

Sorting

Selection Sort

- ◆ For each pass, select the next largest/smallest number and put it in its place
 - Ex: Look for the smallest item, swap it with the item in the first position
- ◆ N-1 passes

Selection Sort

pass	12	4	2	26	7	30	19	11	21
------	----	---	---	----	---	----	----	----	----

Selection Sort

pass	12	4	2	26	7	30	19	11	21
P=1	2	4	12	26	7	30	19	11	21
P=2	2	4	12	26	7	30	19	11	21
P=3	2	4	7	26	12	30	19	11	21
P=4	2	4	7	11	12	30	19	26	21
P=5	2	4	7	11	12	30	19	26	21
P=6	2	4	7	11	12	19	30	26	21
P=7	2	4	7	11	12	19	21	26	30
P=8	2	4	7	11	12	19	21	26	30

Selection Sort

```
void selectionSort ( itemType A[], int N )
    int smallpos
    itemType item
    for (int s = 0; s < N-1; s++)
        temp = A[s]
        smallpos = s
        for ( int x=s+1; x < N; x++)
            if (A[x] < A[smallpos])
                smallpos = x
        A[s] = A[smallpos]
        A[smallpos] = temp
```

Running time = $O(n^2)$

Bubble Sort

- ◆ Compare adjacent items and exchange them if they are out of order
- ◆ N-1 passes

Bubble Sort

pass	10	14	2	16	7	21	8
P=1	10	2	14	7	16	8	21
P=2	2	10	7	14	8	16	21
P=3	2	7	10	8	14	16	21
P=4	2	7	8	10	14	16	21
P=5	2	7	8	10	14	16	21

Bubble Sort

BubbleSort

```
for (int P = 1; P < N; P++)
    for (int j = 0; j < N-1; j++)
        if ( a[j] > a[j+1] )
            swap (a[j], a[j+1])
```

Running time = $O(n^2)$ for naive implementation

Insertion Sort

- ◆ Make N-1 passes
- ◆ For P=1 through N-1, ensure that the items in positions 0 through P are sorted
- ◆ Insert next item into correct position in already sorted set, now set is one bigger

12/5/04

Sorting

9

Insertion Sort

pass	34	26	8	61	17	51	32	9	22
------	----	----	---	----	----	----	----	---	----

12/5/04

Sorting

10

Insertion Sort

pass	34	26	8	61	17	51	32	9	22
P=1	26	34	8	61	17	51	32	9	22
P=2	8	26	34	61	17	51	32	9	22
P=3	8	26	34	61	17	51	32	9	22
P=4	8	17	26	34	61	51	32	9	22
P=5	8	17	26	34	51	61	32	9	22
P=6	8	17	26	32	34	51	61	9	22
P=7	8	9	17	26	32	34	51	61	22
P=8	8	9	17	22	26	32	34	51	61

12/5/04

Sorting

11

Insertion Sort

```

void InsertionSort ( itemtype A[], int N )
    itemtype temp
    for ( int P = 1; P < N; P ++ )
        temp = A[P]
        for ( int x=P; x > 0 && A[x-1] > temp; x-- )
            A[x] = A[x-1]
        A[x] = temp
    
```

Running time = $O(n^2)$ $O(n)$ if presorted
 Worst case - reverse order

12/5/04

Sorting

12

Shell Sort

- ◆ Named after Donald Shell
- ◆ Compare distant items then decrease distance
- ◆ Diminishing increment sort
- ◆ Multiple passes, each time sorts a set of numbers

12/5/04

Sorting

13

Shell Sort

- ◆ With each pass, set size gets larger and the number of sets gets smaller
 - Last set contains entire list
- ◆ Items contained in set are not necessarily contiguous
 - If there are i sets then each set is composed of every i -th element

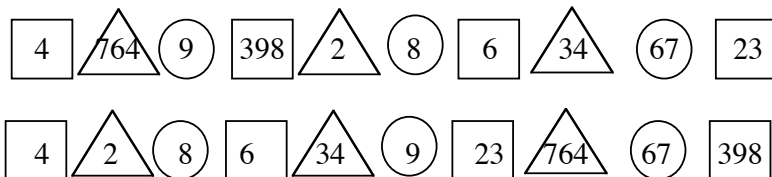
12/5/04

Sorting

14

Shell Sort

- ◆ For $i = 3$



12/5/04

Sorting

15

Shell Sort

- ◆ Increment sequence
 - $h_1, h_2, h_3, \dots, h_k$ for $h_1=1$
 - Each is called a phase
- ◆ After each phase, all elements spaced h_k apart are sorted
- ◆ $h_k = N/2$ $h_{k+1} = h_k/2$ for each pass k

12/5/04

Sorting

16

Shell Sort

	0	1	2	3	4	5	6	7	8
Pass	81	94	11	96	12	35	17	95	28

12/5/04

Sorting

17

Shell Sort

	0	1	2	3	4	5	6	7	8
Pass	81	94	11	96	12	35	17	95	28
4 - sort	12	35	11	95	28	94	17	96	81
2 - sort	11	35	12	94	17	95	28	96	81
1 - sort	11	12	17	28	35	81	94	95	96

Running time - $O(n^2)$

12/5/04

Sorting

18

Merge Sort

- ◆ Divide and conquer
- ◆ Merge sets of sorted lists
- ◆ Recursive sorting algorithm
 - Divide array in half then merge sort each half

12/5/04

Sorting

19

Merge Sort - Merge Step

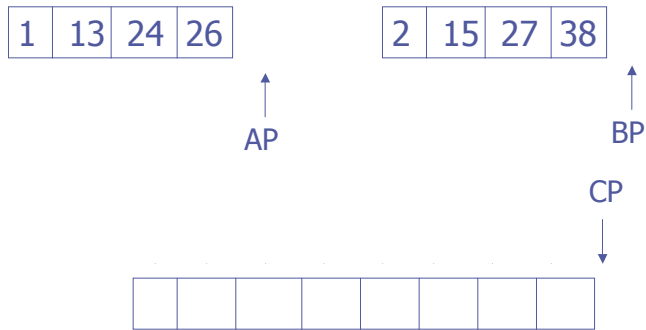
- ◆ Takes 2 sorted arrays and merges them to 1 sorted array
 - Input = 2 arrays A and B
 - Output = array C
 - Three pointers - AP, BP, and CP
 - ◆ Initially set to the beginning of each array
 - $C[CP] = \min (A[AP], B[BP])$
 - Running time $O(n)$

12/5/04

Sorting

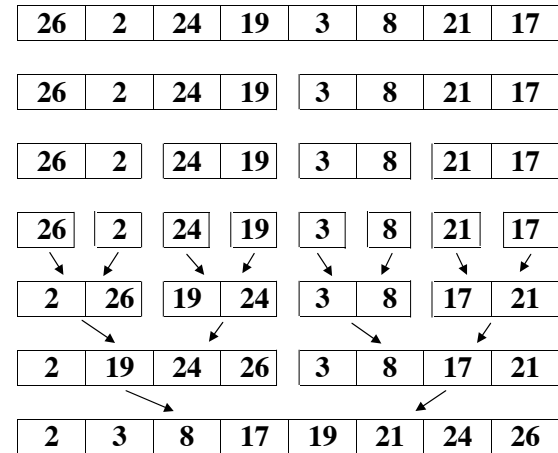
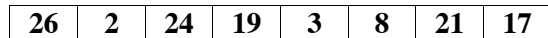
20

Merge Sort - Merge Step



Merge Sort

- ◆ Recursively sort left half
- ◆ Recursively sort right half
- ◆ Merge two sorted arrays



Running time - $O(n \lg n)$ regardless of the initial order

Quick Sort - Partition Step

Pivot	S1			S2	Unknown		
5	3	2	4	6	1	3	7

12/5/04

Sorting

29

Quick Sort - Partition Step

Pivot	S1			S2	Unknown		
5	3	2	4	6	1	3	7

Pivot	S1				S2	Unknown		
5	3	2	4	1	6	3	7	

Pivot	S1					S2	Unknown
5	3	2	4	1	3	6	7

Pivot	S1						S2
5	3	2	4	1	3	6	7

12/5/04

Sorting

30

Quick Sort - Partition Step

(only show this step in homework)

S1					Pivot	S2		
3	2	4	1	3	5	6	7	

Now perform Qsort of S1 and S2

Running Time - Worst case $O(n^2)$

Average case $O(n \lg n)$

12/5/04

Sorting

31

Quick Sort

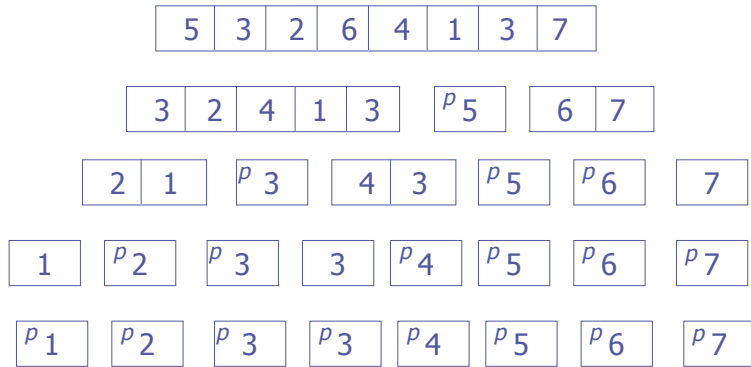
5	3	2	6	4	1	3	7
---	---	---	---	---	---	---	---

12/5/04

Sorting

32

Quick Sort



12/5/04

Sorting

33

Radix Sort

- ◆ Forming groups and then combining them to sort a collection of data
- ◆ Sort based on value at location one, the value at location two, and so on
- ◆ Example
 - Sorting words alphabetically
 - Sorting numbers - treat numbers like string padded with zeros

12/5/04

Sorting

34

Radix Sort

64, 8, 217, 512, 27, 728, 40, 1, 343, 125, 313

Pass	0	1	2	3	4	5	6	7	8	9
P=1										
P=2										
P=3										

12/5/04

Sorting

35

Radix Sort

64, 8, 217, 512, 27, 728, 40, 1, 343, 125, 313

Pass	0	1	2	3	4	5	6	7	8	9
P=1	40	1	512	343 313	64	125		217 27	8 728	
P=2	1 8	512 313 217	125 27 728		40 343		64			
P=3	1 8 27 40 64	125	217	313 343		512		728		

Running time $O(N)$ – with a very large constant

12/5/04

Sorting

36

Sorting Stability

- ◆ Duplicate numbers appear in the same order in the output as they did in the input