IS 2610: Data Structures

Sorting

Feb 16, 2004

Sorting Algorithms: Bubble sort

- Bubble sort
  - Move through the elements exchanging adjacent pairs if the first one is larger than the second
  - Try out !
    - 2 6 3 1 5

```
#define key(A) (A)
#define less(A, B) (key(A) < key(B))
#define exch(A, B) { Item t = A; A = B; B = t; }

void selection (Item a[], int l, int r)
{
   int i, j;
   for (i = l; i < r; i++)
      for (j = r; j > r; j--)
         if (less(a[j-1], a[j]) exch(a[j-1], a[j]);
}
```

Sorting Algorithms: Bubble sort

- Recursive?
- Complexity
  - i^{th} pass through the loop – $(N\text{-}i)$ compare-and-exchange
  - Hence $N(N\text{-}1)/2$ compare-and-exchange is the worst case
  - What would be the minimum number of exchanges?

Sorting Algorithms: Insertion sort

- Insertion sort
  - “People” method

  2 6 3 1 5
- Complexity
  - About $N^2/4$
  - *About half of the left array*

```
#define less(A, B) (key(A) < key(B))
#define exch(A, B) { Item t = A; A = B; B = t; }
#define compexch(A, B) if (less(B, A)) exch(A, B)

void insertion(Item a[], int l, int r) {
   int i;
   for (i = r; i > l ; i--) compexch(a[i-1], a[i]);
   for (i = l+2; i <= r; i++)   {
      int j = i; Item v = a[i];
      while (less(v, a[j-1])) {
         a[j] = a[j-1]; j--;
      }
      a[j] = v;
   }
}
```

Sorting Algorithms: Shell sort

- Extend of insertion sort
  - Taking every h^{th} element will
  - Issues
    - Increment sequence?

h = 13
ASORTINGEXAMPLE
ASORTINGEXAMPLE
ASORTINGEXAMPLE
AEORTINGEXAMPLS

```
void shellsort(Item a[], int l, int r)
  { int i, j, h;
    for (h = 1; h <= (r-l); h = 3*h+1) ;
    for ( ; h > 0;)
      h = h/3;
      for (i = l+h; i <= r; i++)
        { int j = i; Item v = a[i];
          while (j >= l+h && less(v, a[j-h]))
            { a[j] = a[j-h]; j -= h; }
          a[j] = v;
        }
  }
```

Sorting Algorithms: Shell sort

h = 4

AEORTINGEXAMPLS

AEORTINGEXAMPLS

AEORTINGEXAMPLS

AENRTIOGEXAMPLS

AENRTIOGEXAMPLS

AENGTIOREXAMPLS

…

```
void shellsort(Item a[], int l, int r)
  { int i, j, h;
    for (h = 1; h <= (r-l); h = 3*h+1) ;
    for ( ; h > 0;)
      h = h/3;
      for (i = l+h; i <= r; i++)
        { int j = i; Item v = a[i];
          while (j >= l+h && less(v, a[j-h]))
            { a[j] = a[j-h]; j -= h; }
          a[j] = v;
        }
  }
```

Sorting Algorithms: Quick sort

- A divide and conquer algorithm
  - Partition an array into two parts
  - Sort the parts independently
  - Crux of the method is the partitioning process
    - Arranges the array to make the following three conditions hold
      - The element a[i] is in the final place in the array for some I
      - None of the elements in a[l]….a[i-1] is greater than a[i]
      - None of the elements in a[i+1]….a[r] is less than a[i]

Sorting Algorithms: Quick sort

```
int partition(Item a[], int l, int r);

void quicksort(Item a[], int l, int r)
  { int i;
    if (r <= l) return;
    i = partition(a, l, r);
    quicksort(a, l, i-1);
    quicksort(a, i+1, r);
  }
```

```
int partition(Item a[], int l, int r)
 { int i = l-1, j = r; Item v = a[r];
   for (;;)
     {
       while (less(a[++i], v)) ;
       while (less(v, a[--j])) if (j == l) break;
       if (i >= j) break;
       exch(a[i], a[j]);
     }
   exch(a[i], a[r]);
   return i;
 }
```

Sorting Algorithms: Quick sort

7 20 5 1 3 11 8 10 2 9

7 20 5 1 3 11 8 10 2 9

7 2 5 1 3 11 8 10 20 9

7 2 5 1 3 11 8 10 20 9

7 2 5 1 3 8 11 10 20 9

i *j*

7 2 5 1 3 8 9 10 20 11

i

```
int partition(Item a[], int l, int r)
 { int i = l-1, j = r; Item v = a[r];
   for (;;)
     {
       while (less(a[++i], v)) ;
       while (less(v, a[--j])) if (j == l) break;
       if (i >= j) break;
       exch(a[i], a[j]);
     }
   exch(a[i], a[r]);
   return i;
 }
```

Quick-sort characteristics

- Assume a sorted array
  - 1 2 3 4 5 6 7
- Assume a reverse array
  - 7 6 5 4 3 2 1
- Quick-sort uses about $N^2/2$ comparisons in the *worst case*
- Best case: when partition divides the input into exactly half: $C_N = 2C_{N/2} + N$
- Improvement: choose partitioning element that is more likely to divide the file near the middle
  - Random element

Merge-sort

- Sort two sub arrays
- Merge the sorted arrays

2 7 9 12 3 8 10 15

$i = 0$ $m = 3$ $r = 7$

```
void mergesort(Item a[], int l, int r)
 { int m = (r+l)/2;
   if (r <= l) return;
   mergesort(a, l, m);
   mergesort(a, m+1, r);
   merge(a, l, m, r);
 }
```

- Complexity expression:
  - $M_N = M_{cl(N/2)} + M_{fl(N/2)} + N$
  - $N\ lg\ N$ comparisons

Merge-sort

2 7 9 12 3 8 10 15

$i = 0$ $m = 3$ $r = 7$

aux: (bitonic)

2 7 9 12 15 10 8 3

```
 Item aux[maxN];
merge(Item a[], int l, int m, int r)
  { int i, j, k;
    for (i = m+1; i > l; i--) aux[i-1] = a[i-1];
    for (j = m; j < r; j++) aux[r+m-j] = a[j+1];
    for (k = l; k <= r; k++)
       if (less(aux[i], aux[j]))
          a[k] = aux[i++]; else a[k] = aux[j--];
  }
```

Create a = 2 3 7 8 9 10 12 15

Merge-sort

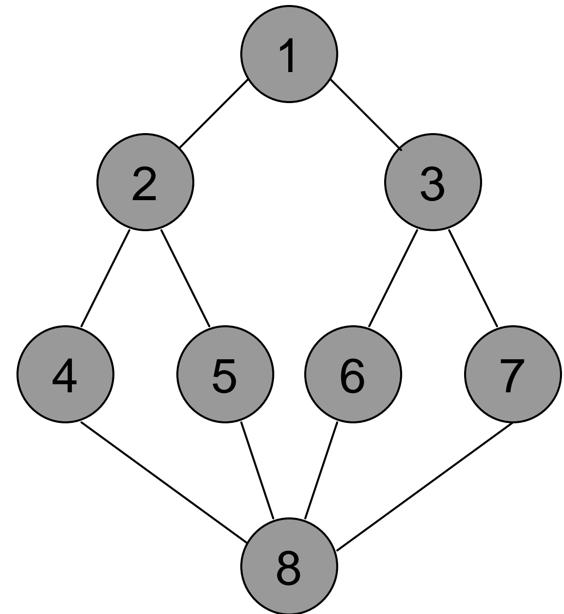
Graph

- A graph is a set of nodes (V) together with a set of edges (E) that connect pairs of distinct nodes (with at most one edge connecting any pair of nodes)
- Graph traversal
  - Depth first search
  - Breadth first search

Depth first

- Steps
  - Visit v
  - Recursively visit each unvisited node attached to v

```
void traverse(int k, void (*visit)(int))
 { link t;
   (*visit)(k); visited[k] = 1;
   for (t = adj[k]; t != NULL; t = t->next)
     if (!visited[t->v]) traverse(t->v, visit);
 }
```



Breadth first

- Steps
  - Visit v
  - Visit all nodes connected to v first

```
void traverse(int k, void (*visit)(int))
 { link t;
   QUEUEinit(V); QUEUEput(k);
   while (!QUEUEempty())
     if (visited[k = QUEUEget()] == 0)
       {
         (*visit)(k); visited[k] = 1;
         for (t = adj[k]; t != NULL; t = t->next)
           if (visited[t->v] == 0) QUEUEput(t->v);
       }
 }
```