

  if (list == nullptr || list->next ==
    nullptr) return list;

  nodeList =
  list; k =
  list->next;
  nodeList->next = nullptr;

  while (k != nullptr)
  {
    Node<T> *ptr;

    if (nodeList->info > k->info)
    {
      Node<T> *temp = k;
```

```
        k = k->next;
        temp->next =
        nodeList; nodeList
        = temp; continue;
    }

    for (ptr = nodeList; ptr->next != 0; ptr = ptr->next)
    {
        if (ptr->next->info >
            k->info) break;
    }

    if (ptr->next != 0)
    {
        Node<T> *temp =
        k; k = k->next;
        temp->next =
        ptr->next; ptr->next
        = temp; continue;
    }
    else
    {
        ptr->next =
        k; k =
        k->next;
        ptr->next->next = nullptr;
        continue;
    }
}
```

```
    return nodeList;
}

template <class T>
int getBucketIndex(T value)
{
    return value * BUCKET_SIZE;
}

template <class T>
void BucketSort(T array[])
{
    int i, j;
    Node<T> **buckets;

    buckets = (Node<T> **)malloc(sizeof(Node<T> *) * MAX_BUCKETS);

    for (i = 0; i < MAX_BUCKETS; ++i)
        buckets[i] = nullptr;

    for (i = 0; i < MAX_SIZE; ++i)
    {
        int pos =
        getBucketIndex(array[i]);
        Node<T> *current = new
        Node<T>(); current->info =
        array[i];
        current->next =
        buckets[pos];
        buckets[pos] = current;
    }
```

```
        cout << "Binning..." << endl;

        for (i = 0; i < MAX_BUCKETS; i++)
        {
           cout << "\tBucket[" << i
           << "]: ";
           printBuckets(buckets[i]);
           cout << endl;
        }

        for (i = 0; i < MAX_BUCKETS; ++i)
 buckets[i]  =  insertionSort(buckets[i]);

cout << "Sorting within bins..." << endl;

        for (i = 0; i < MAX_BUCKETS; i++)
        {
           cout << "\tBucket[" << i
           << "]: ";
           printBuckets(buckets[i]);
           cout << endl;
        }

        cout << "Concatenating buckets..." << endl;

        for (j = 0, i = 0; i < MAX_BUCKETS; ++i)
        {
           Node<T> *node = buckets[i];
           while (node)
```

```
      {
         array[j++] =
         node->info; node =
         node->next;
      }
   }

   for (i = 0; i < MAX_BUCKETS; ++i)
   {
      Node<T> *node = buckets[i];
      while (node)
      {
         Node<T> *temp =
         node; node =
         node->next;
         free(temp);
      }
   }

   free(buckets);

   return;
}

template <class
T> void print(T
ar[])
{
   int i;
   for (i = 0; i <
      MAX_SIZE; ++i) cout
      << ar[i] << " ";
```

```
   cout << endl;
}

template <class T>
void printBuckets(Node<T> *list)
{
   Node<T> *cur = list;

   while (cur != nullptr)
   {
      cout << cur->info <<
      " "; cur = cur->next;
   }
}

int main()
{
   try
   {
      srand(time(0));

      double
      array[MAX_SIZE]; int
      size = MAX_SIZE;

      for (int i = 0; i < size; i++)
         array[i] = double(rand() % (1000 + 1 - 100) + 100) / double(1000);

      cout << "Before Sorting: ";
```

```
        print<double>(array);

        cout << "Sorting using Radix Sort...\n";
        BucketSort<double>(array);

        cout << "After
        Sorting: ";
        print<double>(array)
        ;

        return 0;
    }
    catch (exception e)
    {
        cerr << e.what();
        return -1;
    }
}
```

Output

```
Before Sorting: 0.935 0.926 0.594 0.205 0.109 0.301 0.901 0.15
Sorting using Radix Sort...
Binning...
    Bucket[0]:
    Bucket[1]: 0.15 0.109
    Bucket[2]: 0.205
    Bucket[3]: 0.301
    Bucket[4]:
    Bucket[5]: 0.594
    Bucket[6]:
    Bucket[7]:
    Bucket[8]:
Bucket[9]: 0.901 0.926 0.935
```

7. Implement Randomized Select

Source code

```
#include
<bits/stdc++.h> using
namespace std;
int partition(int arr[], int l, int r)
{
  int x = arr[r], i = l;
  for (int j = l; j <= r - 1;
    j++) { if (arr[j] <= x) {
      swap(arr[i],
      arr[j]); i++;
    }
  }
  swap(arr[i],
  arr[r]); return i;
}
int kthSmallest(int arr[], int l, int r, int k)
{
  if (k > 0 && k <= r - l + 1) {
    int index = partition(arr,
    l, r); if (index - l == k - 1)
      return
    arr[index]; if
    (index - l > k - 1)
      return kthSmallest(arr, l, index
    - 1, k); return kthSmallest(arr,
    index + 1, r,
             k - index + l - 1);
  }
  return INT_MAX;
```

```
}
int main()
{
  int arr[] = { 10, 4, 5, 8, 6, 11, 26 };
  int n = sizeof(arr) /
  sizeof(arr[0]); int k = 3;
  cout << "K-th smallest element is "

    << kthSmallest(arr, 0, n -
  1, k); return 0;
}
```

Output

```
K-th smallest element is 6
```

8. Implement Breadth-First Search in a graph

Source code

```
#include<iostrea
m> #include
<list>

using namespace std;

// This class represents a directed graph using
// adjacency list
representation class
Graph
```

```
{
  int V;  // No. of vertices

  // Pointer to an array containing adjacency
  // lists
  list<int>
  *adj;
public:
  Graph(int V); // Constructor

  // 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 = new list<int>[V];
}

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

void Graph::BFS(int s)
```

```
{
  // Mark all the vertices as not
  visited bool *visited = new
  bool[V];
  for(int i = 0; i < V;
    i++) visited[i] =
    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);

  // 'i' will be used to get all adjacent
  // vertices of a
  vertex
  list<int>::iterator
  i;

  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 (i = adj[s].begin(); i !=
    adj[s].end(); ++i)
```

```
      {
         if (!visited[*i])
         {
            visited[*i] = true;
            queue.push_back(*i);
         }
      }
   }
}

// Driver program to test methods of
graph class int main()
{
   // Create a graph given in the above
   diagram 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);

   return 0;
}
```

Output

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

9. Implement Depth-First-Search in a graph

Source code

```
// C++ program to print DFS traversal from
// a given vertex in a given
graph #include <bits/stdc++.h>
using namespace std;

// Graph class represents a directed graph
// using adjacency list
representation 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>::iterat
  or i;
  for (i = adj[v].begin(); i !=
    adj[v].end(); ++i) if (!visited[*i])
      DFS(*i);
}

// Driver
code int
main()
{
  // Create a graph given in the above
  diagram 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

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

10. . Write a program to determine the minimum spanning tree of a graph using both Prims and Kruskals algorithm.

Prims MST

Source code

```
// A C++ program for Prim's Minimum
// Spanning Tree (MST) algorithm. The program is
// for adjacency matrix representation of the graph
```

```
#include
<bits/stdc++.h> using
namespace std;

// Number of vertices in the
graph #define V 5

// A utility function to find the vertex with
// minimum key value, from the set of vertices
// not yet included in MST
int minKey(int key[], bool mstSet[])
{
  // Initialize min value
  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;
}

// A utility function to print the
// constructed MST stored in parent[]
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";
```

```
}

// Function to construct and print MST for
// a graph represented using adjacency
// matrix representation
void primMST(int graph[V][V])
{
  // Array to store constructed
  MST int parent[V];

  // Key values used to pick minimum weight
  edge in cut int key[V];

  // To represent set of vertices included
  in MST bool mstSet[V];

  // Initialize all keys as
  INFINITE for (int i = 0; i <
  V; i++)
    key[i] = INT_MAX, mstSet[i] = false;

  // Always include first 1st vertex in MST.
  // Make key 0 so that this vertex is picked as
  first vertex. key[0] = 0;
  parent[0] = -1; // First node is always root of MST

  // The MST will have V vertices
  for (int count = 0; count < V - 1; count++)
  {
```

```
        // Pick the minimum key vertex from the
        // set of vertices not yet included
        in MST int u = minKey(key,
        mstSet);

        // Add the picked vertex to the
        MST Set mstSet[u] = true;

        // Update key value and parent index of
        // the adjacent vertices of the picked vertex.
        // Consider only those vertices which are not
        // yet included in
        MST for (int v = 0; v
        < V; v++)

          // graph[u][v] is non zero only for adjacent vertices of m
          // mstSet[v] is false for vertices not yet included in MST
          // Update the key only if graph[u][v] is smaller than key[v]
          if (graph[u][v] && mstSet[v] == false &&
            graph[u][v] < key[v]) parent[v] = u, key[v] =
            graph[u][v];
    }

    // print the constructed
    MST printMST(parent,
    graph);
}

// Driver
code int
main()
{
    /* Let us create the following graph
```

```
   2 3
(0)--(1)--(2)
| / \ |
6| 8/ \5 |7
| / \ |
(3)-------(4)
     9    */
int graph[V][V] = { { 0, 2, 0, 6, 0 },
                  { 2, 0, 3, 8, 5 },
                  { 0, 3, 0, 0, 7 },
                  { 6, 8, 0, 0, 9 },
                  { 0, 5, 7, 9, 0 } };

// Print the solution
primMST(graph);

return 0;
}
```

Output

```
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```

Kruskals MST

Source code

```
#include<bits/stdc++.h
> using namespace
std;

// Creating shortcut for an
integer pair typedef pair<int,
int> iPair;

// Structure to represent a
graph struct Graph
{
  int V, E;
  vector< pair<int, iPair> > edges;

  // Constructor
  Graph(int V,
  int E)
  {
    this->V =
    V; this->E
    = E;
  }

  // Utility function to add an
  edge void addEdge(int u,
  int v, int w)
  {
    edges.push_back({w, {u, v}});
  }

  // Function to find MST using Kruskal's
```

```
  // MST
  algorithm int
  kruskalMST();
};

// To represent Disjoint
Sets struct
DisjointSets
{
  int *parent,
  *rnk; int n;

  // Constructor.
  DisjointSets(in
  t n)
  {
    // Allocate
    memory this->n
    = n;
    parent = new
    int[n+1]; rnk =
    new int[n+1];

    // Initially, all vertices are in
    // different sets and have
    rank 0. for (int i = 0; i <= n;
    i++)
    {
      rnk[i] = 0;

      //every element is parent of
      itself parent[i] = i;
    }
  }
```

```
    // Find the parent of a node 'u'
    // Path
    Compression int
    find(int u)
    {
        /* Make the parent of the nodes in
          the path from u--> parent[u] point to
          parent[u] */
        if (u != parent[u])
            parent[u] = find(parent[u]);
        return parent[u];
    }

    // Union by rank
    void merge(int x, int y)
    {
        x = find(x), y = find(y);

        /* Make tree with smaller
          height a subtree of the
          other tree */
        if (rnk[x] >
            rnk[y])
            parent[y] =
            x;
        else // If rnk[x] <=
            rnk[y] parent[x] =
            y;

        if (rnk[x] == rnk[y])
            rnk[y]++;
    }
};
```

```
/* Functions returns weight of the MST*/

int Graph::kruskalMST()
{
   int mst_wt = 0; // Initialize result

   // Sort edges in increasing order on basis
   of cost sort(edges.begin(), edges.end());

   // Create disjoint
   sets DisjointSets
   ds(V);

   // Iterate through all sorted
   edges vector< pair<int, iPair>
   >::iterator it;
   for (it=edges.begin(); it!=edges.end(); it++)
   {
      int u =
      it->second.first; int v
      = it->second.second;

      int set_u =
      ds.find(u); int
      set_v = ds.find(v);

      // Check if the selected edge is creating
      // a cycle or not (Cycle is created if u
      // and v belong to same
      set) if (set_u != set_v)
      {
```

```
        // Current edge will be in the MST
        // so print it
        cout << u << " - " << v << endl;

        // Update MST
        weight mst_wt +=
        it->first;

        // Merge two sets
        ds.merge(set_u, set_v);
      }
    }

    return mst_wt;
}

// Driver program to test above
functions int main()
{
    /* Let us create above shown
      weighted and undirected graph
      */
    int V = 9, E = 14;
    Graph g(V, E);

    // making above shown
    graph g.addEdge(0, 1, 4);
    g.addEdge(0, 7, 8);
    g.addEdge(1, 2, 8);
    g.addEdge(1, 7, 11);
```

```
    g.addEdge(2, 3, 7);
    g.addEdge(2, 8, 2);
    g.addEdge(2, 5, 4);
    g.addEdge(3, 4, 9);
    g.addEdge(3, 5, 14);
    g.addEdge(4, 5, 10);
    g.addEdge(5, 6, 2);
    g.addEdge(6, 7, 1);
    g.addEdge(6, 8, 6);
    g.addEdge(7, 8, 7);

    cout << "Edges of MST
    are \n"; int mst_wt =
    g.kruskalMST();

    cout << "\nWeight of MST is " << mst_wt;

    return 0;
}
```

Output

```
Edges of MST are
6 - 7
2 - 8
5 - 6
0 - 1
2 - 5
2 - 3
0 - 7
3 - 4

Weight of MST is 37
```

11. Write a program to solve the weighted interval scheduling problem.

Source code

```
#include
<iostream>
#include
<algorithm> using
namespace std;

int M[20];

// Job data
structure struct
Job {
```

```
  int s, // Start
   Time f, //
   Finish Time w;
   //weight
};

int p(Job jobs[], int j)
{

 for(int i = j - 1; i >= 0; i--)
  if (jobs[i].f <=
    jobs[j].s) return i;

 // return the negative index if no non-conflicting job
 is found return 0;
}

// Function to compare jobs used to sort them according to
finish time bool compareJob(Job j1, Job j2)
{
 return (j1.f < j2.f);
}

// Function to compute optimal value using
memoization int ComputeOpt(Job jobs[], int j)
{
 if (M[j] == -1)
 {
  M[j] = max(jobs[j].w + ComputeOpt(jobs, p(jobs, j)),
```

```
    ComputeOpt(jobs, j -
   1)); return M[j];
  }
}

// Function to print optimal
solution void FindSolution(Job
jobs[], int j)
{
  if (j == 0)
   cout <<
   "";
  else if ((jobs[j].w + M[p(jobs, j)]) > M[j - 1])
  {
   cout << j << " ";
   FindSolution(jobs,
   p(jobs, j));
  }
  else
   FindSolution(jobs, j - 1);
}

// Main function to find the optimal solution
void weightedIntervalScheduling(Job jobs[], int n)
{

  for(int i = 0; i < n; i++)
  {
   cout << endl << p(jobs, i) << " ";
  }
  cout << endl;
```

```
// Sort jobs according to
finish time sort(jobs, jobs + n,
compareJob);

// Find value of optimal
solution M[0] = 0;

// for(int j = 1; j < n; j++)
// M[j] = max(jobs[j].w + M[p(jobs, j)], M[j - 1]);

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

    int index = p(jobs, i);

    int incl =
    jobs[i].w; if
    (index != -1) {
       incl += M[index];
    }

    M[i] = max(incl, M[i-1]);
 }

cout << M[n-1] << " is the optimal value.";

// Find optimal solution
cout << "\nOptimal
Solution: ";
FindSolution(jobs, n);
}
```

```
/*
 Driver Code
*/
int main()
{

 int n;
 cout << "\nEnter the no of
 jobs: "; cin >> n;
 cout << "Enter the job details:\n";


 Job jobs[n];
 for(int i = 0; i < n; i++)

 {

  cout << "Starting time of Job " <<
  i+1 << ": "; cin >> jobs[i].s;
  cout << "Finishing time of Job " <<
  i+1 << ": "; cin >> jobs[i].f;
  cout << "Weight of Job " << i+1
  << ": "; cin >> jobs[i].w;
  cout << endl;

 }

 weightedIntervalScheduling(jobs, n);
```

```
 return 0;
}
```

Output:

```
Enter the no of jobs: 3
Enter the job details:
Starting time of Job 1: 1
Finishing time of Job 1: 3
Weight of Job 1: 2

Starting time of Job 2: 2
Finishing time of Job 2: 5
Weight of Job 2: 4

Starting time of Job 3: 3
Finishing time of Job 3: 7
Weight of Job 3: 5



0
0
0
5 is the optimal value.
Optimal Solution: 2
Process returned 0 (0x0)   execution time : 36.356 s
Press any key to continue.
```

12. Write a program to solve the 0-1 knapsack problem.

Source code

```
#include
<iostream> using
namespace std;

int SubsetSum(int set[], int n, int W)
{

 int M[n + 1][W + 1];

 for(int w = 0; w <= W;
  w++) M[0][w] = 0;

 for(int i = 1; i <= n; i++)
  for(int w = 0; w <= W;
  w++) if (w < set[i-1])
   {
    M[i][w] = M[i-1][w];
   }
   else
   {
    M[i][w] = max(M[i - 1][w], (set[i - 1] + M[i - 1][w - set[i - 1]]));
   }

 //--- Printing the array M
 cout << "Final
 Iteration:\n"; for(int i = 0;
 i <= n; i++)
```

```
 {

   for(int j = 0; j <= W;
   j++) cout << M[i][j]
   << " "; cout <<
   endl;
 }
 cout << endl;

 return M[n][W];
}


int main()
{

 int n, W;
 cout << "\nEnter the number of elements
 in set: "; cin >> n;
 int set[n];
 cout << "Enter the set
 elements: "; for(int i = 0; i <
 n; i++)
   cin >> set[i];
 cout << "Enter the
 sum: "; cin >> W;
 cout << endl;
 cout << SubsetSum(set, n, W) << " is the optimum solution value.\n";

 cout << endl;
```

return 0;

}

Output

```
Enter the number of elements in set: 5
Enter the set elements: 9 8 11 21 12
Enter the sum: 40

Final Iteration:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
0 0 0 0 0 0 0 0 8 9 9 9 9 9 9 9 9 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
0 0 0 0 0 0 0 0 8 9 9 11 11 11 11 11 11 17 17 19 20 20 20 20 20 20 20 20 28 28 28 28 28 28 28 28 28 28 28 28 28
0 0 0 0 0 0 0 0 8 9 9 11 11 11 11 11 11 17 17 19 20 21 21 21 21 21 21 21 28 29 30 30 32 32 32 32 32 32 38 38 40
0 0 0 0 0 0 0 0 8 9 9 11 12 12 12 12 12 17 17 19 20 21 21 23 23 23 23 23 28 29 30 31 32 33 33 33 33 33 38 38 40

40 is the optimum solution value.


Process returned 0 (0x0)   execution time : 40.006 s
Press any key to continue.
```