The University of Auckland

Data Sorting

- **Ordering relation:** places each pair α, β of ***countable*** items in a fixed order denoted as (α,β) or $\langle\alpha,\beta\rangle$
- **Order notation:** $\alpha \le \beta$ (*less than or equal to*)
- **Countable item:** labelled by a specific ***integer key***
- **Comparable objects in Java:** if an object can be ***less than***, ***equal to***, or ***greater than*** another object:

  object1.compareTo(object2) **<0, =0, >0**

The University of Auckland

Order of Data Items

- **Numerical order** - by value:
  $5 \le 5 \le 6.45 \le 22.79 \le \ldots \le 1056.32$
- **Alphabetical order** - by position in an alphabet:
  a ≤ b ≤ c ≤ d ≤ … ≤ z
  Such ordering depends on the alphabet used: look into any bilingual dictionary...
- **Lexicographic order** - by first differing element:
  5456 ≤ 5457 ≤ 5500 ≤ 6100 ≤ …
  pork ≤ ward ≤ word ≤ work ≤ …

The University of Auckland

Features of ordering

- Relation on an array $\mathbf{A} = \{a, b, c, \ldots\}$ is:
  - ***reflexive:*** $a \le a$
  - ***transitive:*** if $a \le b$ and $b \le c$, then $a \le c$
  - ***symmetric:*** if $a \le b$ and $b \le a$, then $a = b$
- **Linear order** if for any pair of elements a and b either $a \le b$ or $b \le a$: $a \le b \le c \le \ldots$
- **Partial order** if there are incomparable elements

The University of Auckland

Insertion Sort

- Splits an array into a unordered and ordered parts
- Sequentially contracts the unordered part, one element per stage:

  ordered part unordered part

  $\boxed{a_0, \ldots, a_{i-1}}$ $\boxed{a_i, \ldots, a_{n-1}}$

- Stage $i = 1, \ldots, n-1$:

  $n - i$ unordered and i ordered elements

The University of Auckland

Example of Insertion Sort

- N_c - number of comparisons per insertion
- N_m - number of moves per insertion

13	18	35	44	15	10	20	N_c	N_m
			15	44			<	→
		15	35				<	→
	15	18					<	→
15							≥	
13	15	18	35	44	10	20	4	3

The University of Auckland

Example of Insertion Sort

13	15	18	35	44	10	20	N_c	N_m
				10	44		<	→
			10	35			<	→
		10	18				<	→
	10	15					<	→
10	13						<	→
10	13	15	18	35	44	20	5	5

The University of Auckland

Implementation of Insertion Sort

```
begin InsertionSort ( integer array a of size n )
   for i ← 1 while i < n step i ← i + 1 do
      s_tmp ← a[ i ];    k ← i − 1
      while k ≥ 0 AND s_tmp < a[k] do
         a[ k+1 ] ← a[ k ];    k ← k − 1
      end while
      a[ k+1 ] ← s_tmp
   end for
end InsertionSort
```


The University of Auckland

Average Complexity at Stage i

- $i + 1$ positions to place a next item: $0\ 1\ 2\ \ldots\ i\text{-}1\ i$
- $i - j + 1$ comparisons and $i - j$ moves for each position $j = i, i{-}1, \ldots, 1$
- i comparisons and i moves for position $j = 0$
- Average number of comparisons:

$$E_i = \frac{1+2+\ldots+i+i}{i+1} = \frac{i}{2} + \frac{i}{i+1}$$

The University of Auckland

Total Average Complexity

- $n - 1$ stages for n input items: the total average number of comparisons:

$$E = E_1 + E_2 + \ldots + E_{n-1} = \tfrac{n^2}{4} + \tfrac{3n}{4} - H_n$$

$$\left(\tfrac{1}{2}+\tfrac{1}{2}\right)+\left(\tfrac{2}{2}+\tfrac{2}{3}\right)+\ldots+\left(\tfrac{n-1}{2}+\tfrac{n-1}{n}\right)$$

$$= \tfrac{1}{2}(1+2+\ldots+(n-1)) + \left(\tfrac{1}{2}+\tfrac{2}{3}+\ldots+\tfrac{n-1}{n}\right)$$

$$= \tfrac{(n-1)n}{4} + n - \left(\tfrac{1}{2}+\tfrac{1}{3}+\ldots+\tfrac{1}{n}\right)$$

- $H_n \cong \ln n + 0.577$ when $n \to \infty$ is the n-th **harmonic** number

The University of Auckland

Analysis of Inversions

- An **inversion** in an array $\mathbf{A} = [a_1, a_2, \ldots, a_n]$ is any ordered pair of positions (i, j) such that $i < j$ but $a_i > a_j$: e.g., $[\ldots, 2, \ldots, 1]$ or $[100, \ldots, 35, \ldots]$

A	# invers.	$\mathbf{A}_{\text{reverse}}$	# invers.	Total #
3,2,5	1	5,2,3	2	3
3,2,5,1	4	1,5,2,3	2	6
1,2,3,4,7	0	7,4,3,2,1	10	10

The University of Auckland

Analysis of Inversions

- Total number of inversions both in an arbitrary array $\mathbf{A}$ and its reverse $\mathbf{A}_{\text{reverse}}$ is equal to the total number of the ordered pairs ($i < j$):

$$\binom{n}{2} = \frac{(n-1)\cdot n}{2}$$

- A sorted array has no inversions
- A reverse sorted array has $\frac{(n-1)n}{2}$ inversions

The University of Auckland

Analysis of Inversions

- Exactly one inversion is removed by swapping two neighbours $a_{i-1} > a_i$
- An array with k inversions results in $O(n + k)$ running time of insertionSort
- Worst-case time: $c\,\frac{n^2}{2}$, or $O(n^2)$
- Average-case time: $c\,\frac{n^2}{4}$, or $O(n^2)$

More Efficient Shell's Sort

- **Efficient sort** must eliminate more than just one inversion between the neighbours per exchange! (insertion sort eliminates one inversion per exchange)
- D.Shell (1959): compare first the keys at a distance of gap_T, then of $\text{gap}_{T-1} < \text{gap}_T$, and so on until of $\text{gap}_1 = 1$
- After a stage with gap_t, all elements spaced gap_t apart are sorted; it can be proven that any gap_t-sorted array remains gap_t-sorted after being then gap_{t-1}-sorted

Lecture 6 COMPSCI.220.FS.T - 2004 13

Implementation of ShellSort

```
begin Shell Sort ( integer array a of size n )
1.   for gap ← ⌊n/2⌋ while gap > 0
      step gap ← ( if gap = 2 then 1 else ⌊gap/2.2⌋ ) do
2.    for i ← gap while i < n step i ← i + 1 do
3.      s_tmp ← a[i];  k ← i
4.      while( k ≥ gap AND s_tmp < a[ k – gap ] do
5.        a[ k ] ← a[ k – gap ];  k ← k – gap
6.      end while
7.      a[ k ] ← s_tmp;  8.  end for; 9.  end for
end Shell Sort
```

Lecture 6 COMPSCI.220.FS.T - 2004 14

Example of ShellSort: step 1

gap	i:C:M	Data to be sorted									
		25	8	2	91	70	50	20	31	15	65
5	5:1:0	25					50				
	6:1:0		8					20			
	7:1:0			2					31		
	8:1:1				**15**					**91**	
	9:1:1					**65**					**70**
		25	8	2	15	65	50	20	31	91	70

Lecture 6 COMPSCI.220.FS.T - 2004 15

Example of ShellSort: step 2

gap	i:C:M	25	8	2	15	65	50	20	31	91	70
2	2:1:1	**2**		**25**							
	3:1:0		8		15						
	4:1:0			25		65					
	5:1:0				15		50				
	6:3:2	2		**20**		**25**		**65**			
	7:2:1				15		**31**		**50**		
	8:1:0							65		91	
	9:1:0								50		70

Lecture 6 COMPSCI.220.FS.T - 2004 16

Example of ShellSort: step 3

gap	i:C:M	2	8	20	15	25	31	65	50	91	70
1	1:1:0	2	8								
	2:1:0		8	20							
	3:2:1		8	**15**	**20**						
	4:1:0				20	25					
	5:1:0					25	31				
	6:1:0						31	65			
	7:2:1						31	**50**	**65**		
	8:1:0								65	91	
	9:2:1								65	**70**	**91**

Lecture 6 COMPSCI.220.FS.T - 2004 17

Time complexity of ShellSort

- Heavily depends on gap sequences
- Shell's sequence: $^n/_2, {}^n/_4, \ldots, 1$:

  $O(n^2)$ worst; $O(n^{1.5})$ average
- "Odd gaps only" (if even: $^{\text{gap}}/_2 + 1$):

  $O(n^{1.5})$ worst; $O(n^{1.25})$ average
- Heuristic sequence: $^{\text{gap}}/_{2.2}$: better than $O(n^{1.25})$
- A very simple algorithm with an extremely complex analysis

Lecture 6 COMPSCI.220.FS.T - 2004 18