Sorting

- A way to organize data
- Facilitates retrieval.
- The ***need*** to sort is entirely dependent on the way the data is used
- First, determine if sorting is needed
- Match the sort to the problem.
- Sorts are not like greeting cards: There is no all-occasion sort.

Sorting Example

SORTING RECORDS

APPLICATION : WOODSHOP INVENTORY .....

UNSORTED RECORDS:

QUANTITY	WOOD	COST	SIZE
3	OAK	...	.....
2	PINE	...	.....
2	WALNUT	...	.....
1	MAPLE	...	.....

SORTED RECORDS: "SORTED BY QUANTITY"

QUANTITY	WOOD	COST	SIZE
1	MAPLE	...	.....
2	PINE	...	.....
2	WALNUT	...	.....
3	OAK	...	.....

Basic Terminology

- record - fixed or variable
- file (of size n)
- key
  - part of record examined
  - primary vs. secondary
- external key
- external vs. internal sorting

SORTING BY ADDRESS

ADVANTAGE:

INCREASES EFFICIENCY - RECORDS ARE NOT MOVED FROM ONE STORAGE LOCATION TO ANOTHER

APPROACH:

MAINTAIN A POINTER TABLE OF INDEX VALUES

SORT INDEX VALUE NOT DATA

UNSORTED POINTER TABLE:	QUANTITY	WOOD	COST	SIZE	. . .	SORTED POINTER TABLE:
1	7	BERCH	....	.....		6
2	3	OAK	...	.....		4
3	4	BIRCH	...	....		5
4	2	PINE	...	....		2
5	2	WALNUT	...	....		3
6	1	MAPLE	...	....		1

Points to consider

- size of data
- order of data
- distribution of data
- reuse of existing code
- programmer time
- frequency of sorting
- how is data used
- efficiency of sorts available
- space requirements

Sort Types

- Exchange Sorts
- Selection Sorts
- Insertion Sorts
- Merge Sorts
- Radix Sorts

- Range **O**(n) to **O**(n^3) overall
  O(n) to **O**(n^2) typical sorts
  O(n logn) to **O**(n^2) on ave.

123 - Inorder data
321 - Reverse order data
Ran - Random data

Sort Types

	Best	Worst	AVE	Sensitive to Order	Space Requirements	Implementation
Exchange Sorts						
Bubble Sort						
Quicksort						
Selection Sorts						
Simple Selection Sort						
Binary Tree Sorts						
Simple BT Sort						
HeapSort						
Quadratic Selection						
Insertion Sorts						
Simple Insertion Sort						
Shell Sort						
Address Calculation						
Merge Sorts						
Straight Merge						
Natural Merge						
Radix Sorts						

Exchange Sorts

- Bubble Sort
- Quicksort

Selection Sorts

- Straight (Simple) Selection Sort
- Binary Tree Sort
- Simple BT Sort
- HeapSort
- Quadratic Selection Sort

Insertion Sorts

- Simple Insertion Sort
- Shell Sort
- Address Calculation Sort

Merge Sorts

- Straight Merge
- Natural Merge

Bubble Sort

- Simple, inefficient
- Good for small data files
- Good for throwaway appl.
- Simple double nested for-loop
- Easy modifications

Bubble Sort

C	E	25	57	48	37	12	92	86	33
7	5	25	48	37	12	57	86	33	92
7	3	25	37	12	48	57	33	86	92
7	2	25	12	37	48	33	57	86	92

Bubble Sort

C	E	25	57	48	37	12	92	86	33
7	5	25	48	37	12	57	86	33	92
6	3	25	37	12	48	57	33	86	92
5	2	25	12	37	48	33	57	86	92
4	2	12	25	37	33	48	57	86	92
3	1	12	25	33	37	48	57	86	92
2	0	12	25	33	37	48	57	86	92
1	0	12	25	33	37	48	57	86	92

Quicksort

- Quite efficient
- Recursive
- Select pivot
- Do partitioning
- Repeat on partitions until sorted
- Any sort using this strategy is a Quicksort
- Lots of ways to do partitioning
- Many ways to select pivot

Quicksort - Partitioning

1) Select Pivot - Make copy
2) Assign Down Ptr to start of partition
3) Assign Up Ptr to end of partition
4) Move Up Left to find item smaller than pivot
5) Copy item to Down
6) Move Down Right to find item bigger than pivot
7) Copy value to Up
8) Repeat (4) - (7) until Up == Down
9) Copy Pivot into Down

Quicksort

Pivot	25	57	48	37	12	92	86	33
25	25^{D}	57	48	37	12	92	86	33^{U}
25	25^{D}	57	48	37	12^{U}	92	86	33
25	12^{D}	57	48	37	12^{U}	92	86	33
25	12	57^{D}	48	37	12^{U}	92	86	33
25	12	57^{D}	48	37	57^{U}	92	86	33
25	12	57^{DU}	48	37	57	92	86	33
	(12)	25	(48	37	57	92	86	33)

Quicksort

Pivot	(12) 25 (48 37 57 92 86 33)
48	(12) 25 (48^{D} 37 57 92 86 33^{U})
48	(12) 25 (33^{D} 37 57 92 86 33^{U})
48	(12) 25 (33 37 57^{D} 92 86 33^{U})
48	(12) 25 (33 37 57^{D} 92 86 57^{U})
48	(12) 25 (33 37 57^{DU} 92 86 57)
	(12) 25 (33 37) 48 (92 86 57)

Quicksort

Pivot	(12) 25 (33 37) 48 (92 86 57)
33	(12) 25 (33^{D} 37^{U}) 48 (92 86 57)
33	(12) 25 (33^{DU} 37) 48 (92 86 57)
	(12) 25 33 (37) 48 (92 86 57)
92	(12) 25 33 (37) 48 (92^{D} 86 57^{U})
92	(12) 25 33 (37) 48 (57^{D} 86 57^{U})
92	(12) 25 33 (37) 48 (57 86 57^{DU})
	(12) 25 33 (37) 48 (57 86) 92

Quicksort

Pivot	(12) 25 33 (37) 48 (57 86) 92
57	(12) 25 33 (37) 48 (57^{D} 86^{U}) 92
57	(12) 25 33 (37) 48 (57^{UD} 86) 92
	(12) 25 33 (37) 48 57 (86) 92
	12 25 33 37 48 57 86 92

Quicksort - Pivot Selection

Straight (Simple) Selection Sort

- Simple, inefficient
- Not used much
- Selected items put in final position

Straight (Simple) Selection Sort

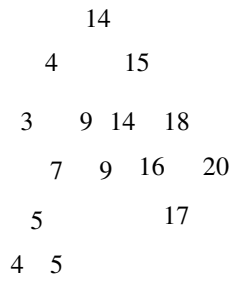
C	E	25	57	48	37	12	92	86	33
7	1	25	57	48	37	12	33	86	92
6	0	25	57	48	37	12	33	86	92
5	1	25	33	48	37	12	57	86	92
4	1	25	33	12	37	48	57	86	92
3	1	25	33	12	37	48	57	86	92
2	1	25	12	33	37	48	57	86	92
1	1	12	25	33	37	48	57	86	92

Binary Tree Sort

- Simple BT Sort
- Builds general tree
- Will do with trees
- Uses Inorder traversal to get sorted ordering

Binary Trees: Application

Recall the general tree built earlier:



What do you get when you do an inorder transversal of the tree?

Binary Trees: Application

A binary tree sort is a type of selection sort. It has two phases:

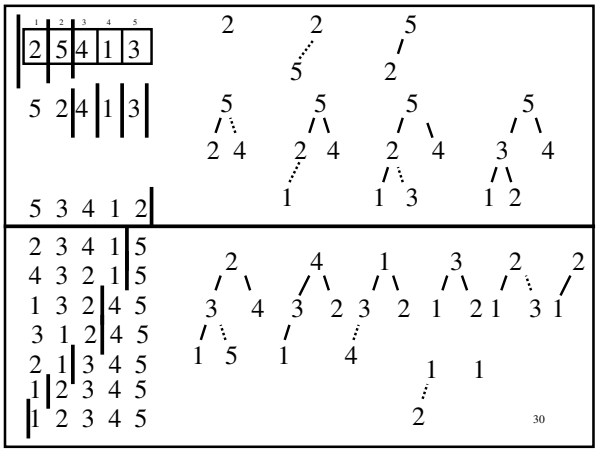
1) Build the tree (with Search Tree characteristic) using MakeTree, SetLeft, SetRight

2) Create the sorted list by doing the inorder transversal.

28

HeapSort

- Heaps are used for a type of selection sort called Heap Sort.
- Efficient
- Uses sequential array representation of a tree
- This is a two phase sort:
  - 1) Build Heap
  - 2) Generate sorted list.



30

Quadratic Selection Sort

5 8 13 16 | 2 6 10 14 | 11 1 4 15 | 7 3 9 12

16	14	15	12	⇨	16
13	14	15	12	⇨	15
13	14	11	12	⇨	14
13	10	11	12	⇨	13
8	10	11	12	⇨	12
8	10	11	9	⇨	11

etc…..

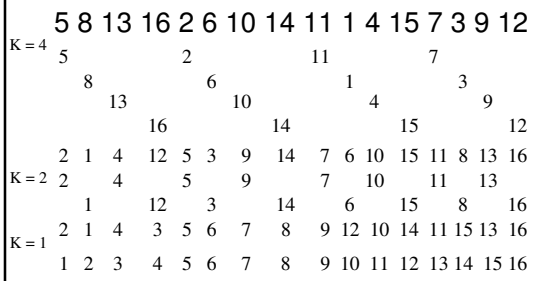
Simple Insertion Sort

C	25 57 48 37 12 92 86 33
0	25 ⟵ List
1	25 ← 57 ⟵ List
2	25 ← 48 ← 57 ⟵ List
3	25 ← 37 ← 48 ← 57 ⟵ List
4	12 ← 25 ← 37 ← 48 ← 57 ⟵ List
1	12 ← 25 ← 37 ← 48 ← 57 ← 92 ⟵ List
2	12 ← 25 ← 37 ← 48 ← 57 ← 86 ← 92 ⟵ List
6	12 ← 25 ← 33 ← 37 ← 48 ← 57 ← 86 ← 92 ⟵ List

Shell Sort

- see D. Knuth, Art of Computer Programming, Vol. I.
- Create subfiles - not usual way
- Use insertion sort within subfile
- Multiple passes
- How do you set up subfile sizes

Shell Sort



Shell Sort - K values

Optimal sequence per Knuth

Recurrence Relation

h(1) = 1

h(n) = 3*h(n-1) + 1

1 4 13 40 121 364 1093 ...

Address Calculation Sort

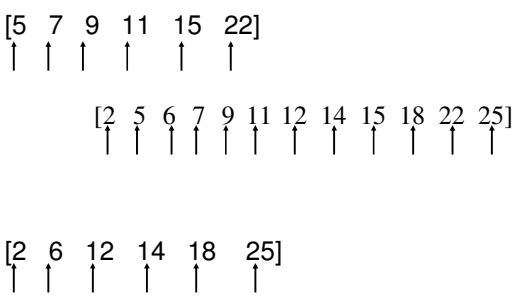
- Bucket Sort
- Will come back to after Hashing

Straight 2-way Merge

- Basis for external sorting
- Can do higher orders
- Pretend 1 file of size n is n files of size 1
- A file of size 1 is trivially sorted
- Works with arrays or lists

Straight 2-way Merge:

Sample Merging



Straight 2-way Merge

[5] [8] [13] [16] [2] [6] [10] [14] [11] [1] [4] [15] [7] [3] [9] [12]

[5 8] [13 16] [2 6] [10 14] [1 11] [4 15] [3 7] [9 12]

[5 8 13 16] [2 6 10 14] [1 4 11 15] [3 7 9 12]

[2 5 6 8 10 13 14 16] [1 3 4 7 9 11 12 15]

[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]

4 complete passes needed

Natural Merge

- Uses replacement selection
- Merging part is still the same
- Exploits order in data

Natural Merge

[5 8 13 16][2 6 10 14][11][1 4 15][7][3 9 12]

[2 5 6 8 10 13 14 16] [1 4 11 15] [3 7 9 12]

[1 2 4 5 6 8 10 11 13 14 15 16] [3 7 9 12]

[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]

4 passes:

3 merge passes plus one to set up the initial subfiles

Radix Sorts

2636	8410	3902	8410	2636
8410	3902	2907	8625	2907
8625	8625	8410	2636	3902
2907	2636	8625	3902	8410
3902	2907	2636	2907	8625

Radix Sorts

Sally	Sue	
Susan	Susan	Pad with blanks on end if string of different lengths.
Sue	Sally	"blank" < "all other characters"
Sheila	Sheila	
Serena	Serena	

Radix Sorts

The sorting within a column is based on Counting Sort (see Cormen, et al.), if interested.

Radix sort is an example of a Stable sort

If A comes before B, and on pass i, A and B are duplicates, then at the end of pass i, A still comes before B