Source File: `~/2336/33/lab33.cpp`
Input: under control of `main` function
Output: under control of `main` function
Value: 3

The Shell sort, named after its inventor Donald Shell, provides a simple and efficient sorting algorithm. The sort begins by subdividing an n-element vector v into k sublists, which have members

$$\begin{array}{llll} v[0], & v[k+0], & v[2k+0], & \ldots \\ v[1], & v[k+1], & v[2k+1], & \ldots \\ \vdots & & & \\ v[k-1], & v[k+(k-1)], & v[2k+(k-1)], & \ldots \end{array}$$

A sublist starts with a first element $v[i]$ in the range from $v[0]$ through $v[k-1]$ and includes every successive k^{th} element. For instance, with $k = 4$ and the vector

7	5	8	6	2	4	9	1	3	0

the first sublist is

7	2	3

Sorting the sublist using the insertion sort yields

2	3	7

After placing the elements from the sorted sublist back in the original vector, we have

2	5	8	6	3	4	9	1	7	0

The second sublist from the above vector is

5	4	0

Sorting the sublist using the insertion sort yields

0	4	5

After placing the elements from the sorted sublist back in the original vector, we have

2	0	8	6	3	4	9	1	7	5

The third sublist from the above vector is

8	9

Sorting the sublist using the insertion sort yields

8	9

After placing the elements from the sorted sublist back in the original vector, we have

2	0	8	6	3	4	9	1	7	5

The fourth sublist from the above vector is

6	1

Sorting the sublist using the insertion sort yields

1	6

After placing the elements from the sorted sublist back in the original vector, we have

2	0	8	1	3	4	9	6	7	5

Repeat the process with successively smaller values of k, and continue through $k = 1$. When $k = 1$, the algorithm corresponds to the ordinary insertion sort that assures the vector is in order. The values of k the algorithm uses are called the *increment sequence*. It can be shown that a very effective increment sequence is to choose as the starting value of k the largest number from the sequence $1, 4, 13, 40, 121, 364, 1093, 4193, 16577, \ldots$ that is less than or equal to $n/9$. After each iteration, replace k with $k/3$ so that the increments move backward in the sequence from the starting value of k through $k = 1$. The data swapping occurs in noncontiguous segments of the vector, which moves an element a greater distance toward its final location than a swap of adjacent entries in the ordinary insertion sort. It can be shown that the Shell sort does less than $O(n^{3/2})$ comparisons for this increment sequence.

Write a function template to implement the Shell sort. You may use the following code to find the starting value for k.

```
1 for (k = 1; k <= n / 9; k = 3 * k + 1)
2   ; // null statement
```

A sample `main` function for testing your implementation is shown in Figure 1 and a sample execution sequence is shown in Figure 2. You will need to add a target of `lab33main` to the definition of `targets1srcfile` in your `Makefile`.

```
#include <cstdlib>
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <d_random.h>

using namespace std;

template <typename T>
void insertionSort(vector<T>& v)
{
  T *i, *j, *n = v.data() + v.size(), *start = v.data();
  T target;

  for (i = v.data() + 1; i < n; i++)
  {
    j = i;
    target = *i;
    while (j > start && target < *(j-1))
    {
      // shift elements up list to make room for insertion
      *j = *(j-1);
      j--;
    }
    // the location is found; insert target
    *j = target;
  }
}

template <typename T>
void shellSort(vector<T>& v);

#include <lab33.cpp>

const int N = 25000;
enum TEST_TYPE {RANDOM, ASCENDING, DESCENDING};

int main()
{
  vector<int> v, w, x;
  randomNumber rnd;
  TEST_TYPE testType;
  int i;

```

Figure 1. `/usr/local/2336/src/lab33main.C` (Part 1 of 2)

```
46    for (testType = RANDOM;
47         testType <= DESCENDING;
48         testType = static_cast<TEST_TYPE>(testType + 1))
49    {
50      if (!v.empty())
51        v.clear();
52      switch (testType)
53      {
54        case RANDOM:
55          for (i = 0; i < N; ++i)
56            v.push_back(rnd.random(1000000));
57          cout << "Random Data:" << endl;
58          break;
59        case ASCENDING:
60          for (i = 0; i < N; ++i)
61            v.push_back(i);
62          cout << "Ascending Data:" << endl;
63          break;
64        case DESCENDING:
65          for (i = 0; i < N; ++i)
66            v.push_back(N - i);
67          cout << "Descending Data:" << endl;
68          break;
69      }
70      x = v;
71      sort(x.begin(), x.end());
72
73      w = v;
74      auto start = chrono::system_clock::now();
75      shellSort(w);
76      auto stop = chrono::system_clock::now();
77      cout << "Shell Sort: "
78           << chrono::duration_cast<chrono::milliseconds>(stop-start).count()
79           << "ms" << endl;
80      if (x != w)
81        cout << "Sort didn't work correctly" << endl;
82      cout << endl;
83    }
84    return EXIT_SUCCESS;
85  }
```

Figure 1. `/usr/local/2336/src/lab33main.C` (Part 2 of 2)

```
newuser@csunix ~> cd 2336
newuser@csunix ~/2336> ./getlab.ksh 33
  * Checking to see if a folder exists for Lab 33. . .No
  * Creating a folder for Lab 33
  * Checking to see if Lab 33 has sample input and output files. . .No
  * Checking to see if /usr/local/2336/src/lab33main.C exists. . .Yes
  * Copying file /usr/local/2336/src/lab33main.C to folder ./33
  * Checking to see if /usr/local/2336/include/lab33.h exists. . .No
  * Copying file /usr/local/2336/src/Makefile to folder ./33
  * Adding a target of lab33main to targets1srcfile
  * Touching file ./33/lab33.cpp
  * Edit file ./33/lab33.cpp in Notepad++
newuser@csunix ~/2336> cd 33
newuser@csunix ~/2336/33> ls
Makefile     lab33.cpp    lab33main.C
newuser@csunix ~/2336/33> make lab33main
g++ -g -Wall -std=c++11 -c lab33main.C -I/usr/local/2336/include -I.
g++ -o lab33main lab33main.o -L/usr/local/2336/lib -lm -lbits
newuser@csunix ~/2336/33> ./lab33main
Random Data:
Shell Sort: 12ms

Ascending Data:
Shell Sort: 6ms

Descending Data:
Shell Sort: 7ms

newuser@csunix ~/2336/33>
```

Figure 2. Commands to Compile, Link, & Run Lab 33