Better Sorting Algorithms

1. Set Up
   - sorting a vector of pairs
2. Shell Sort
   - the Shell sort algorithm
   - C++ code for Shell sort
3. Merge Sort
   - the merge sort algorithm
   - C++ code for merge sort
4. Heap Sort
   - the heap sort algorithm
   - C++ code for heap sort

MCS 360 Lecture 31
Introduction to Data Structures
Jan Verschelde, 8 April 2020

Better Sorting Algorithms

1. Set Up
   - sorting a vector of pairs
2. Shell Sort
   - the Shell sort algorithm
   - C++ code for Shell sort
3. Merge Sort
   - the merge sort algorithm
   - C++ code for merge sort
4. Heap Sort
   - the heap sort algorithm
   - C++ code for heap sort

sorting a vector of pairs

Consider `vector< pair<int,char> > v` as a frequency table.
We sort only on the `int` key.

```
5 random items : (82,i)(42,d)(42,x)(54,r)(31,z)
```

With `char` as second data field
we can check if the sort is stable.
In a *stable* sort, equal keys retain their relative order.

We assume we sort container with *random-access iterator*.
For convenience: use subscripting operator `[ ]` on vectors.

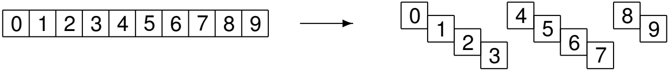
We generate random vectors of 2-digit keys.

Better Sorting Algorithms

Shell Sort

The idea of Donald Shell is to apply insertion sort to smaller subsequences separated by a gap value.

For example, when gap equals 4:



First, sort four sequences:
`[0, 4, 8]`, `[1, 5, 9]`, `[2, 6]`, and `[3, 7]`.
Then, repeat with smaller value of gap.

Start with gap value equal to $n/2$.
Divide each time gap by 2.2, at end set gap to 1.

The division factor of 2.2 appears to be empirically to give performance of $O(n^{5/4})$.

If the value for gap is $2^k - 1$, then the cost is $O(n^{3/2})$.

running an example

```
5 random items : (68,q)(79,f)(65,u)(58,j)(76,b)

insertion sort starts at 0 with gap 2
after 2 <-> 0 : (65,u)(79,f)(68,q)(58,j)(76,b)

insertion sort starts at 1 with gap 2
after 3 <-> 1 : (65,u)(58,j)(68,q)(79,f)(76,b)

insertion sort starts at 0 with gap 1
after 1 <-> 0 : (58,j)(65,u)(68,q)(79,f)(76,b)

after 4 <-> 3 : (58,j)(65,u)(68,q)(76,b)(79,f)

the sorted vector : (58,j)(65,u)(68,q)(76,b)(79,f)
```

Better Sorting Algorithms

1. Set Up
   - sorting a vector of pairs
2. Shell Sort
   - the Shell sort algorithm
   - C++ code for Shell sort
3. Merge Sort
   - the merge sort algorithm
   - C++ code for merge sort
4. Heap Sort
   - the heap sort algorithm
   - C++ code for heap sort

insertion sort with gap

```
void insertion_sort
 ( vector< pair<int,char> >& v,
   int start, int gap )
{
   for(int i=start+gap; i<v.size(); i=i+gap)
   {
      int j=i-gap;
      while((j >= start)
         && (v[i].first < v[j].first)) j=j-gap;
      j = j + gap;
      if(j < i)
      {
         pair<int,char> tmp = v[i];
         for(int k=i-gap; k>=j; k=k-gap)
            v[k+gap] = v[k];
         v[j] = tmp;
      }
   }
}
```

code for Shell sort

```
void sort ( vector< pair<int,char> >& v )
{
   int gap = v.size()/2;

   while(gap > 0)
   {
      for(int i=0; i<gap; i++)
         insertion_sort(v,i,gap);

      if(gap == 2)
         gap = 1;
      else
         gap = int(gap/2.2);
   }
}
```

Better Sorting Algorithms

Merge Sort

To sort a sequence of n elements, if $n > 1$:

1. split sequence in two equal halves,
2. sort first and second half,
3. merge the sorted halves.

Merge sort works on tapes, external data files.

Some data sets are too large to be stored entirely in internal memory.

running an example

```
4 random items : (15,m)(90,r)(63,p)(94,w)
in : (15,m)(90,r)(63,p)(94,w)
  in : (15,m)(90,r)
    in : (15,m)

    out : (15,m)
    in : (90,r)
    out : (90,r)

  out : (15,m)(90,r)

  in : (63,p)(94,w)
    in : (63,p)

    out : (63,p)

    in : (94,w)
    out : (94,w)
  out : (63,p)(94,w)

out : (15,m)(63,p)(90,r)(94,w)
the sorted vector : (15,m)(63,p)(90,r)(94,w)
```

Better Sorting Algorithms

1. Set Up
   - sorting a vector of pairs
2. Shell Sort
   - the Shell sort algorithm
   - C++ code for Shell sort
3. Merge Sort
   - the merge sort algorithm
   - C++ code for merge sort
4. Heap Sort
   - the heap sort algorithm
   - C++ code for heap sort

code for merge

```
vector< pair<int,char> > merge
 ( vector< pair<int,char> > u,
   vector< pair<int,char> > v )
{
   vector< pair<int,char> > w;
   int i = 0;
   int j = 0;
   while((i < u.size()) && (j < v.size()))
      if(u[i].first <= v[j].first)
         w.push_back(u[i++]);
      else
         w.push_back(v[j++]);
   while(i < u.size()) w.push_back(u[i++]);
   while(j < v.size()) w.push_back(v[j++]);

   return w;
}
```

function `merge_sort`

```
vector< pair<int,char> > merge_sort
 ( vector< pair<int,char> >& v )
{
   vector< pair<int,char> > w;
   if(v.size() < 2)
      w = v;
   else
   {
      int middle = v.size()/2;
      vector< pair<int,char> > a,b,s_a,s_b;
      for(int i=0; i<middle; i++) a.push_back(v[i]);
      for(int i=middle; i<v.size(); i++) b.push_back(v[i]);
      s_a = merge_sort(a);
      s_b = merge_sort(b);
      w = merge(s_a,s_b);
   }
   return w;
}
```

Cost Analysis

The cost of a sort depends on

1. the number of comparisons,
2. the number of assignments.

How many levels in the recursion?
For a sequence of size n: split $\log_2(n)$ times.

Splitting and merge takes $O(n)$ operations:

$$\frac{n}{2} + \frac{n}{4} + \cdots + 2 + 1 = n - 1.$$

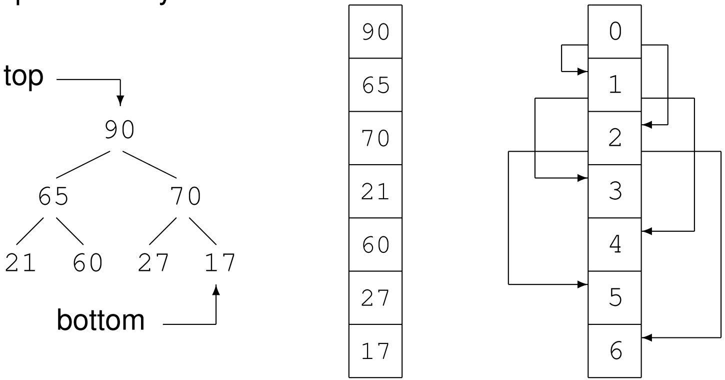
Best case? Worst case?

Better Sorting Algorithms

1. Set Up
   - sorting a vector of pairs
2. Shell Sort
   - the Shell sort algorithm
   - C++ code for Shell sort
3. Merge Sort
   - the merge sort algorithm
   - C++ code for merge sort
4. Heap Sort
   - the heap sort algorithm
   - C++ code for heap sort

a heap as a vector

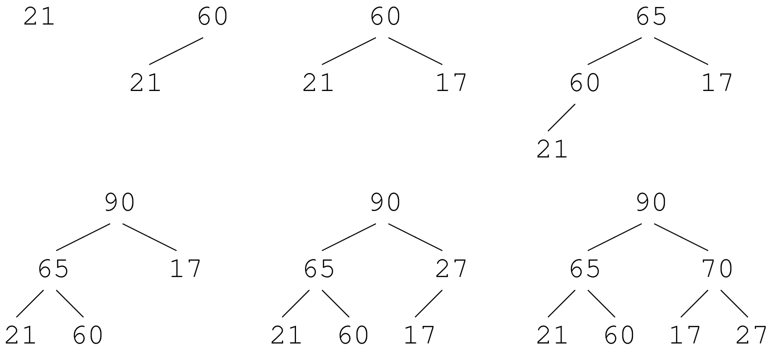
A heap is a binary tree:



For node at p: left child is at $2p + 1$, right child is at $2p + 2$.
Parent of node at p is at $(p - 1)/2$.

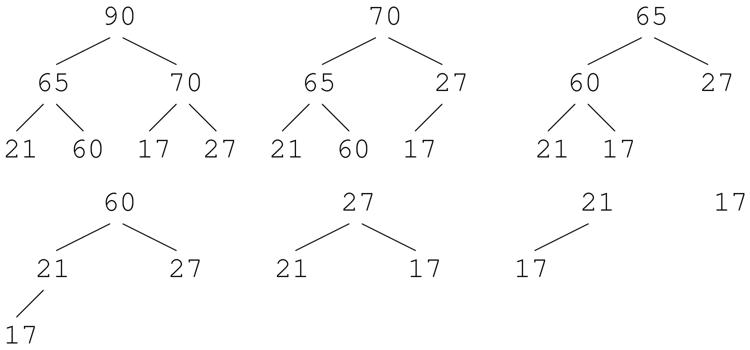
storing integer numbers

pushing `21 60 17 65 90 27 70`



sorting with a heap

popping largest element



running an example

```
5 random items : (81,x)(74,m)(78,i)(78,j)(50,i)

insert i=0 : (81,x)

insert i=1 : (81,x)(74,m)

insert i=2 : (81,x)(74,m)(78,i)

insert i=3 : (81,x)(78,j)(78,i)(74,m)

insert i=4 : (81,x)(78,j)(78,i)(74,m)(50,i)

remove i=0 : (78,i)(78,j)(50,i)(74,m)

remove i=1 : (78,j)(74,m)(50,i)

remove i=2 : (74,m)(50,i)

remove i=3 : (50,i)

remove i=4 :
the sorted vector : (50,i)(74,m)(78,j)(78,i)(81,x)
```

Better Sorting Algorithms

1. Set Up
   - sorting a vector of pairs
2. Shell Sort
   - the Shell sort algorithm
   - C++ code for Shell sort
3. Merge Sort
   - the merge sort algorithm
   - C++ code for merge sort
4. Heap Sort
   - the heap sort algorithm
   - C++ code for heap sort

code for heap sort

```
vector< pair<int,char> > sort
 ( vector< pair<int,char> >& v )
{
   vector< pair<int,char> > h,w;
   int bottom = -1;

   for(int i=0; i<v.size(); i++)
      insert(h,bottom,v[i]);

   for(int i=0; i<v.size(); i++)
   {
      w.insert(w.begin(),h[0]);
      remove(h,bottom);
   }
   return w;
}
```

Cost Analysis

Inserting and removing an element:

- depends on the depth of the binary tree,
- the heap is a complete binary tree,
- the depth is $\log_2(n)$.

Therefore: the cost for insert/remove is $O(\log_2(n))$.

We have n elements, so the cost is $2n \log_2(n)$.

Best case? Worst case?

Summary + Exercises

More of chapter 10 on sorting algorithms with Shell sort, heap sort, and merge sort.

Exercises:

1. Modify the code for Shell sort to count the #comparisons and #swaps. Run at least 10 sorts of random sequences of sizes 100, 200, and 400. Do data fitting on the counts. Do you observe $O(n^{3/2})$?
2. Describe an inplace merge sort. Extra input parameters are the begin and end index in the vector.
3. Give an example (with 7 numbers) of the best and worst case for heap sort.